Towards Efficient MPPI Trajectory Generation with Unscented Guidance: U-MPPI Control Strategy

Ihab S. Mohamed[1], Junhong Xu[1], Gaurav S. Sukhatme[2], and Lantao Liu[1]

Abstract—The classical Model Predictive Path Integral (MPPI) control framework, while effective in many applications, lacks reliable safety features due to its reliance on a *risk-neutral* trajectory evaluation technique, which can present challenges for safety-critical applications such as autonomous driving. Furthermore, when the majority of MPPI sampled trajectories concentrate in high-cost regions, it may generate an *infeasible* control sequence. To address this challenge, we propose the U-MPPI control strategy, a novel methodology that can effectively manage system uncertainties while integrating a more efficient trajectory sampling strategy. The core concept is to leverage the Unscented Transform (UT) to propagate not only the mean but also the covariance of the system dynamics, going beyond the traditional MPPI method. As a result, it introduces a novel and more efficient trajectory sampling strategy, significantly enhancing state-space exploration and ultimately reducing the risk of being trapped in local minima. Furthermore, by leveraging the uncertainty information provided by UT, we incorporate a *risk-sensitive* cost function that explicitly accounts for risk or uncertainty throughout the trajectory evaluation process, resulting in a more resilient control system capable of handling uncertain conditions. By conducting extensive simulations of 2D aggressive autonomous navigation in both known and unknown cluttered environments, we verify the efficiency and robustness of our proposed U-MPPI control strategy compared to the baseline MPPI. We further validate the practicality of U-MPPI through real-world demonstrations in unknown cluttered environments, showcasing its superior ability to incorporate both the UT and local costmap into the optimization problem without introducing additional complexity.

Index Terms—Autonomous vehicle navigation, MPPI, unscented transform, occupancy grid map path planning.

Supplementary Material

A video showcasing both the simulation and real-world results, including the behavior of the vanilla MPPI, is provided in the supplementary video: https://youtu.be/1xsh4BxIrng
Moreover, the GPU implementation of our proposed U-MPPI algorithm is available at: https://github.com/IhabMohamed/U-MPPI

I. Introduction and Related Work

PLANNING and controlling autonomous vehicles under uncertainty is a partially-solved yet highly challenging problem in robotics. The uncertainty or risk can arise from

[1]Ihab S. Mohamed, Junhong Xu, and Lantao Liu are with the Luddy School of Informatics, Computing, and Engineering, Indiana University, Bloomington, IN 47408 USA (e-mail: {mohamedi, xu14, lantao}@iu.edu)
[2]Gaurav S. Sukhatme is with the Department of Computer Science, University of Southern California, Los Angeles, CA 90089, USA (e-mail: gaurav@usc.edu)
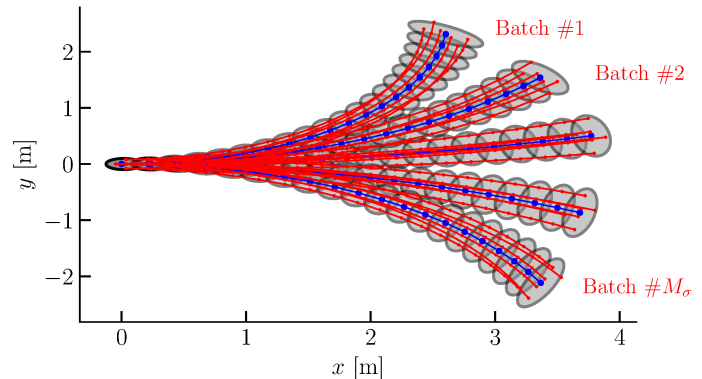
Fig. 1: Our proposed sampling strategy, for a ground vehicle model, under the U-MPPI control strategy based on unscented transform; such a sampling strategy propagates both the mean $\bar{\mathbf{x}}_k$ (blue dots) and covariance $\boldsymbol{\Sigma}_k$ (gray ellipses) of the state vector at each time-step k; to generate M sampled trajectories, we propagate M_σ sets of batches, where each batch contains n_σ trajectories corresponding to the n_σ sigma points, where $M = n_\sigma M_\sigma$, $n_\sigma = 2n_x + 1$, and red lines refer to $2n_x$ sigma-point trajectories surrounding the nominal trajectories (blue lines); for our validation-used robot, $n_x = 3$.

multiple sources, such as the vehicle's dynamics, the dynamics of other objects in the environment, the presence of unexpected obstacles, and the accuracy of the sensors used to perceive the environment. These factors can lead to unpredictable vehicle behavior, making it challenging to anticipate its movements and actions. Therefore, it is crucial for motion planning and control algorithms to account for the uncertainties of the system's states, sensing, and control actions. This eventually enables autonomous vehicles to adapt to unexpected environmental changes and make reliable decisions in real-time [1].

Model Predictive Control (MPC), also referred to as *receding-horizon* optimal control has been introduced as an effective solution for improving system safety and managing random disturbances and uncertainties, owing to its flexibility and ability to handle system constraints and nonlinearities while optimizing system performance. It plans a sequence of optimal control inputs over a finite time-horizon by repeatedly solving the optimal control problem using the *receding-horizon* principle (i.e., optimal control solvers [2]), with the first control input applied to the system. There are two main categories of MPC frameworks that can handle various forms of uncertainty: Robust MPC (RMPC) and Stochastic MPC (SMPC). RMPC considers worst-case scenarios of uncertainty to ensure stability and constraint satisfaction, while SMPC takes into account expected costs and probabilistic constraints (e.g., chance constraints) to optimize performance under uncertain conditions. In RMPC, a *minimax* optimization problem is solved to ensure the optimal performance of the control

strategy under worst-case scenarios while guaranteeing the satisfaction of constraints for all possible uncertainties [3]. This method is known for generating the safest solutions among MPC categories due to its incorporation of worst-case scenarios into the optimization process. Nevertheless, its emphasis on minimizing worst-case objective functions often results in complex, intractable optimization problems, leading to overly conservative control actions and reduced system efficiency [4], [5].

On the other hand, despite the superior ability of SMPC to leverage the probabilistic nature of uncertainties, many SMPC approaches have performance limitations, including being tailored to specific forms of stochastic noise, requiring dynamics linearization to ensure a *real-time* performance and implementation of the control strategy, and employing chance constraints that may not accurately reflect the severity of constraint violations or potential accidents and can be computationally demanding to evaluate, particularly for complex or high-dimensional probability distributions [6]–[8]. Additionally, SMPC has a further limitation in that it relies on a *risk-neutral* expectation to predict future uncertain outcomes, which may not be reliable in the case of a tail-end probability event (i.e., a rare event with a low probability of occurring) actually happening [9]. To address the challenges posed by uncertainties, Risk-Sensitive MPC (RSMPC) approaches have gained traction in recent years, thanks to their ability to balance the benefits and drawbacks of robust and stochastic MPC methods. By integrating the concept of *risk measures* or *risk metrics* into the optimization problem, RSMPC can evaluate the impact of uncertainty and adjust responses accordingly to different levels of uncertainty [10]–[12].

The Model Predictive Path Integral (MPPI) framework, a type of SMPC method, has emerged as a promising control strategy for complex robotics systems with stochastic dynamics and uncertainties [13]–[17]. Such a method solves the stochastic optimal control problem in a *receding-horizon* control setting by: (i) leveraging Monte Carlo simulation to rollout real-time simulated trajectories propagated from the system dynamics, (ii) evaluating these trajectories, (iii) computing the optimal control sequence by taking the weighted average of the costs of the sampled trajectories, and (iv) applying the first control input to the system while using the remaining control sequence to warm-start the optimization in the next time-step, enabling the method to solve the optimization problem effectively [13], [18]. MPPI stands out among alternative MPC methods due to its attractive features, such as being a sampling-based and derivative-free optimization method, not relying on assumptions or approximations of objective functions and system dynamics, being effective for highly dynamic systems, and benefiting from parallel sampling and the computational capabilities of Graphics Processing Units (GPUs) to achieve optimized and *real-time* performance [19].[1] Due to these attractive features, even the simplified Predictive Sampling method introduced in [20], which can variously

be described as MPPI with infinite temperature, can achieve competitive performance with more established gradient-based algorithms.

While MPPI has appealing characteristics, it may also pose challenges in practice. One particular concern is that, much like any sampling-based optimization algorithm, it could generate an *infeasible* control sequence if all the resulting MPPI sampled trajectories are concentrated in a high-cost region, which may lead to violations of system constraints or a higher likelihood of being trapped in local minima [21], [22]. In [23], Tube-MPPI was proposed as a solution to alleviate the situation by incorporating an iterative Linear Quadratic Gaussian (iLQG) controller (which, unfortunately, requires the linearization of dynamics) as an ancillary controller to track the MPPI-generated nominal trajectory. Similarly, in [16], an augmented version of MPPI is employed, which includes a nonlinear $\mathcal{L}_1$ adaptive controller to address model uncertainty. In [24], the authors propose the GP-MPPI control strategy to ensure efficient and safe navigation in unknown and complex environments while avoiding local minima. This strategy combines MPPI with a Sparse Gaussian Process (SGP)-based local perception model, incorporating online learning to effectively explore the surrounding navigable space. Meanwhile, in [25], the authors introduce Projection Guided Sampling Based Optimization (PRIEST), a new optimizer designed to push the infeasible sampled trajectories of MPPI towards feasible regions, thereby reducing the risk of local minima and infeasible control sequences.

Recently, various sampling techniques have been introduced to enhance the performance of MPPI [21], [22], [26]–[31]. To name a few examples, [21] enhances the MPPI algorithm by incorporating the covariance steering (CS) principle, while [22] proposes sampling trajectories from a product of normal and log-normal distributions (NLN mixture), instead of solely using a Gaussian distribution. In [26], a novel method embeds safety filters during forward sampling to ensure the safe operation of robotic systems within multiple constraints. Additionally, [27] improves MPPI performance by integrating adaptive importance sampling into the original importance sampling method, while [30] proposes a new importance sampling scheme that combines classical and learning-based ancillary controllers to improve efficiency. These methods result in more efficient trajectories than the vanilla MPPI, leading to better exploration of the system's state-space and reducing the risk of encountering local minima. [31] presents an algorithm, called Shield-MPPI, that enhances the robustness and computational efficiency of MPPI using Control Barrier Functions (CBFs) to ensure safety and reduce the need for extensive computational resources, making it suitable for real-time planning and deployment on hardware without GPUs.

Another constraint of MPPI is its inability to explicitly incorporate risk levels during planning due to its *risk-neutral* technique in evaluating sampled trajectories during the optimization process, making it challenging to achieve the desired balance between risk and robustness. Additionally, the MPPI optimization problem concentrates solely on minimizing the objective function, which is influenced by a minor perturbation injected into the control input, without explicitly considering

---

[1]It is worth noting that a CPU-based MPPI, optimized using vectorization and tensor operations, is now available. See the link for further information: https://github.com/artofnothingness/mppic

any uncertainties or risks associated with the system dynamics or the environment. This can eventually lead to sub-optimal or overly aggressive control actions, as MPPI may select trajectories that appear to have a low expected cost but may actually be riskier or less robust in practice. Consequently, MPPI cannot guarantee safety when environmental conditions change, which limits its applicability for safety-critical applications such as autonomous driving. [32] introduces a general framework for optimizing Conditional Value-at-Risk (CVaR) under uncertain dynamics, parameters, and initial conditions by incorporating additional samples from sources of uncertainty. The Risk-aware MPPI (RA-MPPI) algorithm [33] is a more recent approach that addresses this issue by utilizing CVaR to generate risk-averse controls that evaluate real-time risks and account for systematic uncertainties. However, such a method employs Monte-Carlo sampling to estimate the CVaR, which can be computationally intensive and time-consuming since generating a large number of random samples is required for accurate estimation using Monte Carlo methods.

In contrast to existing solutions aimed at mitigating the shortcomings of MPPI, the aim of our proposed solution is to effectively consider the uncertainties of system states, sensing, and control actions, in addition to integrating a novel and more efficient trajectory sampling strategy. To this end, we introduce the U-MPPI control strategy, a novel methodology that enhances the classical MPPI algorithm by combining the Unscented Transform (UT) with standard optimal control theory (also known as unscented guidance [34]) to effectively manage system uncertainties and improve trajectory sampling efficiency. Such a control strategy leverages the UT for two purposes: regulating the propagation of the dynamical system and proposing a new state-dependent cost function formulation that incorporates uncertainty information. To the best of the authors' knowledge, the proposed control strategy has not been previously explored in the literature, presenting a unique integration of the UT with the MPPI control framework. In summary, the contributions of this work can be summarized as follows:

1) While vanilla MPPI variants propagate only the mean value of the system dynamics, we propose a novel trajectory sampling technique utilizing the UT to propagate both the mean and covariance of the system dynamics, significantly enhancing state-space exploration and reducing the risk of local minima.

2) Then, by utilizing the propagated uncertainty information (i.e., state covariance matrix), we introduce a *risk-sensitive* cost function that explicitly considers risk or uncertainty during the trajectory evaluation process, leading to a safer and more robust control system, especially for safety-critical applications.

3) We provide a detailed analysis of the risk-sensitive function and its derivation, offering an in-depth understanding of the U-MPPI control strategy's behavior under different risk conditions, which elucidates how it adapts and performs in various scenarios, thereby enhancing its robustness and safety in uncertain environments.

Through extensive simulations and real-world experiments, we demonstrate the effectiveness of our U-MPPI control strategy for aggressive collision-free navigation in both known and unknown cluttered environments, highlighting its superiority over the baseline MPPI control framework by producing more efficient trajectories, achieving higher success and task completion rates, reducing the risk of local minima, and ultimately enabling the robot to find feasible collision-free trajectories.

The rest of the paper is organized as follows. Section II addresses the problem statement of stochastic optimal control and provides an overview of the vanilla MPPI control strategy. Section III introduces the unscented transform and its application in generating robust control strategies by combining it with standard optimal control theory to manage system uncertainties. Section IV details the proposed U-MPPI control strategy, including the unscented-based sampling strategy, the *risk-sensitive* cost function, and its impact on the controller's behavior, and explains the real-time control algorithm. Following this, Section V presents simulation results comparing the performance of the U-MPPI control strategy with the baseline MPPI in various scenarios, highlighting improvements in trajectory generation and robustness. Subsequently, Section VI demonstrates the practicality of the U-MPPI control strategy through real-world experiments in unknown cluttered environments, showcasing its effectiveness and adaptability. Finally, Section VII concludes the study and proposes potential future research directions.

II. STOCHASTIC OPTIMAL CONTROL

This section aims to establish the problem statement of stochastic optimal control and present a concise overview of MPPI as a potential solution to address this problem.

A. Problem Formulation

Within the context of discrete-time stochastic systems, let us consider the system state $\mathbf{x}_k \in \mathbb{R}^{n_x}$, the control input $\mathbf{u}_k \in \mathbb{R}^{n_u}$, and the underlying non-linear dynamics

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{w}_k). \tag{1}$$

The actual (i.e., disturbed) control input, $\mathbf{w}_k$, is represented as $\mathbf{w}_k = \mathbf{u}_k + \delta\mathbf{u}_k \sim \mathcal{N}(\mathbf{u}_k, \Sigma_{\mathbf{u}})$, where $\delta\mathbf{u}_k \sim \mathcal{N}(\mathbf{0}, \Sigma_{\mathbf{u}})$ is a zero-mean Gaussian noise with covariance $\Sigma_{\mathbf{u}}$ which represents the injected disturbance into the control input. Within a finite time-horizon N, we denote the control sequence $\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}]^\top \in \mathbb{R}^{n_u N}$ and the corresponding state trajectory $\mathbf{x} = [\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_N]^\top \in \mathbb{R}^{n_x(N+1)}$. Moreover, let $\mathcal{X}^d$ denote the d dimensional space with $\mathcal{X}_{rob}(\mathbf{x}_k) \subset \mathcal{X}^d$ and $\mathcal{X}_{obs} \subset \mathcal{X}^d$ represent the area occupied by the robot and obstacles, respectively. In this scenario, the objective of the stochastic optimal control problem is to find the optimal control sequence, $\mathbf{U}$, that generates a collision-free trajectory, guiding the robot from its initial state, $\mathbf{x}_s$, to the desired state, $\mathbf{x}_f$, under the minimization of the cost function, J, subject to

specified constraints. The optimization problem at hand can be formulated using the vanilla MPPI control strategy as

$$\min_{\mathbf{U}} \quad J(\mathbf{x}, \mathbf{u}) = \mathbb{E}\left[\phi(\mathbf{x}_N) + \sum_{k=0}^{N-1}\left(q(\mathbf{x}_k) + \frac{1}{2}\mathbf{u}_k^\top R\mathbf{u}_k\right)\right], \quad (2a)$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{w}_k), \delta\mathbf{u}_k \sim \mathcal{N}(\mathbf{0}, \Sigma_\mathbf{u}), \quad (2b)$$

$$\mathcal{X}_{rob}(\mathbf{x}_k) \cap \mathcal{X}_{obs} = \emptyset, \ h(\mathbf{x}_k, \mathbf{u}_k) \leq 0, \quad (2c)$$

$$\mathbf{x}_0 = \mathbf{x}_s, \ \mathbf{u}_k \in \mathbb{U}, \ \mathbf{x}_k \in \mathbb{X}, \quad (2d)$$

where $R \in \mathbb{R}^{n_u \times n_u}$ is a positive-definite control weighting matrix, $\mathbb{U}$ denotes the set of admissible control inputs, and $\mathbb{X}$ denotes the set of all possible states $\mathbf{x}_k$; the state terminal cost function, $\phi(\mathbf{x}_N)$, and the running cost function, $q(\mathbf{x}_k)$, can be defined as arbitrary functions, offering a more flexible and dynamic approach to cost modeling that can be adapted to meet the specific requirements of the system being controlled. The function $h(\mathbf{x}_k, \mathbf{u}_k)$ represents the state and control constraints that need to be satisfied at each time step. For instance, $h(\mathbf{x}_k, \mathbf{u}_k)$ can specify speed limits, actuator constraints, or state constraints such as position or orientation bounds. These constraints are met when $h(\mathbf{x}_k, \mathbf{u}_k) \leq 0$, ensuring that the system operates within safe and feasible regions. Additionally, to handle collision avoidance, the constraint $\mathcal{X}_{\text{rob}}(\mathbf{x}_k) \cap \mathcal{X}_{\text{obs}} = \emptyset$ ensures that the robot does not collide with any obstacles.

B. Overview of MPPI Control Strategy

MPPI solves the optimization problem defined in (2) by minimizing the objective function J (2a), taking into account the system dynamics (2b) and constraints, including collision avoidance and control constraints, detailed in (2c). To this end, at each control loop interval Δt, MPPI employs the Monte Carlo simulation to sample thousands of *real-time* simulated trajectories, represented by M, propagated from the underlying system dynamics, as illustrated in Fig. 2. Subsequently, within the time-horizon N, the *cost-to-go* of each trajectory τ_m can be evaluated as

$$\tilde{S}(\tau_m) = \phi(\mathbf{x}_N) + \sum_{k=0}^{N-1}\tilde{q}(\mathbf{x}_k, \mathbf{u}_k, \delta\mathbf{u}_{k,m}), \forall m \in \{1, \cdots, M\}, \quad (3)$$

where $\phi(\mathbf{x}_N)$ refer to the terminal state cost, while the instantaneous running cost $\tilde{q}(\mathbf{x}_k, \mathbf{u}_k, \delta\mathbf{u}_k)$ encompasses both the state-dependent running cost $q(\mathbf{x}_k)$ and the quadratic control cost $q_\mathbf{u}(\mathbf{u}_k, \delta\mathbf{u}_k)$ and is formulated as

$$\tilde{q} = \underbrace{q(\mathbf{x}_k)}_{\textit{State-dep.}} + \underbrace{\gamma_\mathbf{u}\delta\mathbf{u}_{k,m}^\top R\delta\mathbf{u}_{k,m} + \mathbf{u}_k^\top R\delta\mathbf{u}_{k,m} + \frac{1}{2}\mathbf{u}_k^\top R\mathbf{u}_k}_{q_\mathbf{u}(\mathbf{u}_k, \delta\mathbf{u}_k): \textit{ Quadratic Control Cost}}, \quad (4)$$

where $\gamma_\mathbf{u} = \frac{\nu-1}{2\nu} \in \mathbb{R}_{\geq 0}$, and $\nu \geq 1$ determines the level of aggressiveness in exploring the state-space.

As stated in [13], the vanilla MPPI algorithm updates the optimal control sequence $\{\mathbf{u}_k\}_{k=0}^{N-1}$ by considering a weighted average cost from all of the simulated trajectories; mathematically, this control sequence can be expressed as

$$\mathbf{u}_k \leftarrow \mathbf{u}_k + \frac{\sum_{m=1}^{M}\exp\left(\frac{-1}{\lambda}\left[\tilde{S}(\tau_m) - \tilde{S}_{\min}\right]\right)\delta\mathbf{u}_{k,m}}{\sum_{m=1}^{M}\exp\left(\frac{-1}{\lambda}\left[\tilde{S}(\tau_m) - \tilde{S}_{\min}\right]\right)}, \quad (5)$$

which incorporates the *cost-to-go* of the mth trajectory $\tilde{S}(\tau_m)$, the minimum cost trajectory among all simulated rollouts $\tilde{S}_{\min}$ that prevents numerical overflow or underflow, without affecting the optimality of the algorithm, and the inverse temperature λ that governs the selectiveness of the weighted average of trajectories. The resulting sequence is then smoothed with a Savitzky-Galoy filter [35], followed by applying the first control $\mathbf{u}_0$ to the system, while the remaining sequence is utilized for warm-starting the next optimization step.

III. Unscented Optimal Control

In this section, we present a comprehensive overview of the unscented transform, followed by the unscented guidance technique, which combines the unscented transform with standard stochastic optimal control theory to generate a robust *open-loop* method for managing system uncertainties.

A. Unscented Transform

The Unscented Transform (UT), proposed by Julier and Uhlmann [36], aims to create nonlinear filters without the need for linearization. It approximates a probability distribution function (PDF) after it passes through a non-linear transformation using a set of sampled points, known as sigma points [7], [37].

Formally, given the mean $\bar{\mathbf{x}}_k$ and covariance $\mathbf{\Sigma}_k$ of a Gaussian-distributed system state $\mathbf{x}_k$, with $\mathbf{x}_k \sim \mathcal{N}(\bar{\mathbf{x}}_k, \mathbf{\Sigma}_k) \in \mathbb{R}^{n_x}$, UT approximates the distribution over the next state, $\mathbf{x}_{k+1}$, by first introducing a set of sigma points $\{\mathcal{X}_k^{(i)}\}_{i=0}^{2n_x} \in \mathbb{R}^{n_\sigma}$ around the mean $\bar{\mathbf{x}}_k$ and the corresponding weights $\{w^{(i)}\}_{i=0}^{2n_x} \in \mathbb{R}^{n_\sigma}$, where the number of sigma points is given by $n_\sigma = 2n_x + 1$ and n_x is the dimension of the state space. These sigma points are designed to capture the covariance of the distribution at time-step k as follows

$$\begin{aligned}
\mathcal{X}_k^{(0)} &= \bar{\mathbf{x}}_k, \\
\mathcal{X}_k^{(i)} &= \bar{\mathbf{x}}_k + \left(\sqrt{(n_x + \lambda_\sigma)\mathbf{\Sigma}_k}\right)_i, \ \forall i = \{1, \dots, n_x\}, \\
\mathcal{X}_k^{(i)} &= \bar{\mathbf{x}}_k - \left(\sqrt{(n_x + \lambda_\sigma)\mathbf{\Sigma}_k}\right)_i, \ \forall i = \{n_x + 1, \dots, 2n_x\},
\end{aligned} \quad (6)$$

where $\left(\sqrt{(n_x + \lambda_\sigma)\mathbf{\Sigma}_k}\right)_i$ represents the ith row or column of the square root of the weighted covariance matrix $(n_x + \lambda_\sigma)\mathbf{\Sigma}_k$, and $\lambda_\sigma = \alpha^2(n_x + k_\sigma) - n_x$ is influenced by the scaling parameters $k_\sigma \geq 0$ and $\alpha \in (0, 1]$ that determine how far the sigma points are spread from the mean [38], as demonstrated in Fig. 4. Each $\mathcal{X}_k^{(i)}$ is associated with two weights, $w_m^{(i)}$ for computing the mean and $w_c^{(i)}$ for determining the covariance of the transformed distribution, computed as

$$\begin{aligned}
w_m^{(0)} &= \frac{\lambda_\sigma}{n_x + \lambda_\sigma}, \\
w_c^{(0)} &= w_m^{(0)} + (1 - \alpha^2 + \beta), \\
w_m^{(i)} &= w_c^{(i)} = \frac{1}{2(n_x + \lambda_\sigma)}, \ \forall i = \{1, \dots, 2n_x\},
\end{aligned} \quad (7)$$

where β is a hyper-parameter controlling the relative importance of the mean and covariance information. In other words, β is employed to incorporate prior knowledge about the distribution of state $\mathbf{x}$. For Gaussian distributions, the

optimal value for β is 2 [39]. In the second step, we propagate the $(2n_x + 1)$ sigma points through the non-linear system to produce the transformed sigma points $\mathcal{X}_{k+1}^{(i)} = f(\mathcal{X}_k^{(i)})$ at the next time-step. Finally, the mean and covariance of $\mathbf{x}_{k+1}$ can be estimated using the transformed sigma points and their corresponding weights, as follows

$$
\begin{aligned}
\bar{\mathbf{x}}_{k+1} &= \sum_{i=0}^{2n_x} w_m^{(i)} \mathcal{X}_{k+1}^{(i)}, \\
\mathbf{\Sigma}_{k+1} &= \sum_{i=0}^{2n_x} w_c^{(i)} (\mathcal{X}_{k+1}^{(i)} - \bar{\mathbf{x}}_{k+1})(\mathcal{X}_{k+1}^{(i)} - \bar{\mathbf{x}}_{k+1})^\top.
\end{aligned}
\tag{8}
$$

B. Unscented Optimal Control

By incorporating the unscented transform with standard optimal control, the unscented optimal control, also referred to as unscented guidance [34], presents a novel methodology for addressing the uncertainties in non-linear dynamical systems within an *open-loop* framework [40], [41]. Given the sigma points $\mathcal{X}_k^{(i)}$ and disturbed control input $\mathbf{w}_k$ at time-step k, each sigma point can be propagated through the underlying non-linear dynamics given in (1), as follows

$$
\mathcal{X}_{k+1}^{(i)} = f\left(\mathcal{X}_k^{(i)}, \mathbf{w}_k\right), \forall i = 0, \cdots, 2n_x \in \mathbb{R}^{n_\sigma}. \tag{9}
$$

Consider an $n_\sigma n_x$-dimensional vector $\mathbf{X}$, defined as $\mathbf{X} = \left[\mathcal{X}^{(0)}, \mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(2n_x)}\right]^\top \in \mathbb{R}^{n_x n_\sigma}$. Then, the dynamics of $\mathbf{X}$ are characterized by n_σ instances of the function f, specified as

$$
\boldsymbol{X}_{k+1} = \begin{bmatrix} f\left(\mathcal{X}_k^{(0)}, \mathbf{w}_k\right) \\ f\left(\mathcal{X}_k^{(1)}, \mathbf{w}_k\right) \\ \vdots \\ f\left(\mathcal{X}_k^{(2n_x)}, \mathbf{w}_k\right) \end{bmatrix} := \mathbf{f}(\boldsymbol{X}_k, \mathbf{w}_k). \tag{10}
$$

With these preliminaries, the original stochastic optimal control problem described in (2) can be re-formulated within the context of the unscented guidance framework as

$$
\min_{\mathbf{U}} \mathbf{J}(\mathbf{X}, \mathbf{u}) = \mathbb{E}\left[\boldsymbol{\Phi}(\mathbf{X}_N) + \sum_{k=0}^{N-1}\left(\mathbf{q}(\mathbf{X}_k) + \frac{1}{2}\mathbf{u}_k^\top R\mathbf{u}_k\right)\right], \tag{11a}
$$

$$
\text{s.t.} \quad \mathbf{X}_{k+1} = \mathbf{f}\left(\mathbf{X}_k, \mathbf{w}_k\right), \delta\mathbf{u}_k \sim \mathcal{N}(\mathbf{0}, \Sigma_{\mathbf{u}}), \tag{11b}
$$

$$
\mathcal{X}_{rob}\left(\mathbf{X}_k\right) \cap \mathcal{X}_{obs} = \emptyset, \ \mathbf{h}(\mathbf{X}_k, \mathbf{u}_k) \leq 0, \tag{11c}
$$

$$
\mathbf{X}_0 = \left[\mathcal{X}_0^{(0)}, \ldots, \mathcal{X}_0^{(2n_x)}\right]^\top, \mathbf{u}_k \in \mathbb{U}, \mathbf{X}_k \in \mathbb{X}, \tag{11d}
$$

where $\boldsymbol{\Phi}\left(\mathbf{X}_N\right) = \left[\phi\left(\mathcal{X}_N^{(0)}\right), \ldots, \phi\left(\mathcal{X}_N^{(2n_x)}\right)\right]^\top \in \mathbb{R}^{n_\sigma}$, and $\mathbf{q}\left(\mathbf{X}_k\right) = \left[q\left(\mathcal{X}_k^{(0)}\right), \ldots, q\left(\mathcal{X}_k^{(2n_x)}\right)\right]^\top \in \mathbb{R}^{n_\sigma}$. The objective of our proposed U-MPPI control strategy is to minimize the objective function, $\mathbf{J}$, in (11a) by finding the optimal control sequence, $\mathbf{U} = \{\mathbf{u}_k\}_{k=0}^{N-1}$, while taking into account (i) the system constraints previously discussed in Section II-A, and (ii) the uncertainties associated with both system states and control actions. In U-MPPI, minimizing the vector-valued cost function $\mathbf{J}(\mathbf{X}, \mathbf{u})$, which represents the costs of sigma-point trajectories, is achieved through a sampling-based stochastic
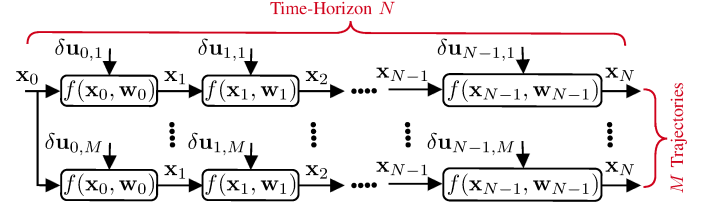

Fig. 2: Schematic illustration of system dynamics propagation in MPPI for M sampled trajectories over a finite time-horizon N.

optimization approach similar to MPPI by: (i) leveraging the UT to propagate the sigma points, (ii) evaluating their trajectory costs using a risk-sensitive formulation, and (iii) updating the control sequence through a weighted sampling-based optimization, as detailed in the next section.

IV. U-MPPI CONTROL STRATEGY

As previously outlined in Section II-B, the control noise variance $\Sigma_{\mathbf{u}}$ is not updated by MPPI, and the state-space exploration is performed by adjusting ν (refer to (4)). Nevertheless, a too-high value of ν can cause control inputs with considerable chatter. Similarly, increasing $\Sigma_{\mathbf{u}}$ might violate system constraints and eventual divergence from the desired state [22]. Additionally, the MPPI problem, as stated in (2), is focused solely on minimizing the cost function that is affected by a minor perturbation in the control input, represented by $\delta\mathbf{u}_k$, without explicitly incorporating the uncertainties that may be associated with either the system states or the surrounding environment. To ensure effectiveness in practice, the motion control strategy should be able to reflect the uncertainties of system states, sensing, and control actions. To this end, we introduce the U-MPPI control strategy, a new technique that leverages unscented transform to deal with these uncertainties. More precisely, the UT is utilized for the purpose of regulating the propagation of the dynamical system, thereby introducing a novel and more efficient trajectory sampling strategy than the standard MPPI variants, as demonstrated in Section IV-A. Furthermore, as discussed in Section IV-B, UT proposes a new cost function formulation incorporating uncertainty information, leading to a safer and more robust control system, especially for safety-critical applications.

A. Unscented-Based Sampling Strategy

To gain a comprehensive understanding of our proposed unscented-based sampling strategy, it is essential to first examine how the system dynamics are propagated within the vanilla MPPI control strategy. As illustrated in Fig. 2, given the current robot state $\mathbf{x}_0$ and an $N \times M$ matrix of random control variations $\delta\mathbf{u}$ generated on a GPU using CUDA's random number generation library, the MPPI algorithm optimizes control inputs by propagating the system dynamics over a specified time-horizon N for M trajectories. To be more precise, for a particular trajectory $m \in \{1, \cdots, M\}$ (represented by each row), the process involves generating a control input sequence over the time horizon N by perturbing the nominal inputs $\{\mathbf{u}_k\}_{k=0}^{N-1}$, which represent the optimal control sequence provided by MPPI during the previous control loop interval Δt
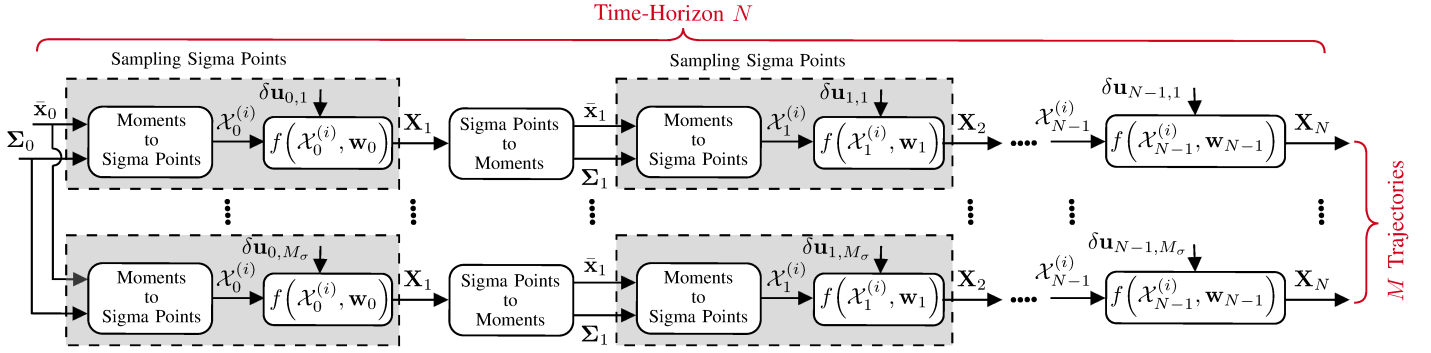
Fig. 3: Schematic illustration of nonlinear dynamical system propagation under the proposed U-MPPI control strategy for M sampled trajectories over a finite time-horizon N, where $M = n_\sigma M_\sigma$.

as previously explained in Section II-B, with $\{\delta\mathbf{u}_{k,m}\}_{k=0}^{N-1}$. At each discrete time-step k, the next state $\mathbf{x}_{k+1}$ is computed using the system's dynamic model f as defined in (1), such that $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k + \delta\mathbf{u}_{k,m})$. This results in a sequence of states $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \equiv \{\mathbf{x}_k\}_{k=1}^N$. This process is repeated for multiple trajectories by varying the perturbations $\{\delta\mathbf{u}_{k,m}\}_{k=0}^{N-1}$, resulting in M trajectories. Each trajectory is then evaluated using a predefined cost function, incorporating factors such as target deviation and control effort. It is important to notice that, as illustrated in the schematic diagram, the vanilla MPPI algorithm leverages Monte Carlo simulation to simulate a large number of real-time trajectories M by solely manipulating the injected Gaussian noise $\delta\mathbf{u}$ into the mean control sequence $\mathbf{U}$. Moreover, within the time horizon N, only the first moment (i.e., mean) of the propagated states is computed given the initial state $\mathbf{x}_0$, without propagating the covariance of $\mathbf{x}_k$. To enhance the performance of the classic MPPI algorithm, we propose a new trajectory sampling technique that utilizes the UT to propagate both the mean $\bar{\mathbf{x}}_k$ and covariance $\mathbf{\Sigma}_k$ of the state vector $\mathbf{x}_k$ at each discrete time-step k.

In the proposed U-MPPI control strategy, as displayed in Fig. 3, given the current robot state $\mathbf{x}_0 \sim \mathcal{N}(\bar{\mathbf{x}}_0, \mathbf{\Sigma}_0)$ and $N \times M_\sigma$ random control perturbations, the system dynamics are propagated over the entire time-horizon N by converting the moments of the state distribution into sigma points at each time-step k, then propagating these sigma points through the system's dynamic model f as expressed in (9), followed by converting the propagated sigma points back into moments to compute the updated mean state and covariance at $k + 1$. This methodology leads to a total of M rollouts, achieved by sampling M_σ sets of batches, also referred to as *cones*, with each batch comprising n_σ trajectories, such that $M = n_\sigma M_\sigma$ (each row in the figure represents a batch). For each batch $m \in \{1, \cdots, M_\sigma\}$, the propagation process of our proposed sampling strategy involves generating a control input sequence over the time-horizon N by perturbing the nominal inputs $\{\mathbf{u}_k\}_{k=0}^{N-1}$ with $\{\delta\mathbf{u}_{k,m}\}_{k=0}^{N-1}$, similar to the MPPI propagation process. First, the initial mean state $\bar{\mathbf{x}}_0$ and its covariance $\mathbf{\Sigma}_0$ are used to generate a set of sigma points $\{\mathcal{X}_0^{(i)}\}_{i=0}^{2n_x}$ via (6). Afterwards, at each subsequent time-step k, these sigma points $\{\mathcal{X}_k^{(i)}\}_{i=0}^{2n_x}$ are propagated through the nonlinear dynamics

expressed in (9) as $\mathcal{X}_{k+1}^{(i)} = f\left(\mathcal{X}_k^{(i)}, \mathbf{u}_k + \delta\mathbf{u}_{k,m}\right)$, along with the control inputs perturbed by Gaussian noise $\delta\mathbf{u}_{k,m}$. This results in a new set of propagated sigma points $\{\mathcal{X}_{k+1}^{(i)}\}_{i=0}^{2n_x}$, which can be merged into a single vector $\mathbf{X}_{k+1}$ using (10). The propagated sigma points are then converted back into moments by applying (8) to compute the updated mean state $\bar{\mathbf{x}}_{k+1}$ and covariance $\mathbf{\Sigma}_{k+1}$. This iterative process of "*Moments to Sigma Points*" and "*Sigma Points to Moments*" is repeated for each time-step k over the entire time-horizon N, ensuring accurate propagation of both the mean and covariance of the state distribution. This results in a sequence of state vectors $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_N\} \equiv \{\mathbf{X}_k\}_{k=1}^N$, representing the n_σ propagated sigma points. This sequence generation is performed for multiple trajectories M_σ by varying the perturbations $\delta\mathbf{u}_{k,m}$ for each trajectory m, resulting in a set of M possible state trajectories. In this study, we assume that during each time-step k, all sigma points belonging to the mth batch are influenced by the same Gaussian control noise, represented by $\delta\mathbf{u}_{k,m}$. Figure 1 provides a visual representation of our proposed unscented-based sampling strategy depicted in Fig. 3.

The primary objective of the new sampling strategy is to achieve significantly better exploration of the state-space of the given system and more efficient sampling of trajectories compared to MPPI, while utilizing the same injected control noise $\Sigma_\mathbf{u}$. To exemplify how leveraging the UT for trajectory sampling can enhance the performance of the MPPI algorithm and to investigate the effect of UT parameters on the distribution of sampled rollouts, we present a concrete example in Fig. 4. In particular, using the discrete-time kinematics model of a differential wheeled robot described in (17) and control schemes parameters listed in Section V-A1, we generate 210 rollouts by sampling $\delta\mathbf{u}_k$ from a zero-mean Gaussian distribution with a covariance of $0.025\mathbf{I}_2$ under the classical MPPI framework, as illustrated in Fig. 4(a), where $\mathbf{I}_n$ denotes an $n \times n$ identity matrix. Similarly, we employ the unscented-based sampling strategy to draw 210 trajectories, considering: (i) the same injected disturbance into the control input, i.e., $\delta\mathbf{u}_k \sim \mathcal{N}(\mathbf{0}, 0.025\mathbf{I}_2)$, and (ii) setting the UT scaling parameters to $\alpha = 1$ and $k_\sigma = 0$, along with an initial state covariance matrix $\mathbf{\Sigma}_0$ of $0.01\mathbf{I}_3$, as illustrated in Fig. 4(b). It is noteworthy to observe in Fig. 4(b) that our proposed sampling strategy is more efficient than
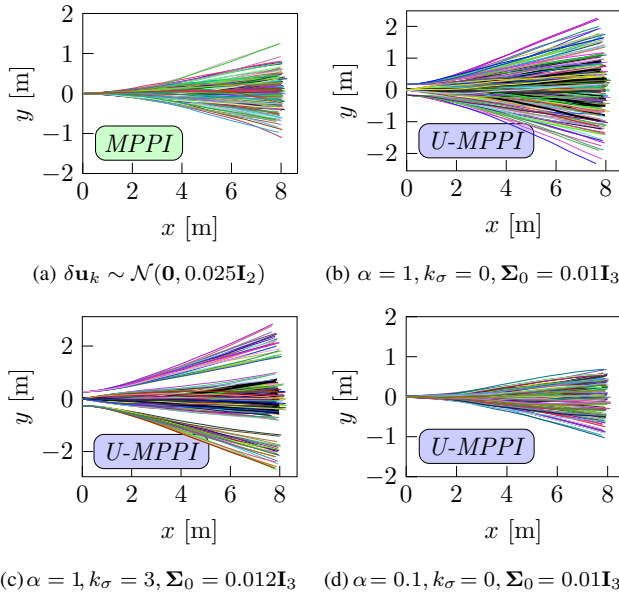
(a) $\delta\mathbf{u}_k \sim \mathcal{N}(\mathbf{0}, 0.025\mathbf{I}_2)$ (b) $\alpha = 1, k_\sigma = 0, \mathbf{\Sigma}_0 = 0.01\mathbf{I}_3$

(c) $\alpha = 1, k_\sigma = 3, \mathbf{\Sigma}_0 = 0.012\mathbf{I}_3$ (d) $\alpha = 0.1, k_\sigma = 0, \mathbf{\Sigma}_0 = 0.01\mathbf{I}_3$

Fig. 4: Distribution of 210 sampled trajectories generated by (a) MPPI with $\delta\mathbf{u}_k \sim \mathcal{N}(\mathbf{0}, 0.025\mathbf{I}_2)$ and (b) U-MPPI with the same perturbation in the control input $\delta\mathbf{u}_k$ but with different UT parameters; in both methods, the robot is assumed to be initially located at $\mathbf{x} = [x, y, \theta]^\top = [0, 0, 0]^\top$ in ([m], [m], [deg]), with a commanded control input $\mathbf{u} = [v, \omega]^\top = [1, 0]^\top$ in ([m/s], [rad/s]).

the classical MPPI sampling strategy, as it generates more spread-out trajectories that cover a larger state space. This enables the robot to explore the environment more extensively and find better solutions, thereby reducing the likelihood of getting trapped in local minima, as revealed in Section V-A4. Nevertheless, as depicted in Fig. 4(c), employing higher values of UT parameters (specifically, $\alpha, k_\sigma, \mathbf{\Sigma}_0$) may result in a loss of precision and continuity in the distribution of trajectories across the state-space, as the sigma points become more spread out from the mean $\mathcal{X}_k^{(0)}$. This can impact the resulting control actions of the system and, in the context of autonomous navigation, potentially lead to collisions with obstacles. In contrast, Fig. 4(d) demonstrates that using lower values of α and k_σ results in a sampling strategy that closely resembles MPPI. Similarly, if trajectories are sampled only from $\mathcal{X}_k^{(0)}$ (as depicted by the blue trajectories in Fig. 1), while excluding other sigma-point trajectories, the same sampling strategy can be achieved. We refer to this approach as sampling mode 0 (SM_0), while the default strategy that includes all sigma points is referred to as SM_1.

B. Risk-Sensitive Cost

One of the main limitations of the vanilla MPPI is that it typically assumes a *risk-neutral* approach when assessing the sampled trajectories during the optimization process, without explicitly considering risk or uncertainty in the trajectory evaluation process, as outlined in Section II-B, particularly in (4). The commonly employed method in sampling-based MPC algorithms involves using a quadratic cost function to guide the current state $\mathbf{x}_k$ towards its desired state $\mathbf{x}_f$, denoted as $q_{\mathrm{state}}(\mathbf{x}_k)$, and expressed mathematically as follows

$$q_{\mathrm{state}}(\mathbf{x}_k) = (\mathbf{x}_k - \mathbf{x}_f)^\top \mathrm{Q}\, (\mathbf{x}_k - \mathbf{x}_f) = \|\mathbf{x}_k - \mathbf{x}_f\|_{\mathrm{Q}}^2, \quad (12)$$

where Q is a positive definite weighting matrix. Hence, our key objective in this work is to integrate a risk measure into the U-MPPI optimization problem, enabling the adjustment of responses to varying uncertainty levels and improving adaptability over the conventional MPPI algorithm.

Whittle introduced in [42] an interesting risk measure method for incorporating risk in decision-making by replacing the expected quadratic cost with a risk-sensitive benchmark in the form of an exponential-quadratic function. Such a formulation maps uncertainties to a numerical value to assess the potential threat from extreme events. In the context of risk-sensitive control, this involves assessing the impact of uncertainty on system performance and adjusting responses to manage risks effectively. Mathematically, the *risk-sensitive* (RS) cost $q_{\mathrm{rs}}(\mathbf{x}_k)$ can be obtained by evaluating the log-expectation of the exponentiated quadratic cost as follows

$$
\begin{aligned}
q_{\mathrm{rs}}(\mathbf{x}_k) &= -\frac{2}{\gamma} \log \mathbb{E}\left[\exp\left(-\frac{1}{2}\gamma q_{\mathrm{state}}(\mathbf{x}_k)\right)\right] \\
&= -\frac{2}{\gamma} \log \mathbb{E}\left[\exp\left(-\frac{1}{2}\gamma \|\mathbf{x}_k - \mathbf{x}_f\|_{\mathrm{Q}}^2\right)\right].
\end{aligned}
\quad (13)
$$

Notably, as γ approaches zero, the RS cost $q_{\mathrm{rs}}(\mathbf{x}_k)$ converges to the standard quadratic cost $q_{\mathrm{state}}(\mathbf{x}_k)$, i.e.,[2]

$$\lim_{\gamma \to 0} q_{\mathrm{rs}}(\mathbf{x}_k) = \|\mathbf{x}_k - \mathbf{x}_f\|_{\mathrm{Q}}^2 = q_{\mathrm{state}}(\mathbf{x}_k),$$

thus avoiding the singularity issue when $\gamma = 0$. Here, γ is a real scalar denoted as the *risk-sensitivity* parameter, dictating how the controller reacts to risk or uncertainty. For example, when $\gamma > 0$, the controller exhibits *risk-seeking* or *risk-preferring* behavior, favoring trajectories that may offer higher rewards, even if they come with higher uncertainties or risks. As a result, a controller minimizing the *risk-seeking* cost q_{rs} will actively steer the system state $\mathbf{x}_k$ towards areas of uncertainty (i.e., uncertain regions) where the cost q_{rs} imposes lower penalties. Such behavior encourages the exploration of unknown or uncertain areas, which can lead to a more comprehensive understanding of the system dynamics. Conversely, when $\gamma < 0$, the controller demonstrates *risk-averse* or *risk-avoiding* behavior, prioritizing safer trajectories and leading to more conservative control actions. In this scenario, the controller minimizes *risk-averse* cost q_{rs} by driving the system state towards areas with minimal uncertainty and lower penalties. This approach prioritizes stability and safety by avoiding risky or uncertain areas, making it suitable for applications where minimizing risk is crucial. Examples include autonomous medical robots performing delicate surgeries, where precision and safety are crucial to prevent patient harm. It is noteworthy that a control strategy aimed at minimizing the maximum possible loss (i.e., solving a minimax optimization problem) aligns with the *risk-averse* behavior inherent in the risk-sensitive function, thereby representing a specific instance of risk-sensitive control designed to ensure robust and reliable performance. As discussed in [3] (see Chapter 8), the author examines the relationship between *risk-sensitive* and *minimax*

---

[2]This is because, as γ becomes very small, the exponential function can be approximated using a Taylor series expansion, and the logarithmic term can be simplified accordingly.

optimization problems in MPC, allowing us to assert that a controller designed to minimize the worst-case scenario (i.e., a minimax control approach) can be interpreted as minimizing a *risk-averse* performance measure (i.e., objective function). When $\gamma = 0$, the controller is considered *risk-neutral*, treating all uncertainties uniformly without preference towards risk. The risk-sensitivity parameter γ thus directly influences the optimization problem described in (11), determining whether the control policy will adopt *risk-seeking* or *risk-averse* behaviors based on the selected value of γ. We refer to [10] for more details and clear visualizations demonstrating the relationship between the sign of γ and the RS cost $q_{\rm rs}$.

Building on Whittle's foundational work, our proposed U-MPPI control strategy leverages this risk-sensitive approach to enhance the robustness and safety of autonomous vehicle navigation. By integrating the exponential-quadratic cost function $q_{\rm rs}$ into our trajectory evaluation process, we can effectively manage system uncertainties and enhance overall control efficiency. Our control strategy assumes that the system state $\mathbf{x}_k \sim \mathcal{N}(\bar{\mathbf{x}}_k, \boldsymbol{\Sigma}_k)$ follows Gaussian distribution. With this assumption, we can approximate the RS state-dependent cost expressed in (13) as

$$q_{\rm rs}(\mathbf{x}_k) = \frac{1}{\gamma} \log \det \left(\mathbf{I} + \gamma Q \boldsymbol{\Sigma}_k \right) + \|\bar{\mathbf{x}}_k - \mathbf{x}_f\|^2_{Q_{\rm rs}}, \quad (14)$$

where $Q_{\rm rs}$ represents the *risk-sensitive* penalty coefficients matrix or adaptive weighting matrix, given by $Q_{\rm rs}(\boldsymbol{\Sigma}_k) = \left(Q^{-1} + \gamma \boldsymbol{\Sigma}_k\right)^{-1}$. The derivation of (14) is given in Appendix A. Such a new formulation is employed by U-MPPI to assess each batch of sigma-point trajectories, where the predicted mean state $\bar{\mathbf{x}}_k$ is replaced with the predicted sigma points $\left\{\mathcal{X}_k^{(i)}\right\}_{i=0}^{2n_x}$. Therefore, the modified RS cost for the $i^{\rm th}$ U-MPPI sampled trajectory within a certain batch is defined by

$$q_{\rm rs}\left(\mathcal{X}_k^{(i)}, \boldsymbol{\Sigma}_k\right) = \frac{1}{\gamma} \log \det \left(\mathbf{I} + \gamma Q \boldsymbol{\Sigma}_k\right) + \left\|\mathcal{X}_k^{(i)} - \mathbf{x}_f\right\|^2_{Q_{\rm rs}}. \ (15)$$

It is noteworthy to observe in (15) that $q_{\rm rs}(\cdot)$ incorporates uncertainty information $\boldsymbol{\Sigma}_k$ as a feedback into the weighting matrix Q, which measures the difference between the predicted sigma point $\mathcal{X}_k^{(i)}$ and the desired state $\mathbf{x}_f$. By incorporating this uncertainty feedback mechanism, the proposed control strategy exhibits a risk-sensitive behavior that effectively minimizes the RS cost over the time-horizon N. Additionally, by utilizing the relationship between γ and the penalty on state reference tracking $Q_{\rm rs}$, we can adjust the behavior of our control strategy toward system uncertainty. This, in turn, enables the development of a more robust and risk-conscious control policy, as empirically validated through intensive simulations outlined in Section V. To be more precise, if $\gamma < 0$, the penalty coefficients matrix $Q_{\rm rs}$ utilized for tracking the desired state increases with the level of system uncertainty $\boldsymbol{\Sigma}_k$. This can be expressed mathematically over a finite time-horizon N as

$$Q_{\rm rs}(\boldsymbol{\Sigma}_0) < Q_{\rm rs}(\boldsymbol{\Sigma}_1) < \cdots < Q_{\rm rs}(\boldsymbol{\Sigma}_{N-1}) \quad \forall \gamma < 0.$$

Consequently, the RS cost function $q_{\rm rs}$ also increases, reflecting the higher penalties imposed for deviation from the desired state $\mathbf{x}_f$. In the context of autonomous navigation in a cluttered environment, this means that as uncertainty increases,
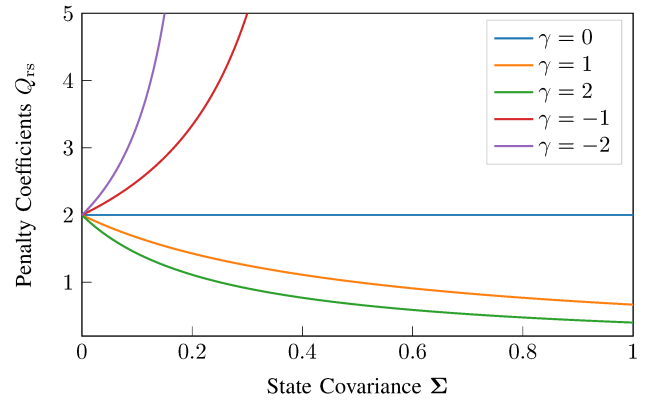

Fig. 5: Influence of different γ values on the penalty coefficients $Q_{\rm rs}$ for state reference tracking with $Q = 2$ and varying 1-dimensional state covariance $\boldsymbol{\Sigma}$.

the control strategy becomes more aggressive in trying to reach the desired state, leading to higher chances of collisions and aggressive maneuvers. On the other hand, if $\gamma > 0$, the penalty matrix decreases as the uncertainty level increases. As a result, the cost function $q_{\rm rs}$ decreases, indicating lower penalties for deviation from the desired state under higher uncertainty. This results in a more conservative approach, where the control policy is more exploratory and less forceful in reaching the desired state, thereby reducing the risk of collisions. Additionally, when $\gamma = 0$, the penalty remains constant and equal to the weighting matrix Q, regardless of the level of the uncertainty. Figure 5 provides, as an example, an intuitive visualization of how the risk-sensitivity parameter γ impacts the penalty coefficients $Q_{\rm rs}$ for state reference tracking, demonstrating transitions between *risk-seeking* ($\gamma > 0$) and *risk-averse* ($\gamma < 0$) behaviors across various sizes of state covariance $\boldsymbol{\Sigma}$.[3] For instance, in the case of $\gamma = 1$, the feedback provided by a large system uncertainty $\boldsymbol{\Sigma}$ will significantly reduce the penalty coefficients $Q_{\rm rs}$, thereby decreasing the cost $q_{\rm rs}$ [11]. Additionally, we can observe that assigning higher positive values to γ (e.g., increasing from $\gamma = 1$ to $\gamma = 2$) results in $Q_{\rm rs}$ decreasing more rapidly with increasing uncertainty $\boldsymbol{\Sigma}$, signifying greater sensitivity to uncertainty. This leads to a more conservative control strategy, reducing penalties on state deviations. In contrast, assigning higher negative values to γ (e.g., decreasing from $\gamma = -1$ to $\gamma = -2$) causes $Q_{\rm rs}$ to increase more sharply with rising uncertainty, indicating lower sensitivity to uncertainty. This fosters a more aggressive control approach, potentially resulting in higher penalties on state deviations. Therefore, higher positive γ values enhance stability by lowering the penalty weight in uncertain conditions, whereas higher negative γ values may lead to instability by increasing the penalty weight. More in-depth analysis of how the U-MPPI performance is influenced by the sign of γ is discussed in Section V-A1.

By incorporating the system uncertainty $\boldsymbol{\Sigma}_k$ into (3), we

---

[3]It is noteworthy that when $\gamma < 0$, $Q_{\rm rs}$ may not always be positive. To ensure $Q_{\rm rs}$ remains positive in this scenario, the condition $\boldsymbol{\Sigma} < -\frac{1}{\gamma Q}$ must be met, where Q and $\boldsymbol{\Sigma}$ are scalars. For an n-dimensional $\boldsymbol{\Sigma}$ and Q, ensuring a positive definite $Q_{\rm rs}$ when $\gamma < 0$ is discussed in [11], which presents one possible solution by bounding $Q_{\rm rs}$ using a sigmoid function.

can obtain the modified *cost-to-go* for each trajectory $\tau_m^{(i)}$ in batch m as

$$\tilde{S}\left(\tau_m^{(i)}\right) = \phi\left(\mathcal{X}_N^{(i)}\right) + \sum_{k=0}^{N-1} \tilde{q}\left(\mathcal{X}_k^{(i)}, \boldsymbol{\Sigma}_k, \mathbf{u}_k, \delta\mathbf{u}_{k,m}\right), \quad (16)$$
$$\forall m \in \{1, \cdots, M_\sigma\}, \quad \forall i \in \{0, \cdots, 2n_x\},$$

where the instantaneous running cost $\tilde{q}$ is a combination of the state-dependent running cost $q\left(\mathcal{X}_k^{(i)}, \boldsymbol{\Sigma}_k\right)$, which relies on the RS cost defined in (15) (as shown in (19) as an example), as well as the quadratic control cost $q_\mathbf{u}\left(\mathbf{u}_k, \delta\mathbf{u}_k\right)$ from (4). Note that the *cost-to-go* for all sigma-point trajectories in the mth batch can be expressed in vector form as $\tilde{\mathbf{S}}\left(\boldsymbol{\tau}_m\right) = \left[\tilde{S}\left(\tau_m^{(0)}\right), \ldots, \tilde{S}\left(\tau_m^{(2n_x)}\right)\right]^\top \in \mathbb{R}^{n_\sigma}$. Similarly, $\tilde{\mathbf{q}}\left(\mathbf{X}_k, \boldsymbol{\Sigma}_k, \mathbf{u}_k, \delta\mathbf{u}_{k,m}\right) = \left[\tilde{q}\left(\mathcal{X}_k^{(0)}, \cdot\right), \ldots, \tilde{q}\left(\mathcal{X}_k^{(2n_x)}, \cdot\right)\right]^\top$.

Algorithm 1 *Real-Time* U-MPPI Control Algorithm

Given:
    M, M_σ, N: # of trajectories, batches, time-horizon,
    $\mathbf{f}, n_x, \Delta t$: Dynamics, state dimension, time-step size,
    $\phi, q, Q, \gamma, \lambda, \nu, \Sigma_\mathbf{u}, R$: Cost/Control parameters,
    $\lambda_\sigma, k_\sigma, n_\sigma, \alpha, \beta, \boldsymbol{\Sigma}_0$: UT parameters,
    SGF: Savitzky-Galoy (SG) convolutional filter,
Input:
    $\mathbf{U} = ControlSequenceInitializer(\mathbf{x}_{\text{init}})$,
    SM: Set U-MPPI sampling mode (SM_0 *OR* SM_1),

1: **while** *task not completed* **do**
2:    $\bar{\mathbf{x}}_0 \leftarrow StateEstimator()$, $\triangleright$ $\bar{\mathbf{x}}_0 \in \mathbb{R}^{n_x}$
3:    $\delta\mathbf{u} \leftarrow RandomNoiseGenerator(\mathbf{0}, \Sigma_\mathbf{u})$, $\triangleright\, \delta\mathbf{u} \in \mathbb{R}^{N \times M_\sigma}$
4:    **for** $m \leftarrow 1$ **to** M_σ *in parallel* **do**
5:       $(\bar{\mathbf{x}}, \boldsymbol{\Sigma}) \leftarrow (\bar{\mathbf{x}}_0, \boldsymbol{\Sigma}_0)$, $\triangleright$ *Actual state* $\mathbf{x}_0 \sim \mathcal{N}(\bar{\mathbf{x}}_0, \boldsymbol{\Sigma}_0)$
6:       $\tilde{\mathbf{S}}\left(\boldsymbol{\tau}_m\right) \leftarrow [0, \ldots, 0]^\top$, $\triangleright$ $\tilde{\mathbf{S}}\left(\boldsymbol{\tau}_m\right) \in \mathbb{R}^{n_\sigma}$
7:       **for** $k \leftarrow 0$ **to** $N-1$ **do**
8:          $\mathbf{X}_k \leftarrow Moments2SigmaPoints(\bar{\mathbf{x}}_k, \boldsymbol{\Sigma}_k)$,
9:          $\mathbf{X}_{k+1} \leftarrow \mathbf{X}_k + \mathbf{f}\left(\mathbf{X}_k, g\left(\mathbf{u}_k + \delta\mathbf{u}_{k,m}\right)\right)\Delta t$,
10:         $\tilde{\mathbf{S}}\left(\boldsymbol{\tau}_m\right) \leftarrow \tilde{\mathbf{S}}\left(\boldsymbol{\tau}_m\right) + \tilde{\mathbf{q}}\left(\mathbf{X}_k, \boldsymbol{\Sigma}_k, \mathbf{u}_k, \delta\mathbf{u}_{k,m}\right)$,
11:         $(\bar{\mathbf{x}}_{k+1}, \boldsymbol{\Sigma}_{k+1}) \leftarrow SigmaPoints2Moments(\mathbf{X}_{k+1})$,
12:       **end for**
13:       $\tilde{\mathbf{S}}\left(\boldsymbol{\tau}_m\right) \leftarrow \tilde{\mathbf{S}}\left(\boldsymbol{\tau}_m\right) + \boldsymbol{\Phi}\left(\mathbf{X}_N\right)$,
14:    **end for**
15:    $\tilde{S}_{\min} \leftarrow \min_m[\tilde{\mathbf{S}}\left(\boldsymbol{\tau}_m\right)]$, $\forall m = \{1, \ldots, M_\sigma\}$
16:    **for** $k \leftarrow 0$ **to** $N-1$ **do**
17:       $\mathbf{u}_k \leftarrow SGF\left(\mathbf{u}_k + \frac{\sum_{m=1}^{M_\sigma} \exp\left(\frac{-1}{\lambda}[\tilde{\mathbf{S}}(\boldsymbol{\tau}_m) - \tilde{S}_{\min}]\right)\delta\mathbf{u}_{k,m}}{\sum_{m=1}^{M_\sigma} \exp\left(\frac{-1}{\lambda}[\tilde{\mathbf{S}}(\boldsymbol{\tau}_m) - \tilde{S}_{\min}]\right)}\right)$,
18:    **end for**
19:    $\mathbf{u}_0 \leftarrow SendToActuators(\mathbf{U})$,
20:    **for** $k \leftarrow 1$ **to** $N-1$ **do**
21:       $\mathbf{u}_{k-1} \leftarrow \mathbf{u}_k$,
22:    **end for**
23:    $\mathbf{u}_{N-1} \leftarrow ControlSequenceInitializer(\mathbf{x}_{\text{init}})$,
24:    Check for task completion
25: **end while**

C. Real-Time U-MPPI Control Algorithm

We are now prepared to describe the *real-time* control-loop of our U-MPPI algorithm, as depicted in Algorithm 1,

employing the default sampling strategy that considers all sigma points (referred to as SM_1). We assume that the system dynamics, control parameters (including UT parameters), cost function for the given task, and control policy for initializing the control sequence for a given initial system state $\mathbf{x}_{\text{init}}$ are provided.[4] First, at every control loop interval Δt, the algorithm estimates the current system state $\bar{\mathbf{x}}_0$ with an external state estimator, generates $N \times M_\sigma$ random control perturbations $\delta\mathbf{u}$ on the GPU using CUDA's random number generation library, and then produces M_σ sets of batches in parallel on the GPU (lines $2:4$). Subsequently, for each batch and starting from the actual (i.e., initial) state $\mathbf{x}_0 \sim \mathcal{N}(\bar{\mathbf{x}}_0, \boldsymbol{\Sigma}_0)$, the algorithm samples and propagates n_σ sigma-point trajectories by applying the non-linear dynamics in (10) to the sigma points computed via (6) (lines $5:9$). These trajectories are then evaluated using (16), and the first and second moments of the propagated state are estimated by applying (8) (lines $10:15$). Then, the algorithm updates the optimal control sequence $\{\mathbf{u}_k\}_{k=0}^{N-1}$, applies a Savitzky-Galoy filter for smoothing, and applies the first control $\mathbf{u}_0$ to the system (lines $16:19$). It then slides down the remaining sequence of length $N-1$ to be utilized in the next control loop interval (lines $20:22$). Lastly, the final control input $\mathbf{u}_{N-1}$ is assigned the value determined by the predefined initialization control policy (line 23). It is noteworthy that when executing the algorithm in sampling mode 0 (SM_0), where only the mean $\mathcal{X}_k^{(0)}$ is used for trajectory sampling, it is essential to set M_σ to M instead of $int(\frac{M}{n_\sigma})$. Additionally, it is necessary to evaluate solely the *cost-to-go* of the nominal trajectory $\tilde{S}\left(\tau_m^{(0)}\right)$.

D. Constraint Handling and Scalability in U-MPPI

Herein, we explore two critical aspects of the U-MPPI control strategy: the effective handling of constraints within MPPI-based control frameworks, including our proposed control strategy, and the scalability and adaptability of the U-MPPI algorithm to higher-dimensional robotic systems, highlighting the advantages and disadvantages of the algorithmic structure given in Algorithm 1.

1) Constraint Handling for MPPI Variants: One of the primary advantages of MPPI variants, including our proposed control strategy, is their ability to include costs in the form of *large-weighted* indicator terms, as they do not require explicit gradient computations. These terms provide *impulse-like* penalties when constraints are violated, effectively handling task-related state constraints and collision avoidance constraints as soft constraints. By incorporating appropriate terms, such as indicator functions, into the running cost function $\tilde{q}$ in (3) and (16), trajectories that violate these constraints can be effectively penalized. Control constraints, such as actuator limits, which are considered hard constraints that must not be violated, can similarly be integrated into the running cost. However, a common issue with this approach is that control constraints are treated as soft constraints, making it challenging to ensure that the control input consistently remains

---

[4]In this study, the default initialization policy is applied, producing a control sequence of zeros.

within its allowed bounds, even after rejecting trajectories that exceed the control input limits. Instead, in this study, control constraints are integrated directly into the underlying non-linear dynamics described in (1) and (10). Consequently, the system dynamics are represented as $\mathbf{x}_{k+1} = f(\mathbf{x}_k, g(\mathbf{w}_k))$ and $\mathbf{X}_{k+1} = \mathbf{f}(\mathbf{X}_k, g(\mathbf{w}_k))$, where $g(\mathbf{w}_k)$ is an element-wise clamping function that confines the control input, $\mathbf{w}_k = \mathbf{u}_k + \delta\mathbf{u}_k$, within specified bounds for all samples drawn from the dynamic system. Specifically, $g(\mathbf{w})$ is defined as $g(\mathbf{w}) = \max(\mathbf{w}_{min}, \min(\mathbf{w}, \mathbf{w}_{max}))$, with $\mathbf{w}_{min}$ and $\mathbf{w}_{max}$ denoting the lower and upper control input limits, respectively. This method effectively converts the optimization problem with control constraints into an unconstrained problem, ensuring that constraints are not violated and the convergence of the MPPI-based algorithm is not compromised, as $g(\mathbf{w})$ only influences the dynamics. Further details on system constraints, including other practical concerns, can be found in [14], [19].

2) Scalability and Adaptability of U-MPPI: Our investigation centers on the question: How effectively can the U-MPPI algorithm scale to accommodate robotic systems with higher-dimensional state spaces, such as those in unmanned aerial vehicles (UAVs) and mobile manipulators? We assert that the proposed U-MPPI algorithm effectively scales and adapts to systems with higher degrees of freedom (DoFs) beyond the 3-DoF mobile robot used in our current experiments for the following reasons. First, the core principles of U-MPPI, including the use of the UT for trajectory sampling and the incorporation of risk-sensitive cost functions, are inherently scalable and adaptable to systems with higher DoFs. The UT, a non-linear transformation method, is designed to efficiently handle higher-dimensional state spaces, making it a superior choice over methods like Gaussian Processes (GP), which require substantially more computational resources. For systems such as UAVs and mobile manipulators, the state space includes additional dimensions such as altitude, pitch, roll, joint angles, and velocities. Our UT-based sampling strategy can extend to these dimensions without significant modifications to the algorithmic structure.

Second, in prior work [15], we demonstrated the applicability of the MPPI algorithm in achieving real-time autonomous navigation of a UAV with a higher-dimensional state space comprising 12 dimensions, and we anticipate that U-MPPI will achieve a comparable level of performance. However, the increased configuration space might impose a greater computational burden due to the need for additional sigma points. This increased burden highlights the importance of several factors to ensure real-time performance, such as the computational capabilities of the GPU, the number of threads utilized, and the efficiency of the implementation. Our GPU-based implementation is designed to exploit parallel processing, allowing for efficient handling of increased state dimensionality. Additionally, the number of sampled trajectories, the chosen prediction time horizon, the optimization of the computational pipeline, and the use of CUDA built-in functions and libraries all play crucial roles in achieving real-time performance. Memory management, data transfer rates between CPU and GPU, and the optimization of kernel functions are also critical factors impacting the algorithm's efficiency.

Additionally, we have investigated the comparative advantages and disadvantages of parallelizing sigma-point trajectories using a reduced number of threads relative to the traditional MPPI approach, where each thread computes the dynamics and costs for an entire trajectory [13]. As delineated in Algorithm 1, our approach involves each thread processing a batch of sigma-point trajectories, thereby reducing computational overhead as fewer threads result in less context-switching and more efficient utilization of GPU resources. This efficiency accounts for U-MPPI's execution times being nearly equivalent to those of the MPPI algorithm, as evidenced in the results section. However, a notable disadvantage of our algorithmic structure is the increased latency in generating each trajectory, attributed to the elevated workload per thread. This issue is worsened in higher-dimensional state spaces or mathematical functions with exponential computational complexity, which can significantly amplify the computational load per thread. To mitigate this issue, a potential solution is to adopt a per-sigma-point trajectory threading model, akin to the MPPI algorithm, which would distribute the workload more evenly across the available threads, potentially reducing latency and enhancing overall performance. In conclusion, while our current work provides a foundational evaluation using a 3-DoF mobile robot, we are confident in the U-MPPI algorithm's potential to scale to higher-dimensional robotic systems.

V. Simulation-Based Evaluation

In this section, we evaluate the effectiveness of our proposed control strategy by comparing it to the standard MPPI control framework. Herein, we focus on two goal-oriented autonomous ground vehicle (AGV) navigation tasks in 2D cluttered environments. The first task, presented in Section V-A, involves maneuvering the AGV within a given map. This experiment allows us to understand the algorithmic advantages of the proposed U-MPPI control scheme without adding real-world complexity. Section V-B introduces a more complex and realistic scenario where the map is unknown a priori. It examines the adaptability and robustness of our proposed control strategy, providing a thorough evaluation of its potential in real-world applications.

A. Aggressive Navigation in Known Cluttered Environments

1) Simulation Setup: In this study, we employ the kinematics model of a differential drive robot, as specified in (17), for sampling trajectories in the conventional MPPI and propagating sigma points in the proposed U-MPPI method, as outlined in (9). The state of the model encompasses the robot's position and orientation in the world frame $\mathcal{F}_o$, given by $\mathbf{x} = [x, y, \theta]^\top \in \mathbb{R}^3$. The control input consists of the robot's linear and angular velocities, denoted by $\mathbf{u} = [v, \omega]^\top \in \mathbb{R}^2$.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}. \tag{17}$$

To ensure a fair comparison, both MPPI and U-MPPI simulations were conducted under consistent parameters. A time prediction of $8\,\text{s}$ and a control rate of $30\,\text{Hz}$ were employed, resulting in 240 control time-steps (i.e., $N = 240$). At each time-step Δt, a total of 2499 rollouts were sampled, accompanied by an exploration noise of $\nu = 1200$. To account for control weighting, a control weighting matrix R was utilized, which was formulated as $\lambda\Sigma_{\mathbf{u}}^{-\frac{1}{2}}$. Additionally, the inverse temperature parameter was set to $\lambda = 0.572$, while the control noise variance matrix $\Sigma_{\mathbf{u}} = \text{Diag}\left(\sigma_v^2, \sigma_w^2\right)$ was defined as $\Sigma_{\mathbf{u}} = \text{Diag}\left(0.023, 0.028\right)$. To smooth the control sequence, we utilized the Savitzky-Galoy (*SG*) convolutional filter with a quintic polynomial function, i.e., $n_{sg} = 5$, and a window length l_{sg} of 61. As described in Section IV-A, U-MPPI has additional parameters in the unscented transform to regulate the spread of the sigma points. These parameters are set to $\alpha = 1, k_\sigma = 0.5,$ and, $\beta = 2$. Additionally, the initial state covariance matrix Σ_0 is set to $0.001\mathbf{I}_3$. It is worth mentioning that λ and $\Sigma_{\mathbf{u}}$, which play crucial roles in determining the behavior of sampling-based MPC schemes, are set based on the intensive simulations conducted in [22]. These simulations considered various values of λ and $\Sigma_{\mathbf{u}}$ with the objective of selecting the optimal set of hyperparameters that adhere to the control constraints. Other parameters were fine-tuned using a *trial-and-error* method, guided by an understanding of the role and impact of each parameter on the algorithm's performance [14], [15], [38]. The baseline and U-MPPI are implemented using Python and incorporated within the Robot Operating System (ROS) framework. They are executed in *real-time* on an NVIDIA GeForce GTX 1660 Ti laptop GPU.

In the context of the 2D navigation task, MPPI employs a commonly-used instantaneous state-dependent cost function described in (18). This cost function consists of two terms. The first term, denoted as $q_{\text{state}}(\mathbf{x}_k)$, encourages the robot to reach the desired state; its formulation, which employs a quadratic expression, is provided in (12). The second term, $q_{\text{crash}}(\mathbf{x}_k) = w_{\text{crash}}\mathbb{I}_{\text{crash}}$, serves as an indicator function that imposes a high penalty when the robot collides with obstacles, with $\mathbb{I}_{\text{crash}}$ being a Boolean variable and w_{crash} representing the collision weighting coefficient. While implementing the proposed U-MPPI approach, we replace the quadratic cost $q_{\text{state}}(\mathbf{x}_k)$, as depicted in (19), with our newly introduced RS cost denoted as $q_{\text{rs}}\left(\mathcal{X}_k^{(i)}, \Sigma_k\right)$ and defined in (15). Within the scope of this work, we set the values of Q and w_{crash} as follows: $Q = \text{Diag}(2.5, 2.5, 2)$ and $w_{\text{crash}} = 10^3$. It is worth noting that in this particular task, assigning positive values to γ (i.e., $\gamma > 0$) ensures a trade-off between compelling the robot to reach its desired state and minimizing the risk of collisions with obstacles. This trade-off arises from the fact that the penalty coefficients matrix Q_{rs}, utilized for tracking the desired state, decreases as the system uncertainty Σ_k increases over the time-horizon N. Therefore, we have chosen $\gamma = 1$ to maintain this trade-off and strike a balance between task completion and collision avoidance. On the other hand, we believe that assigning negative values to γ in applications such as autonomous racing [14] and visual servoing [19] could enhance the performance of U-MPPI. In such tasks, it is

crucial to prioritize forcing the current state to reach its desired state, which can be achieved by assigning a higher penalty coefficients matrix Q_{rs}.

$$q(\mathbf{x}_k) = q_{\text{state}}(\mathbf{x}_k) + q_{\text{crash}}(\mathbf{x}_k). \tag{18}$$

$$q\left(\mathcal{X}_k^{(i)}, \Sigma_k\right) = q_{\text{rs}}\left(\mathcal{X}_k^{(i)}, \Sigma_k\right) + q_{\text{crash}}\left(\mathcal{X}_k^{(i)}\right). \tag{19}$$

2) Simulation Scenario: In order to assess the effectiveness of the proposed control framework within cluttered environments, three distinct scenarios with different difficulty levels were examined. In each scenario, we randomly generate one unique forest type consisting of 25 individual forests, resulting in a total of $\mathcal{N}_T = 25$ tasks. Each forest represents a cluttered environment with dimensions of $50\,\text{m} \times 50\,\text{m}$. In the first scenario (referred to as *Scenario #1*), the average distance between obstacles was $1.5\,\text{m}$, indicated as $d_{\text{min}}^{\text{obs}} = 1.5\,\text{m}$; while in the second and third scenarios (i.e., *Scenario #2* and *Scenario #3*), they were placed at average distances of $2\,\text{m}$ and $3\,\text{m}$, respectively. Additionally, we set the maximum desired velocity v_{max} of the robot based on the degree of clutter in each scenario. Specifically, v_{max} is set to $2\,\text{m/s}$, $3\,\text{m/s}$, and $4\,\text{m/s}$ in *Scenario #1, #2*, and *#3*, respectively.

3) Performance Metrics: To achieve a fair comparison between the two control strategies, we use the following criteria: (i) firstly, in all simulation instances, the robot is required to reach the designated desired pose, denoted as $\mathbf{x}_f = [50, 50, 0]^\top$, from a predetermined initial pose $\mathbf{x}_0 = [0, 0, 0]^\top$, measured in $([\text{m}], [\text{m}], [\text{deg}])$; (ii) secondly, a comprehensive set of metrics is defined to evaluate the overall performance [22], including the *task completion percentage* $\mathcal{T}_c$, which is computed as the average distance the robot travels towards the desired state to the total planned distance[5]; the *success rate* $\mathcal{S}_R$, which is quantified as the ratio of successfully completed tasks to the total tasks undertaken; the *average number of collisions* $\mathcal{N}_c$ encountered by the robot during the tasks; the *average number of local minima occurrences* $\mathcal{R}_{\text{lm}}$, indicating how often the robot gets trapped in local minima; the *average distance* traversed by the robot d_{av} to reach the desired state $\mathbf{x}_f$ from its initial state $\mathbf{x}_0$; the *average linear velocity* v_{av} of the robot during the execution of its task in the cluttered environment; and the *average execution time per iteration* t_{exec} of the control algorithm.

These metrics are essential for evaluating the efficacy of a control strategy in autonomous vehicle navigation, highlighting its proficiency in generating safe, efficient, and resilient trajectories under varying conditions. In detail, the task completion percentage $\mathcal{T}_c$ and success rate $\mathcal{S}_R$ measure the effectiveness of the approach, with higher values indicating the control strategy's ability to successfully complete tasks and achieve goals under various conditions. Metrics such as the number of collisions $\mathcal{N}_c$ and local minima occurrences $\mathcal{R}_{\text{lm}}$ provide insights into the safety and robustness of the control strategy, with fewer crashes and reduced occurrences of local minima indicating more reliable navigation. A shorter average distance d_{av} indicates more efficient trajectory planning,

---

[5]In this research, the Euclidean distance between $\mathbf{x}_0$ and $\mathbf{x}_f$ is used to calculate the planned distance.

TABLE I: Performance comparisons of the two control schemes, with gray cells indicating better performance and values for d_{av}, v_{av}, and $t_{\text{exec.}}$ representing the mean values for only the successful tasks.

Scheme	$\mathcal{R}_{\text{lm}}(\mathcal{N}_c)$	$\mathcal{S}_R$ [%]	$\mathcal{T}_c$ [%]	d_{av} [m]	v_{av} [m/s]	$t_{\text{exec.}}$ [ms]
Scenario #1: $v_{\max} = 2\,\text{m/s}$ & $d_{\min}^{\text{obs}} = 1.5\,\text{m}, \gamma = 1, w_{\text{crash}} = 10^3$						
MPPI	9 (2)	78	92.86	75.18	1.84 ± 0.18	9.68
U-MPPI	2 (0)	96	98.78	75.34	1.85 ± 0.17	9.12
Scenario #2: $v_{\max} = 3\,\text{m/s}$ & $d_{\min}^{\text{obs}} = 2\,\text{m}, \gamma = 1, w_{\text{crash}} = 10^3$						
MPPI	2 (1)	94	98	75.31	2.49 ± 0.73	10.03
U-MPPI	0 (0)	100	100	75.78	2.53 ± 0.49	8.87
Scenario #3: $v_{\max} = 4\,\text{m/s}$ & $d_{\min}^{\text{obs}} = 3\,\text{m}, \gamma = 1, w_{\text{crash}} = 10^3$						
MPPI	0 (0)	100	100	72.19	3.51 ± 0.74	8.55
U-MPPI	0 (0)	100	100	71.98	3.54 ± 0.59	7.7

showcasing optimized path generation. Higher average robot speed v_{av} reflects the ability to maintain a faster pace without compromising safety, crucial for timely task completion in real-world applications. Lower average execution time $t_{\text{exec.}}$ highlights efficiency and responsiveness, ensuring real-time performance in practical scenarios. In this work, successful task completion is defined as the robot reaching the desired pose without colliding with obstacles within a predefined finite time, represented by $\mathcal{T}_c = 100\%$, $\mathcal{N}_c = 0$, and $\mathcal{R}_{\text{lm}} = 0$. Furthermore, in all scenarios, if the robot fails to reach the desired pose within $70\,\text{s}$ while successfully avoiding collisions, we classify the simulation episode as reaching a local minimum, indicated by $\mathcal{R}_{\text{lm}} = 1$.

4) Simulation Results: Table I presents the performance analysis of the proposed U-MPPI and the baseline MPPI control strategies, considering the three predefined scenarios. For each scenario, two trials were conducted over the 25 individual forests, resulting in a total of 50 tasks ($\mathcal{N}_T = 50$). In *Scenario #1*, where $d_{\min}^{\text{obs}} = 1.5\,\text{m}$, it is noteworthy that U-MPPI outperforms MPPI. Specifically, U-MPPI achieves a notably higher task completion percentage ($\mathcal{T}_c = 98.78\%$) compared to MPPI ($\mathcal{T}_c = 92.86\%$), effectively avoids collisions ($\mathcal{N}_c = 0$), mitigates local minimum occurrences ($\mathcal{R}_{\text{lm}} = 2$), and achieves a significantly higher success rate ($\mathcal{S}_R = 96\%$ vs. $\mathcal{S}_R = 78\%$ when MPPI is utilized). Furthermore, it successfully navigates the cluttered environment with a slightly improved average linear velocity v_{av}, which exhibits a very low standard deviation and approaches the maximum desired speed $v_{\max}$ of $2\,\text{m/s}$ (likewise observed in the other two scenarios). Similarly, in *Scenario #2*, with a minimum obstacle distance of $d_{\min}^{\text{obs}} = 2\,\text{m}$, U-MPPI achieves a perfect task completion rate of 100%, outperforming MPPI's 98%, as it successfully avoids collisions and local minima, surpassing the baseline MPPI that experienced one collision with obstacles ($\mathcal{N}_c = 1$) and encountered two instances of local minima ($\mathcal{R}_{\text{lm}} = 2$).

In the least cluttered scenario, *Scenario #3*, both control strategies effectively complete all assigned tasks while successfully avoiding obstacles in the cluttered environment. However, U-MPPI stands out by offering a slightly more direct route towards the desired pose, with the robot traveling an average distance d_{av} of approximately $71.98\,\text{m}$, compared to $72.19\,\text{m}$ when utilizing MPPI. On the contrary, in *Scenarios #1* and #2, MPPI demonstrates an enhanced performance in terms of the average distance traveled d_{av} by the robot

when compared to our proposed U-MPPI. Nevertheless, it should be emphasized that the average distance is computed solely on successful task completions, and MPPI has a lower number of successful tasks compared to U-MPPI, which affects the comparison of average distances. This indicates that U-MPPI outperforms MPPI in terms of overall route completion and reliability, as it is more effective in achieving collision-free navigation. In the last column of Table I, despite both control methods ensuring *real-time* performance (since $t_{\text{exec.}} < 33.33\,\text{ms}$), it is worth emphasizing that the average execution time $t_{\text{exec.}}$ of our proposed U-MPPI control strategy is slightly shorter than that of MPPI. This can be attributed to the parallel implementation of the U-MPPI algorithm on GPU, where each thread is responsible for computing the dynamics and costs of the entire batch when sampling from all sigma points (SM_1). On the other hand, the parallel implementation of MPPI, as well as U-MPPI with sampling mode 0 (SM_0), employs a single thread to compute each sampled trajectory, resulting in a relatively longer execution time, as evidenced by the intensive simulations in Table I, as well as Tasks #3 and #4 in Table II, where only the mean $\mathcal{X}_k^{(0)}$ is used for trajectory sampling (i.e., SM_0).

To summarize, the intensive simulations clearly demonstrate that our U-MPPI method consistently outperforms the baseline MPPI control framework in all tested scenarios, particularly in environments with higher levels of clutter. These remarkable results can be credited to two key factors: the effective utilization of an unscented-based sampling strategy, which provides more flexible and efficient trajectories, and the incorporation of a *risk-sensitive* (RS) cost function that explicitly takes into account risk and uncertainty during the trajectory evaluation process; thanks to the incorporation of these crucial components, our approach ensures a significantly enhanced exploration of the state-space of the controlled system, even while leveraging the same injected Gaussian noise $\delta\mathbf{u}_k$ into the mean control sequence, effectively reducing the likelihood of being trapped in local minima and yielding a safer and more resilient control system that is suitable for aggressive navigation in highly complex cluttered environments.

To achieve a comprehensive understanding of how the behavior of the U-MPPI control strategy is affected by integrating the proposed sampling strategy and RS cost function, we expanded our intensive simulations in Table II to include varying operating conditions and hyper-parameters, differing from those utilized in Section V-A1. More precisely, in the first four intensive simulations (namely, Test #1 to Test #4), we investigate the potential benefits of integrating the RS cost function into the U-MPPI control strategy through two approaches: (i) reducing the collision weighting coefficient w_{crash} (specifically, Tests #1 and #2), and (ii) adopting sampling mode 0 (SM_0) as an alternative to the default mode SM_1 (i.e., Tests #3 and #4). Additionally, in the subsequent four tests, we extensively analyze the influence of the UT parameters on the performance of U-MPPI. For Tests #1 and #2, we replicated the U-MPPI simulations presented in Table I, specifically for *Scenarios* #1 and #2, by assuming a reduced collision weighting coefficient w_{crash} of 500, representing half of its nominal value. We can clearly observe that lowering the

TABLE II: Influence of collision weighting coefficient w_{crash}, sampling modes (i.e., SM_0 and SM_1), and UT parameters (namely, $\boldsymbol{\Sigma}_0, k_\sigma, \alpha$) on U-MPPI performance.

Test No.	$\mathcal{R}_{\text{lm}}(\mathcal{N}_c)$	$\mathcal{S}_R$ [%]	$\mathcal{T}_c$ [%]	d_{av} [m]	v_{av} [m/s]	$t_{\text{exec.}}$ [ms]
Scenarios #1 and #2: $w_{\text{crash}} = 500$ instead of $w_{\text{crash}} = 10^3$						
Test #1	2 (0)	96	99.4	74.43	1.86 ± 0.15	8.69
Test #2	0 (0)	100	100	74.55	2.56 ± 0.56	8.9
Scenarios #1 and #2: SM_0 instead of SM_1						
Test #3	3 (0)	94	98.24	75.88	1.84 ± 0.22	12.87
Test #4	1 (0)	98	99.8	76.87	2.47 ± 0.67	12.59
Scenario #1: Impact of UT parameters ($\boldsymbol{\Sigma}_0, k_\sigma, \alpha$)						
Test #5	0 (0)	100	100	77.19	1.79 ± 0.23	8.80
Test #6	1 (0)	98	98.7	74.65	1.80 ± 0.24	8.88
Test #7	4 (0)	92	96.88	75.75	1.81 ± 0.25	9.46
Test #8	9 (0)	82	92.91	75.25	1.83 ± 0.22	9.02

(a) U-MPPI: Success rate $\mathcal{S}_R$

(b) U-MPPI: Average distance d_{av}

(c) MPPI: Success rate $\mathcal{S}_R$

(d) U-MPPI vs. MPPI: Success rate $\mathcal{S}_R$

Fig. 6: Impact of decreasing collision weighting coefficient w_{crash} on (a) U-MPPI success rate $\mathcal{S}_R$, (b) average distance traveled by the robot d_{av} in U-MPPI, and (c) MPPI success rate $\mathcal{S}_R$, as well as (d) the effect of utilizing sampling mode 0 (SM_0) on U-MPPI success rate $\mathcal{S}_R$ compared to MPPI success rate.

value of w_{crash} has no significant impact on the success rate $\mathcal{S}_R$ (as also depicted in Fig. 6(a)) and task completion rate $\mathcal{T}_c$. Nevertheless, it demonstrates improved performance in the robot's average travel distance d_{av} for completing the assigned tasks in both scenarios, outperforming both U-MPPI and MPPI as indicated in Table I. As an example, in *Scenario #2* shown in Fig. 6(b), we can observe that d_{av} is approximately $1.23\,\text{m}$ shorter than that of U-MPPI when w_{crash} is set to 10^3. On the contrary, we empirically observed that reducing w_{crash} in the case of MPPI, which utilizes a *risk-neutral* technique for evaluating sampled trajectories (as expressed in (18)), does not lead to a performance improvement, as depicted in Fig. 6(c). For instance, in *Scenario #1*, it can be noted from Fig. 6(c) that the success rate $\mathcal{S}_R$ experiences a decline from 78% to 72% with the reduction of w_{crash}. It is also noteworthy that, although U-MPPI already demonstrated higher average speeds v_{av} in *Scenarios #1* and *#2* in Table I, adjusting w_{crash} can further enhance the average speed. In contrast, adopting sampling mode 0 (SM_0) and modifying the UT scaling parameters lead to a slight reduction in average speed, as shown in Tests #3 to #8.

By employing sampling mode 0 (SM_0) in Tests #3 and #4 as an alternative to the default sampling SM_1 in U-MPPI, a slight decrease in performance is observed. However, U-MPPI continues to demonstrate impressive capabilities in successfully accomplishing assigned tasks and navigating around obstacles, outperforming the classical MPPI, particularly in *Scenario #1* (refer to Fig. 6(d)), owing to the integration of our efficient RS cost function for trajectory assessment. Furthermore, the comprehensive simulations performed in Tests #3 and #4 highlight the importance of employing the default sampling strategy SM_1, which takes into account all sigma points, in extremely challenging scenarios. This strategy leads to enhancements in both the success rate and the trajectory quality of the robot, as depicted in the illustrative example presented in Fig. 7(a). It is also important to note that the average execution time $t_{\text{exec.}}$ of our proposed U-MPPI algorithm operating in SM_0 is quite longer than in SM_1. This stems from the fact that the algorithm samples M batches in parallel, rather than M_σ batches, while using only the nominal trajectory $\mathcal{X}_k^{(0)}$ for trajectory sampling and evaluating the cost-
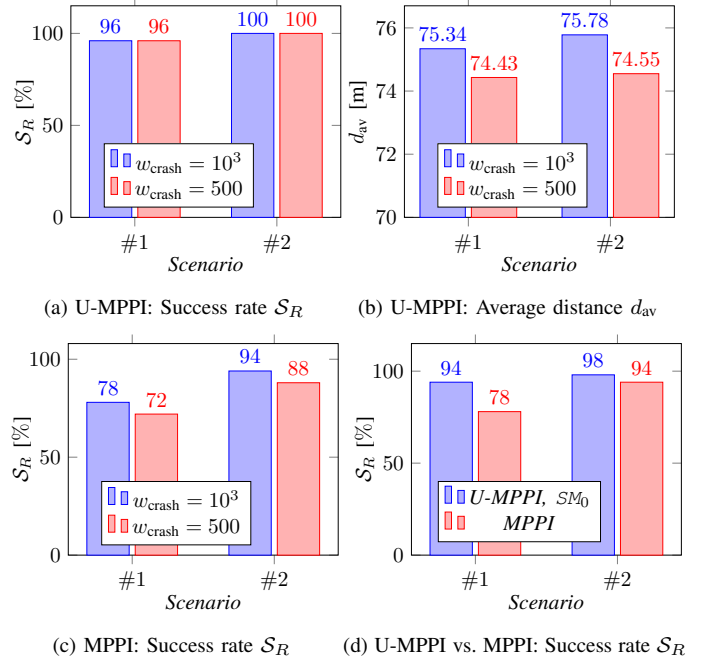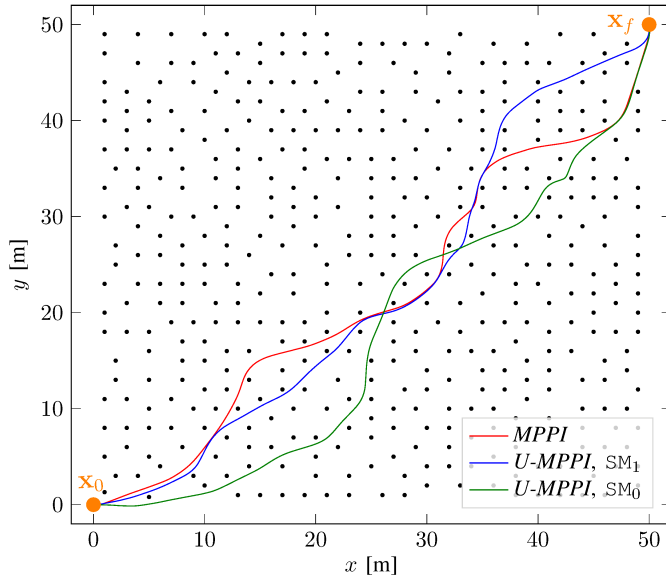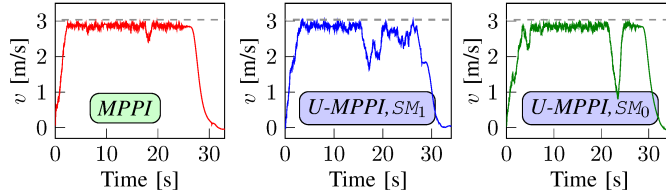
to-go $\tilde{S}\left(\tau_m^{(0)}\right)$, as explained in Section IV-C. By sampling M batches, the U-MPPI algorithm ensures that the total number of M trajectories remains similar to that in MPPI, thereby allowing for a fair comparison. Additionally, to accurately calculate the risk-sensitive cost function over the time horizon N, it is necessary to propagate all sigma points, even if only the nominal trajectory $\mathcal{X}_k^{(0)}$ is evaluated at each time-step k. This propagation is required to obtain the covariance $\boldsymbol{\Sigma}_k$, which is then utilized by the risk-sensitive cost function as illustrated in (15).

Now, it is time to delve into investigating how the key UT parameters (namely, $\boldsymbol{\Sigma}_0, k_\sigma, \alpha$) affect the distribution of sampled rollouts and their subsequent influence on the performance of the U-MPPI algorithm; to achieve this, we will explore their effects in highly cluttered environments, with a specific focus on *Scenario #1*. In Test #5, the initial state covariance matrix $\boldsymbol{\Sigma}_0$ is increased from $0.001\mathbf{I}_3$ to $0.005\mathbf{I}_3$. It is observed that this increase in $\boldsymbol{\Sigma}_0$ results in a more conservative yet safer trajectory, with a success rate $\mathcal{S}_R$ of 100% and an average traveled distance d_{av} of $77.19\,\text{m}$, compared to a success rate of 96% and an average distance of $75.34\,\text{m}$ when $\boldsymbol{\Sigma}_0$ is set to $0.001\mathbf{I}_3$, as shown in Table I. During Tests #6 and #7, we adjust the UT scaling parameters (k_σ and α), which control the spread of sigma points; specifically, we increase k_σ from 0.5 to 3 in Test #6 and decrease α from 1 to 0.1 in Test #7, while keeping all other parameters constant. It is noteworthy to observe in Test #6 that assigning a higher value to k_σ, along with $\alpha = 1$, generates more widely spread trajectories that cover a larger state space. This enables the robot to explore the environment more extensively and

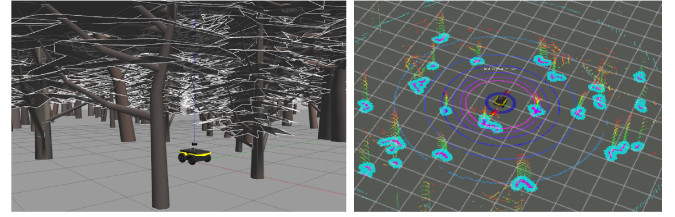(a) Robot trajectories comparison between MPPI and U-MPPI with random obstacles (black dots)

(b) MPPI: Robot velocity v (c) U-MPPI, SM_1: v (d) U-MPPI, SM_0: v

Fig. 7: Performance analysis of MPPI and U-MPPI in a $50\,\mathrm{m} \times 50\,\mathrm{m}$ cluttered environment with $2\,\mathrm{m}$ obstacle spacing (*Scenario #2*), utilizing U-MPPI-specific sampling modes (SM_0 and SM_1).

find better solutions, while improving the quality of the robot trajectory (as $d_{\mathrm{av}} = 74.65\,\mathrm{m}$), thereby reducing the likelihood of getting trapped in local minima. Conversely, reducing the spread of sigma points by assigning very low values to α increases the likelihood of the robot becoming trapped in local minima, as illustrated in the simulations conducted during Test #7. In Test #8, we replicated the simulations from Test #7 using the same reduced UT scaling parameters ($k_\sigma = 0.5$ and $\alpha = 0.1$), while excluding the *risk-sensitive* behavior by setting the *risk-sensitive* parameter γ to 0, resulting in a constant penalty coefficients matrix Q_{rs} that is equal to the weighting matrix Q. Such a setup yields a sampling strategy closely resembling that of MPPI, resulting in the worst performance achieved, albeit only marginally better than the performance of MPPI illustrated in Table I.

In Fig. 7, we showcase the behavior of MPPI and U-MPPI (utilizing U-MPPI-specific sampling modes) in one of the randomly generated $50\,\mathrm{m} \times 50\,\mathrm{m}$ cluttered environments with $2\,\mathrm{m}$ obstacles apart, i.e., *Scenario #2*. This particular environment was selected due to its classification as one of the most challenging environments within *Scenario #2*, wherein MPPI demonstrated difficulty in achieving efficient navigation by providing a shorter route towards the goal over the two conducted trials. As shown in Fig. 7(a), both control strategies successfully achieve collision-free navigation in the

(a) Gazebo forest-like environment   (b) Rviz costmap visualization

Fig. 8: Snapshot capturing (a) our Jackal robot operating in a forest-like environment, equipped with a Velodyne VLP-16 LiDAR sensor, and (b) the corresponding *2D* local costmap generated by the on-board Velodyne LiDAR sensor, where purple and cyan cells depict obstacles and their inflation.

cluttered environment. However, owing to the incorporation of the risk-sensitive cost function and the unscented-based sampling strategy, U-MPPI in the default sampling mode 1 (SM_1) demonstrates a significantly shorter route to the desired pose, with a robot trajectory length d_{av} of $75.45\,\mathrm{m}$, compared to $77.39\,\mathrm{m}$ for the classical MPPI and $78.33\,\mathrm{m}$ for U-MPPI in sampling mode 0 (SM_0). In the given cluttered environment, as depicted in Fig. 7(b), MPPI demonstrates a smoother velocity profile compared to U-MPPI (see Figs. 7(c) and 7(d)), with an average traveling speed v_{av} of $2.63\,\mathrm{m/s}$, as opposed to $v_{\mathrm{av}} = 2.52\,\mathrm{m/s}$ for U-MPPI (SM_1) and $v_{\mathrm{av}} = 2.57\,\mathrm{m/s}$ for U-MPPI (SM_0). The observed fluctuations in the velocity profile under U-MPPI can indeed be attributed to the increased state-space exploration induced by the UT; specifically, the UT propagates both the mean and covariance of the system state, resulting in a more comprehensive exploration of possible trajectories. Consequently, this leads to more aggressive adjustments in control inputs, which manifest as fluctuations in the velocity profile. Furthermore, the risk-sensitive cost function in U-MPPI, which incorporates the system's uncertainty, amplifies this behavior. Notably, in our U-MPPI implementation, the risk sensitivity parameter γ is set to 1, leading to more conservative control actions, as detailed in Section IV-B. This setting prioritizes safety and robustness by explicitly accounting for uncertainty, thereby contributing to the observed velocity fluctuations. In summary, the fluctuations in the velocity profile under U-MPPI are a direct consequence of the algorithm's design to handle uncertainties and explore a wider range of trajectories while maintaining conservative control actions. This is particularly evident with $\gamma > 0$ (e.g., $\gamma = 1$) and when navigating at high speed in extremely crowded environments such as *Scenario #2*. These fluctuations indicate the algorithm's capacity to dynamically adjust to changing conditions, ensuring robust and safe navigation. Furthermore, it is worth noting that none of the control strategies violate the control (namely, velocity) constraint, which is defined as $v \leq v_{\mathrm{max}} = 3\,\mathrm{m/s}$, as observed from the velocity profiles.

B. Aggressive Navigation in Unknown Environments

In Section V-A, extensive simulations of our proposed control strategies were conducted using a fully autonomous differential wheeled ClearPath Jackal robot to ensure their effectiveness and validate their performance in realistic scenar-

TABLE III: Performance statistics of the two control strategies over ten trials in *Forest #1* ($0.1\,\text{trees/m}^2$) and *Forest #2* ($0.2\,\text{trees/m}^2$).

Scheme	$\mathcal{R}_{\text{lm}}(\mathcal{N}_c)$	$\mathcal{S}_R[\%]$	$\mathcal{T}_c[\%]$	d_{av} [m]	v_{av} [m/s]	$t_{\text{exec.}}$ [ms]
Forest #1: $v_{\text{max}} = 3\,\text{m/s}$ & $w_{\text{max}} = 3\,\text{rad/s}, \gamma = 1, w_{\text{crash}} = 10^3$						
MPPI	2 (0)	80	91.01	185.33±0.43	2.50±0.68	8.52±0.61
U-MPPI	0 (0)	100	100	185.37±0.22	2.40±0.75	9.87±0.76
Forest #2: $v_{\text{max}} = 2\,\text{m/s}$ & $w_{\text{max}} = 1\,\text{rad/s}, \gamma = 1, w_{\text{crash}} = 10^3$						
MPPI	4 (1)	50	88.08	118.25±1.7	1.56±0.44	8.08±0.91
U-MPPI	1 (0)	90	97.91	116.76±0.43	1.67±0.41	10.75±0.75

ios, utilizing the kinematics model outlined in (17) for rolling out sampled trajectories for both control strategies. However, it is important to note that the autonomous navigation tasks in cluttered environments are typically carried out under the assumption that the costmap representing the environment is known beforehand. Such a setup or configuration could be limited in many real-world scenarios where autonomous robot systems often need to operate in *partially* observed environments due to the constraints of limited sensor range and the impracticality of obtaining a complete map before task deployment [15]. To tackle this challenge, the robot needs to construct a *real-time* 2D local costmap, also known as a 2D occupancy grid map, centered at its current state; this grid map is utilized to store information about obstacles in the robot's surrounding area, gathered from the incoming sensory data acquired by its on-board sensor. To this end, we utilize the *costmap_2d* ROS package to generate the occupancy grid based on the sensory data, as depicted in Fig. 8(b) [43]. Afterward, the local costmap is incorporated into the optimization problem of the sampling-based MPC algorithm to assess sampled trajectories, aiming to achieve collision-free navigation. We refer to our previous work [22] and the corresponding open-source code.[6] In this work, we successfully integrated the 2D grid map into the MPPI algorithm.

1) Simulation Setup: We employ the same simulation setup for both control strategies previously presented in Section V-A1, with the exception that: (i) the collision indicator function q_{crash} is herein calculated based on the robot-centered 2D grid map, and (ii) the time prediction is reduced to $6\,\text{s}$, resulting in $N = 180$, to be compatible with the size of the grid map. In this study, a 16-beam Velodyne LiDAR sensor mounted on the Clearpath Jackal AGV is used to construct the grid map (local costmap), with the costmap having dimensions of $240\,\text{cell} \times 240\,\text{cell}$ and a resolution of $0.05\,\text{m/cell}$.

2) Simulation Scenarios: For the performance evaluation, we utilize two types of forest-like cluttered environments within the Gazebo simulator, each measuring $50\text{m} \times 50\text{m}$. The first type, referred to as *Forest #1*, consists of trees of different sizes with a density of $0.1\,\text{trees/m}^2$, whereas the second type, known as *Forest #2*, contains tree-shaped obstacles with a density of $0.2\,\text{trees/m}^2$. In *Forest #1*, the maximum desired velocity v_{max} is $3\,\text{m/s}$, and the maximum desired angular velocity w_{max} is $3\,\text{rad/s}$. While in *Forest #2*, they are set to $2\,\text{m/s}$ and $1\,\text{rad/s}$, respectively.

[6]https://github.com/IhabMohamed/log-MPPI_ros

(a) MPPI: @ G_2 (b) After 1 s   (c) After 2 s   (d) After 3 s

(e) Ours: @ G_2 (f) After 1 s   (g) After 2 s   (h) After 3 s
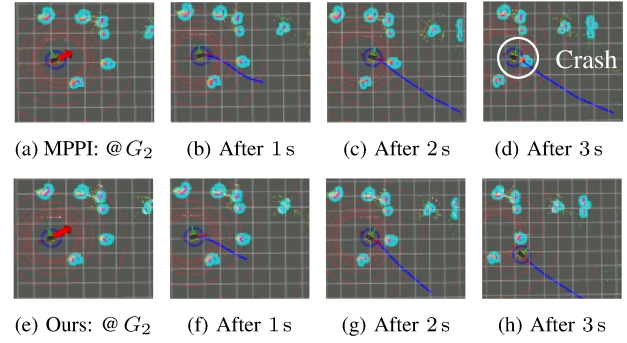
Fig. 9: Planned trajectories by MPPI (first row) and U-MPPI (second row) using the 2D local costmap, starting from the updated pose $G_2 = [-18, 2, 40]^\top$ and navigating towards $G_3 = [20, -21, 90]^\top$.

3) Performance Metrics: To assess the efficiency of the U-MPPI control scheme in unknown environments, we compare it with the baseline MPPI by evaluating the predefined set of performance metrics outlined in Section V-A3, namely: $\mathcal{R}_{\text{lm}}$, $\mathcal{N}_c$, $\mathcal{S}_R$, $\mathcal{T}_c$, d_{av}, v_{av}, and $t_{\text{exec.}}$. In the first forest-like environment (*Forest #1*), the robot is directed to autonomously navigate from an initial pose $G_0 = [0, 0, 0]^\top$ to a sequence of desired poses expressed in ([m], [m], [deg]): $G_1 = [20, 20, 45]^\top, G_2 = [-18, 2, 0]^\top, G_3 = [20, -21, 90]^\top$, $G_4 = [20, 20, 0]^\top$, and ultimately reaching a stop at $G_5 = [0, 0, 100]^\top$. Meanwhile, in *Forest #2*, for the sake of simplicity, the robot navigates solely from G_0 to G_3, where it comes to a stop.

4) Simulation Results: In Table III, we present a comparison of performance statistics for the proposed control strategies in achieving goal-oriented autonomous navigation in both *Forest #1* and *Forest #2*, where the statistics are averaged over 10 trials for each environment. The obtained results validate the anticipated superiority of U-MPPI over MPPI in all scenarios, owing to its efficient unscented-based sampling distribution policy and *risk-sensitive* based trajectory evaluation technique. This superiority is evident in several aspects, including: (i) achieving a higher task completion rate $\mathcal{T}_c$, such as $\mathcal{T}_c = 100\%$ with U-MPPI versus 91.01% when using MPPI in *Forest #1*, (ii) reducing the probability of encountering local minima, as demonstrated by the comparison in *Forest #2*, where U-MPPI leads to $\mathcal{R}_{\text{lm}} = 1$ compared to $\mathcal{R}_{\text{lm}} = 4$ with MPPI, and (iii) improving the quality of the generated robot trajectory, as evidenced by a significantly shorter average distance traveled by the robot d_{av}, particularly noticeable in *Forest #2*. Furthermore, it is worth emphasizing that both control methods guarantee *real-time* performance, highlighting the superiority of the sampling-based MPC algorithm, particularly our proposed U-MPPI, in incorporating not only the local costmap but also the unscented transform into the optimization problem without introducing additional complexity.

To showcase the superior performance of U-MPPI compared to conventional MPPI in enhancing control safety during aggressive navigation, we introduce a specific adjustment to the G_2 configuration within *Forest #1*. This adjustment entails shifting the desired heading angle from 0 to 40 degrees

(see Fig. 9(a)), resulting in $G_2 = [-18, 2, 40]^\top$ instead of $G_2 = [-18, 2, 0]^\top$, which in turn requires the robot to perform additional rotational maneuvers. Throughout five simulation runs, our proposed approach effectively completes all trials while penalizing trajectories with w_{max} values exceeding $3\,\mathrm{rad/s}$. Figures 9(e) to 9(h) depict snapshots of collision-free and predicted optimal trajectories generated by U-MPPI at three successive time points as the robot advances towards its target pose, G_3. On the other hand, when MPPI is employed, the robot experiences collisions with the same tree on three separate occasions out of the five trials, as depicted in Fig. 9(d). It is noteworthy that a lower value of w_{max} is associated with improved MPPI performance. For instance, setting w_{max} to $1\,\mathrm{rad/s}$ results in the successful completion of all trials without any collisions. These outcomes emphasize the enhanced safety delivered by our U-MPPI method without imposing excessive constraints on control inputs.

VI. REAL-WORLD DEMONSTRATION

In this section, we conduct experiments to demonstrate the effectiveness and practicality of the proposed control strategies in achieving *real-time*, collision-free 2D navigation within an unknown and cluttered indoor environment.

A. Navigation in Unknown Corridor Environment

1) Experimental Setup and Validation Environment: In our experimental configuration, we utilize an autonomous Clearpath Jackal robot equipped with a 16-beam Velodyne LiDAR sensor, which serves the purposes of generating the local costmap and estimating the robot's pose through the Lidar Odometry and Mapping (LOAM) algorithm [44].

Additionally, we employ the simulation setup previously outlined in Sections V-A1 and V-B1, with two specific modifications: (i) we adjust the maximum speed v_{max} to $1.3\,\mathrm{m/s}$ to mitigate the robot localization error associated with using the Velodyne as a source of localization, and (ii) we extend the time prediction to $8\,\mathrm{s}$, an increase from the $6\,\mathrm{s}$ duration specified in Section V-B1. We conduct experimental validation in an indoor corridor cluttered environment, measuring $26\,\mathrm{m} \times 2.4\,\mathrm{m}$ and containing randomly placed, varying-sized box-like obstacles, as depicted in Fig. 10. In this setting, the robot's assigned control mission is to navigate from the following desired poses (in [m], [m], [deg]): $G_1 = [0, 0, 0]^T$, $G_2 = [26, 0.3, 170]^T$, and then return to G_1.

2) Performance Metrics: We adhere to the predefined set of performance metrics detailed in Section V-A3. Furthermore,

Fig. 10: Snapshot of our Jackal robot equipped with a Velodyne VLP-16 LiDAR and located in an indoor unknown cluttered corridor environment.

TABLE IV: Performance statistics of the two control strategies over six trials in an indoor corridor environment.

Indicator	MPPI	U-MPPI
$\mathcal{R}_{lm}$ ($\mathcal{N}_c$)	0 (1)	0 (0)
d_{av} [m]	56.58 ± 0.33	56.01 ± 0.46
v_{av} [m/s]	1.01 ± 0.32	1.02 ± 0.28
$t_{exec.}$ [ms]	9.07 ± 0.40	10.78 ± 0.42
J_{acc} [m^2/s^3]	257.75 ± 62.99	228.15 ± 22.01
ζ_{acc} [rad^2/s^3]	306.91 ± 15.74	263.02 ± 25.10
$\mathcal{E}_x$ [m] w.r.t. $\mathcal{F}_o$	0.38 ± 0.67	0.10 ± 0.28
$\mathcal{E}_y$ [m] w.r.t. $\mathcal{F}_o$	0.30 ± 0.34	-0.04 ± 0.04
$\mathcal{E}_\theta$ [deg] w.r.t. $\mathcal{F}_o$	7.20 ± 11.65	0.80 ± 3.71

given the significant uncertainties and noise inherent in real-world conditions (e.g., caused by LOAM) and the presence of unknown environmental factors, the robot's motion tends to be less smooth than in simulations. To facilitate a more thorough performance evaluation between the two proposed control methods, we opt to calculate the cumulative linear and angular jerks (J_{acc}, ζ_{acc}). Jerk, which represents the time derivative of acceleration, is linked to sudden changes in the forces exerted by the vehicle's actuators. Consequently, we can quantify the smoothness of the robot's velocity and steering by examining jerk as a metric [45]:

$$J_{acc} = \frac{1}{T_{tot}} \int_0^{T_{tot}} [\ddot{v}(t)]^2 dt, \tag{20}$$

$$\zeta_{acc} = \frac{1}{T_{tot}} \int_0^{T_{tot}} [\ddot{w}(t)]^2 dt, \tag{21}$$

where T_{tot} represents the complete duration required for the robot to execute its control mission. Additionally, we introduce the variable $\mathcal{E}_{loc.} = [\mathcal{E}_x, \mathcal{E}_y, \mathcal{E}_\theta]^\top$, which signifies the average drift or localization error in the robot's pose relative to the world frame $\mathcal{F}_o$, as displayed in Fig. 11.

3) Experimental Results: The performance statistics comparison between the two control strategies, MPPI and U-MPPI, across six trials in our indoor corridor environment is summarized in Table IV. Throughout all trials, U-MPPI consistently outperforms MPPI in collision avoidance, ensuring collision-free navigation with zero collisions, whereas MPPI averages one collision. While both control strategies exhibit similar performance in terms of the average distance traveled d_{av} by the robot, average linear velocities v_{av}, and *real-time* performance guarantee (with $t_{exec.} < 33.33\,\mathrm{ms}$), U-MPPI demonstrates a slightly longer average execution time per iteration $t_{exec.}$ when compared to MPPI, due to its incorporation of both the local costmap and the unscented transform into the U-MPPI optimization problem. It is noteworthy to observe that U-MPPI showcases smoother motion, as indicated by lower cumulative linear jerk J_{acc} at $228.15\,\mathrm{m}^2/\mathrm{s}^3$ and angular jerk ζ_{acc} at $263.02\,\mathrm{rad}^2/\mathrm{s}^3$, whereas MPPI exhibits higher jerk values ($257.75\,\mathrm{m}^2/\mathrm{s}^3$ and $306.91\,\mathrm{rad}^2/\mathrm{s}^3$, respectively). As a result of this enhanced motion smoothness, U-MPPI demonstrates improved robot localization accuracy based on LOAM, with smaller errors in both the x-direction ($\mathcal{E}_x = 0.1\,\mathrm{m}$) and the y-direction ($\mathcal{E}_y = -0.04\,\mathrm{m}$), relative to the reference frame $\mathcal{F}_o$, compared to MPPI ($0.38\,\mathrm{m}$ and $0.30\,\mathrm{m}$, respectively). However, MPPI has a slightly higher
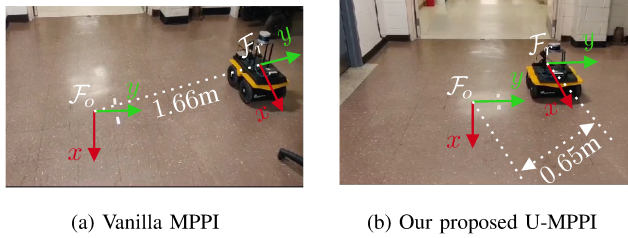
(a) Vanilla MPPI (b) Our proposed U-MPPI

Fig. 11: Visualization of the cumulative errors in the robot's pose estimation generated by LOAM for the worst trial when employing (a) MPPI and (b) U-MPPI.

orientation error, specifically $\mathcal{E}_\theta = 7.20\,\mathrm{deg}$, compared to $\mathcal{E}_\theta = 0.80\,\mathrm{deg}$ when U-MPPI is utilized. Figure 11 depicts the accumulated robot pose estimation errors generated by LOAM after completing the control mission for the worst trial among the six conducted trials for both control strategies. Thanks to incorporating uncertainty information throughout the trajectory evaluation process, U-MPPI can avoid aggressive control sequences (motions) that may, in sequence, negatively impact LOAM localization accuracy. These findings emphasize the benefits of U-MPPI in terms of collision avoidance, motion smoothness, and localization accuracy, all while recognizing the trade-off of a slightly longer execution time. More details about both simulation and experimental results are included in this video: https://youtu.be/1xsh4BxIrng.

B. Discussion

This section offers a thorough analysis of the execution times and performance metrics of our proposed control strategies, emphasizing disparities in U-MPPI performance compared to MPPI and proposing methods to enhance its effectiveness. Additionally, it outlines strategies for dynamic obstacle integration to improve navigation robustness.

1) U-MPPI vs. Vanilla MPPI Execution Times: Let us elaborate on why integrating the Unscented Transform (UT) and the risk-sensitive cost function does not significantly involve additional computational complexity for the U-MPPI optimization problem, resulting in average execution times $t_{\mathrm{exec.}}$ that are nearly equivalent to, and in some cases better than, those of the MPPI algorithm. The primary reasons for this improvement are two-fold. First, the U-MPPI algorithm leverages the UT method, a non-linear transformation method that efficiently handles higher-dimensional state spaces, thereby reducing computational complexity and allowing for parallel processing, all contributing to faster execution times. Second, the key strength of our algorithmic structure detailed in Algorithm 1 is the parallel sampling of M_σ sets of batches on the GPU, with each batch containing n_σ trajectories corresponding to the n_σ sigma points, and each batch employing a single thread. Such an algorithmic structure results in the utilization of fewer threads, specifically $int(\frac{M}{n_\sigma})$, in contrast to the MPPI algorithm, which allocates one thread per trajectory [13]. Consequently, it reduces computational overhead by minimizing context-switching and efficiently utilizing GPU resources. These factors clarify why the average execution times $t_{\mathrm{exec.}}$ of the U-MPPI outperform those of the MPPI algorithm

in the intensive simulations presented in Table I. This can be traced back to the relatively small size of the obstacle costmap, which did not significantly impact the workload per thread. Furthermore, efficient memory management for these maps ensures optimal utilization of the limited GPU memory, further contributing to faster execution times. It is also noteworthy in Table I that the execution time decreases with the reduction in the map's size. This is evident in *Scenario #1*, where the presence of numerous obstacles leads to a larger map size, demanding more memory allocation and management. In contrast, *Scenario #3* features a smaller map size, leading to less memory usage and consequently faster execution times. Nevertheless, it is crucial to recognize that in Tables III and IV, where the local costmap with dimensions of $240\,\mathrm{cell} \times 240\,\mathrm{cell}$ built by the onboard sensor is incorporated into the optimization problem, the average execution times of U-MPPI were slightly longer than those of MPPI. This discrepancy can be attributed to the substantial dimensions of the costmap, which impose a higher workload per thread, thereby increasing latency in trajectory generation and necessitating more memory allocation and management than in the simulations presented in Table I. For further details on mitigating the increased workload and ensuring real-time performance, refer to Section IV-D2, which discusses several key factors.

2) Performance Metrics Evaluation: Through the intensive simulations conducted in Sections V and VI, we have observed that the metrics of task completion percentage $\mathcal{T}_{\mathrm{c}}$ and success rate $\mathcal{S}_R$, number of collisions $\mathcal{N}_{\mathrm{c}}$ and local minima occurrences $\mathcal{R}_{\mathrm{lm}}$, average traveled distance d_{av} by the robot, and average robot speed v_{av} are pivotal in evaluating the effectiveness of the proposed control strategy. These metrics specifically assess the ability to generate safe, efficient, and robust trajectories for autonomous vehicle navigation in cluttered environments under various conditions. The results unequivocally demonstrate that our proposed U-MPPI control strategy outperforms the vanilla MPPI in providing safer and more reliable navigation. This superiority is evidenced by higher success and task completion rates, along with a reduced number of collisions and occurrences of local minima. However, one might argue that this improvement does not extend to the average traveled distance d_{av} and average linear speed v_{av}. While it is true that U-MPPI may not always achieve the shortest path, as observed in *Scenarios #1 and #2* in Table I and *Forest #1* in Table III, it should be emphasized that the average distance is based solely on successful tasks, with MPPI having fewer successful tasks than U-MPPI. This affects the comparison of average distances and indicates that U-MPPI outperforms MPPI in terms of overall route completion and reliability, demonstrating greater effectiveness in achieving collision-free navigation. To further enhance U-MPPI's trajectory quality, specifically by reducing the average traveled distance, we have proposed several methods, including:

(i) reducing the collision weighting coefficient w_{crash}, which has no significant impact on the success rate (see Fig. 6(b) and Tests #1 and #2 in Table II),

(ii) assigning a higher value to k_σ, which also slightly re-

duced the occurrences of local minima from 2 to 1 in *Scenario #1* (see Test #6 in Table II),

(iii) setting $\gamma < 0$, which makes the control strategy more aggressive in attempting to reach the desired state, as explained in Section IV-B, thereby improving the quality of the robot trajectory. For instance, in *Scenario #2*, simulations were conducted with $\gamma = -1$ instead of $\gamma = 1$. This adjustment demonstrated improved performance in the robot's average travel distance d_{av} (namely, $d_{\text{av}} = 74.8\text{m}$), thereby outperforming both U-MPPI and MPPI, as indicated in Table I. Moreover, this configuration achieved a perfect task completion rate of 100%.

Concerning the second argument regarding the improvement in the average linear velocity, it is noteworthy that the MPPI algorithm slightly outperforms the U-MPPI algorithm in only one scenario: *Forest #1*, as reported in Table III. However, it achieves marginally higher average speeds in the remaining tests, as indicated in Tables I, III, and IV. The limited improvements in average linear velocity are primarily due to the risk-sensitive cost function employed by U-MPPI, which prioritizes safer, more robust trajectories, inherently resulting in more conservative control actions with $\gamma > 0$. Additionally, the integration of the UT for state propagation allows for comprehensive state-space exploration by considering both the mean and covariance of the system dynamics, leading to the selection of more cautious trajectories that ensure safety but may not be the fastest. This might result in fluctuations in the velocity profile of U-MPPI, especially when navigating at high speed in extremely crowded environments, as discussed at the end of Section V-A4.

3) Dynamic Obstacles Integration: While our current study evaluates the U-MPPI control strategy in static environments to establish foundational performance metrics, navigating dynamic environments, particularly those shared with humans or other agents, necessitates enhanced safety measures. Despite achieving zero collisions ($\mathcal{N}_c$) in all simulations and real-world validations, the challenge of dynamic obstacle avoidance remains. For instance, our previous work [22] demonstrated successful avoidance of moving agents, such as eight pedestrians, in 2D grid-based navigation scenarios. However, an increase in the number of agents led to a noisier 2D costmap, raising the risk of the vehicle being trapped in local minima and compromising safety. Given our current reliance on a 2D grid for collision avoidance, similar behaviors and safety concerns are anticipated. Therefore, to tackle this issue and ensure safety, an alternative solution involves enhancing our control strategy by incorporating formulations similar to equation (17) in [19], which adjusts the cost function to better account for collision risks, or by applying chance constraints [6] to provide probabilistic guarantees of collision avoidance. These measures aim to ensure safety in highly dynamic scenarios, further improving the reliability of our autonomous navigation system.

VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed the U-MPPI control strategy, a novel methodology that enhances the vanilla MPPI algorithm by leveraging the unscented transform for two primary objectives. Firstly, it regulates the propagation of the dynamical system, resulting in a more effective sampling distribution policy that effectively propagates both the mean $\bar{\mathbf{x}}_k$ and covariance $\mathbf{\Sigma}_k$ of the state vector $\mathbf{x}_k$ at each time-step k. Secondly, it incorporates a *risk-sensitive* cost function that explicitly accounts for risk or uncertainty throughout the trajectory evaluation process. Through extensive simulations and real-world demonstrations, we demonstrated the effectiveness of U-MPPI in achieving aggressive collision-free navigation in both known and unknown cluttered environments. By comparing it to MPPI, our approach accomplished a substantial improvement in state-space exploration while utilizing the same injected Gaussian noise $\delta\mathbf{u}_k$ in the mean control sequence. As a result, it yielded higher success and task completion rates, effectively minimizing the likelihood of getting trapped in local minima, and enabling the robot to identify feasible trajectories that avoid collisions. Our future plan involves incorporating chance constraints into the U-MPPI control architecture to effectively address uncertainties in system dynamics and the environment, including moving obstacles, resulting in an enhanced safety and robustness of the control system to handle uncertain conditions, especially in safety-critical applications. Moreover, we intend to extend our experimental validation to include UAVs, mobile manipulators, and autonomous ground vehicles with higher-dimensional state spaces. This will provide empirical evidence of the algorithm's performance in these contexts, demonstrating its robustness and efficiency in handling more complex systems.

REFERENCES

[1] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, "Motion planning under uncertainty for robotic tasks with long time horizons," *The International Journal of Robotics Research*, vol. 30, no. 3, pp. 308–323, 2011.

[2] J. Mattingley, Y. Wang, and S. Boyd, "Receding horizon control," *IEEE Control Systems Magazine*, vol. 31, no. 3, pp. 52–65, 2011.

[3] J. Löfberg, *Minimax approaches to robust model predictive control*. Linköping University Electronic Press, 2003, vol. 812.

[4] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control*. Springer, 2007, pp. 207–226.

[5] A. Mesbah, "Stochastic model predictive control: An overview and perspectives for future research," *IEEE Control Systems Magazine*, vol. 36, no. 6, pp. 30–44, 2016.

[6] N. E. Du Toit and J. W. Burdick, "Probabilistic collision checking with chance constraints," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 809–815, 2011.

[7] T. Knudsen and J. Leth, "Stochastic MPC using the unscented transform," in *Annual American Control Conference (ACC)*, 2018, pp. 4718–4724.

[8] L. Hewing and M. N. Zeilinger, "Stochastic model predictive control for linear systems using probabilistic reachable sets," in *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 5182–5188.

[9] S. A. Sajadi-Alamdari, H. Voos, and M. Darouach, "Risk-averse stochastic nonlinear model predictive control for real-time safety-critical systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 5991–5997, 2017.

[10] X. Yang and J. Maciejowski, "Risk-sensitive model predictive control with gaussian process models," *IFAC-PapersOnLine*, vol. 48, no. 28, pp. 374–379, 2015.

[11] E. Hyeon, Y. Kim, and A. G. Stefanopoulou, "Fast risk-sensitive model predictive control for systems with time-series forecasting uncertainties," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 2515–2520.

[12] M. Schuurmans, A. Katriniok, H. E. Tseng, and P. Patrinos, "Learning-based risk-averse model predictive control for adaptive cruise control with stochastic driver models," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 128–15 133, 2020.

[13] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

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

[15] I. S. Mohamed, G. Allibert, and P. Martinet, "Model predictive path integral control framework for partially observable navigation: A quadrotor case study," in *16th Int. Conf. on Control, Automation, Robotics and Vision (ICARCV)*, Shenzhen, China, Dec. 2020, pp. 196–203.

[16] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou, "$\mathcal{L}_1$-adaptive MPPI architecture for robust and agile control of multirotors," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020, pp. 7661–7666.

[17] I. S. Mohamed, G. Allibert, and P. Martinet, "Sampling-based MPC for constrained vision based control," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2021, pp. 3753–3758.

[18] M. Kazim, J. Hong, M.-G. Kim, and K.-K. K. Kim, "Recent advances in path integral control for trajectory optimization: An overview in theoretical and algorithmic perspectives," *Annual Reviews in Control*, vol. 57, p. 100931, 2024.

[19] I. S. Mohamed, "MPPI-VS: Sampling-based model predictive control strategy for constrained image-based and position-based visual servoing," *arXiv preprint arXiv:2104.04925*, 2021.

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

[21] J. Yin, Z. Zhang, E. Theodorou, and P. Tsiotras, "Trajectory distribution control for model predictive path integral control using covariance steering," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2022, pp. 1478–1484.

[22] I. S. Mohamed, K. Yin, and L. Liu, "Autonomous navigation of AGVs in unknown cluttered environments: log-MPPI control strategy," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10240–10247, 2022.

[23] G. Williams, B. Goldfain, P. Drews, K. Saigol, J. M. Rehg, and E. A. Theodorou, "Robust sampling based model predictive control with sparse objective information," in *Robotics: Science and Systems*, Pittsburgh, Pennsylvania, USA, 2018, pp. 42—-51.

[24] I. S. Mohamed, M. Ali, and L. Liu, "GP-guided MPPI for efficient navigation in complex unknown cluttered environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 7463–7470.

[25] F. Rastgar, H. Masnavi, B. Sharma, A. Aabloo, J. Swevers, and A. K. Singh, "PRIEST: Projection guided sampling-based optimization for autonomous navigation," *IEEE Robotics and Automation Letters*, 2024.

[26] M. Gandhi, H. Almubarak, and E. Theodorou, "Safe importance sampling in model predictive path integral control," *arXiv preprint arXiv:2303.03441*, 2023.

[27] D. M. Asmar, R. Senanayake, S. Manuel, and M. J. Kochenderfer, "Model predictive optimized path integral strategies," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 3182–3188.

[28] Y. Qu, H. Chu, S. Gao, J. Guan, H. Yan, L. Xiao, S. E. Li, and J. Duan, "RL-driven MPPI: Accelerating online control laws calculation with offline policy," *IEEE Transactions on Intelligent Vehicles*, 2023.

[29] L. Yan and S. Devasia, "Output-sampled model predictive path integral control (o-MPPI) for increased efficiency," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[30] E. Trevisan and J. Alonso-Mora, "Biased-MPPI: Informing sampling-based model predictive control by fusing ancillary controllers," *arXiv preprint arXiv:2401.09241*, 2024.

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

[32] Z. Wang, O. So, K. Lee, and E. A. Theodorou, "Adaptive risk sensitive model predictive control with stochastic search," in *Learning for Dynamics and Control*. PMLR, 2021, pp. 510–522.

[33] J. Yin, Z. Zhang, and P. Tsiotras, "Risk-aware model predictive path integral control using conditional value-at-risk," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 7937–7943.

[34] I. M. Ross, R. J. Proulx, and M. Karpenko, "Unscented guidance," in *American Control Conference (ACC)*. IEEE, 2015, pp. 5605–5610.

[35] A. Savitzky and M. J. Golay, "Smoothing and differentiation of data by simplified least squares procedures." *Analytical chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[36] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte, "A new approach for filtering nonlinear systems," in *Proceedings of 1995 American Control Conference-ACC'95*, vol. 3. IEEE, 1995, pp. 1628–1632.

[37] J. Xu, K. Yin, and L. Liu, "Online planning in uncertain and dynamic environment in the presence of multiple mobile vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 2410–2416.

[38] R. Van Der Merwe, "Sigma-point kalman filters for probabilistic inference in dynamic state-space models," Ph.D. dissertation, Oregon Health & Science University, 2004.

[39] S. J. Julier, "The scaled unscented transformation," in *American Control Conference*, vol. 6. IEEE, 2002, pp. 4555–4559.

[40] I. M. Ross, R. J. Proulx, and M. Karpenko, "Unscented optimal control for space flight," in *International Symposium on Space Flight Dynamics (ISSFD)*, 2014, pp. 1–12.

[41] N. Ozaki, S. Campagnola, and R. Funase, "Tube stochastic optimal control for nonlinear constrained trajectory optimization problems," *Journal of Guidance, Control, and Dynamics*, 2020.

[42] P. Whittle, "Risk-sensitive linear/quadratic/gaussian control," *Advances in Applied Probability*, vol. 13, no. 4, pp. 764–777, 1981.

[43] E. Marder-Eppstein, D. V. Lu!!, and D. Hershberger. Costmap_2d package. [Online]. Available: http://wiki.ros.org/costmap_2d

[44] J. Zhang and S. Singh, "LOAM: Lidar odometry and mapping in real-time." in *Robotics: Science and Systems*, vol. 2, no. 9. Berkeley, CA, 2014, pp. 1–9.

[45] M. Ali and L. Liu, "GP-frontier for local mapless navigation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 10047–10053.

[46] M. A. Woodbury, *Inverting modified matrices*. Department of Statistics, Princeton University, 1950.

APPENDIX A
DERIVATIVE OF *Risk-Sensitive* COST

Considering the *risk-sensitive* cost defined as:

$$q_{\mathrm{rs}}(\mathbf{x}_k) = -\frac{2}{\gamma} \log \mathbb{E}\left[\exp\left(-\frac{1}{2}\gamma \|\mathbf{x}_k - \mathbf{x}_f\|_Q^2\right)\right], \quad (22)$$

our objective is to obtain the following formulation:

$$q_{\mathrm{rs}}(\mathbf{x}_k) = \frac{1}{\gamma} \log \det\left(\mathbf{I} + \gamma Q \boldsymbol{\Sigma}_k\right) + \|\bar{\mathbf{x}}_k - \mathbf{x}_f\|_{Q_{\mathrm{rs}}}^2, \quad (23)$$

where $Q_{\mathrm{rs}} = \left(Q^{-1} + \gamma \boldsymbol{\Sigma}_k\right)^{-1}$, Q is a diagonal matrix, and $\boldsymbol{\Sigma}_k$ is a symmetric matrix. We assume that the system state $\mathbf{x}_k$ follows a Gaussian distribution $\mathcal{N}(\bar{\mathbf{x}}_k, \boldsymbol{\Sigma}_k)$, and $\mathbf{x}_f$ represents the reference (desired) state. To derive the expression for $q_{\mathrm{rs}}(\mathbf{x}_k)$ as presented in (23), we will proceed with a step-by-step derivation as follows. We will start by expanding the quadratic form in (22):

$$\begin{aligned} q_{\mathrm{rs}}(\mathbf{x}_k) &= -\frac{2}{\gamma} \log \mathbb{E}\left[\exp\left\{-\frac{\gamma}{2} \|\mathbf{x}_k - \mathbf{x}_f\|_Q^2\right\}\right] \\ &= -\frac{2}{\gamma} \log \mathbb{E}\left[\exp\left\{-\frac{\gamma}{2}\left(\mathbf{x}_k - \mathbf{x}_f\right)^\top Q\left(\mathbf{x}_k - \mathbf{x}_f\right)\right\}\right]. \end{aligned}$$
$$(24)$$

Since $\mathbf{x}_k \sim \mathcal{N}(\bar{\mathbf{x}}_k, \boldsymbol{\Sigma}_k)$, we can assume that $\mathbf{x}_k = \bar{\mathbf{x}}_k + \mathbf{e}_k$, where $\mathbf{e}_k \sim \mathcal{N}(0, \boldsymbol{\Sigma}_k)$. Therefore, by substituting $\mathbf{x}_k$ in (24), we can express the expectation as follows:

$$\begin{aligned} q_{\mathrm{rs}} &= -\frac{2}{\gamma} \log \mathbb{E}\left[\exp\left\{-\frac{\gamma}{2}\overline{\mathbf{X}}_k^\top Q\overline{\mathbf{X}}_k - \gamma \mathbf{e}_k^\top Q\overline{\mathbf{X}}_k - \frac{\gamma}{2}\mathbf{e}_k^\top Q\mathbf{e}_k\right\}\right] \\ &= -\frac{2}{\gamma} \log \mathbb{E}\left[\exp\left\{-\frac{\gamma}{2}\overline{\mathbf{X}}_k^\top Q\overline{\mathbf{X}}_k\right\}\right. \\ &\qquad \left. \times \exp\left\{-\gamma \mathbf{e}_k^\top Q\overline{\mathbf{X}}_k - \frac{\gamma}{2}\mathbf{e}_k^\top Q\mathbf{e}_k\right\}\right] \\ &= \left\|\overline{\mathbf{X}}_k\right\|_Q^2 - \frac{2}{\gamma} \log \mathbb{E}\left[\exp\left\{-\frac{\gamma}{2}\|\mathbf{e}_k\|_Q^2 - \gamma \mathbf{e}_k^\top Q\overline{\mathbf{X}}_k\right\}\right], \end{aligned}$$
$$(25)$$

where $\overline{\mathbf{X}}_k = \bar{\mathbf{x}}_k - \mathbf{x}_f$. Now, to compute the expectation $\mathbb{E}[\cdot]$ in (25), given by

$$-\frac{2}{\gamma} \log \mathbb{E} \left[\exp \left\{ -\frac{\gamma}{2} \mathbf{e}_k^\top Q \mathbf{e}_k - \gamma \mathbf{e}_k^\top Q \overline{\mathbf{X}}_k \right\} \right], \quad (26)$$

we introduce a new term $\mathbf{y}_k$ to simplify and express the expectation in the quadratic form. To achieve this, let us define $\mathbf{y}_k = -\gamma \mathbf{K}^{-1} Q \overline{\mathbf{X}}_k$, where $\mathbf{K} = \gamma Q + \Sigma_k^{-1}$. By using this definition, we can accordingly prove that

$$\frac{1}{2} \left(\mathbf{e}_k - \mathbf{y}_k \right)^\top \mathbf{K} \left(\mathbf{e}_k - \mathbf{y}_k \right) - \frac{1}{2} \mathbf{y}_k^\top \mathbf{K} \mathbf{y}_k$$
$$= \frac{\gamma}{2} \mathbf{e}_k^\top Q \mathbf{e}_k + \frac{1}{2} \mathbf{e}_k^\top \Sigma_k^{-1} \mathbf{e}_k + \gamma \mathbf{e}_k^\top Q \overline{\mathbf{X}}_k. \quad (27)$$

Building on the equivalence in (27), we can approximate the expectation in (26) by integrating with respect to the distribution of the random variable $\mathbf{e}_k$. Since $\mathbf{e}_k \sim \mathcal{N}(\mathbf{0}, \Sigma_k)$, we can express it as follows:

$$-\frac{2}{\gamma} \log \mathbb{E} \left[\exp \left\{ -\frac{\gamma}{2} \mathbf{e}_k^\top Q \mathbf{e}_k - \gamma \mathbf{e}_k^\top Q \overline{\mathbf{X}}_k \right\} \right]$$
$$= -\frac{2}{\gamma} \log \left(\int_{\mathbb{R}^n} \exp \left\{ \frac{-1}{2} \left(\mathbf{e}_k - \mathbf{y}_k \right)^\top \mathbf{K} \left(\mathbf{e}_k - \mathbf{y}_k \right) \right. \right. \quad (28)$$
$$\left. \left. + \frac{1}{2} \mathbf{y}_k^\top \mathbf{K} \mathbf{y}_k + \frac{1}{2} \mathbf{e}_k^\top \Sigma_k^{-1} \mathbf{e}_k \right\} p(\mathbf{e}_k) \mathrm{d}\mathbf{e}_k \right),$$

where $p(\mathbf{e}_k) = \frac{1}{\sqrt{\mathbf{A}}} \exp \left\{ -\frac{1}{2} \mathbf{e}_k^\top \Sigma_k^{-1} \mathbf{e}_k \right\}$ represents the probability density function of $\mathbf{e}_k$. By substituting $p(\mathbf{e}_k)$ in (28) and simplifying it further, we obtain:

$$-\frac{2}{\gamma} \log \mathbb{E} \left[\exp \left\{ -\frac{\gamma}{2} \mathbf{e}_k^\top Q \mathbf{e}_k - \gamma \mathbf{e}_k^\top Q \overline{\mathbf{X}}_k \right\} \right]$$
$$= -\frac{2}{\gamma} \log \left(\frac{1}{\sqrt{\mathbf{A}}} \int_{\mathbb{R}^n} \exp \left\{ -\frac{1}{2} \left(\mathbf{e}_k - \mathbf{y}_k \right)^\top \mathbf{K} \left(\mathbf{e}_k - \mathbf{y}_k \right) \right. \right.$$
$$\left. \left. + \frac{1}{2} \mathbf{y}_k^\top \mathbf{K} \mathbf{y}_k \right\} \mathrm{d}\mathbf{e}_k \right)$$
$$= -\frac{2}{\gamma} \log \left(\frac{1}{\sqrt{\mathbf{A}}} \exp \left(\frac{1}{2} \mathbf{y}_k^\top \mathbf{K} \mathbf{y}_k \right) \right.$$
$$\left. \times \int_{\mathbb{R}^n} \exp \left\{ \frac{-1}{2} \left(\mathbf{e}_k - \mathbf{y}_k \right)^\top \mathbf{K} \left(\mathbf{e}_k - \mathbf{y}_k \right) \right\} \mathrm{d}\mathbf{e}_k \right), \quad (29)$$

where $\mathbf{A} = (2\pi)^n \det \Sigma_k$. We can now evaluate the integral. Since the integrand is a Gaussian distribution with mean $\mathbf{y}_k$ and covariance matrix $\mathbf{K}^{-1}$, the integral evaluates to $\sqrt{\frac{(2\pi)^n}{\det \mathbf{K}}}$. Substituting this back into the expression given in (29), we have:

$$-\frac{2}{\gamma} \log \mathbb{E} \left[\exp \left\{ -\frac{\gamma}{2} \mathbf{e}_k^\top Q \mathbf{e}_k - \gamma \mathbf{e}_k^\top Q \overline{\mathbf{X}}_k \right\} \right]$$
$$= -\frac{2}{\gamma} \log \left(\frac{\exp \left(\frac{1}{2} \mathbf{y}_k^\top \mathbf{K} \mathbf{y}_k \right)}{\sqrt{(2\pi)^n \det \Sigma_k}} \sqrt{\frac{(2\pi)^n}{\det \mathbf{K}}} \right)$$
$$= -\frac{2}{\gamma} \log \left(\frac{\exp \left(\frac{1}{2} \mathbf{y}_k^\top \mathbf{K} \mathbf{y}_k \right)}{\sqrt{\det(\Sigma_k \mathbf{K})}} \right) \quad (30)$$
$$= \frac{1}{\gamma} \log \det(\Sigma_k \mathbf{K}) - \frac{1}{\gamma} \mathbf{y}_k^\top \mathbf{K} \mathbf{y}_k$$
$$= \frac{1}{\gamma} \log \det \left(\mathbf{I} + \gamma Q \Sigma_k \right) - \frac{1}{\gamma} \mathbf{y}_k^\top \mathbf{K} \mathbf{y}_k.$$

By substituting $\mathbf{y}_k = -\gamma \mathbf{K}^{-1} Q \overline{\mathbf{X}}_k$, and $\mathbf{K} = \gamma Q + \Sigma_k^{-1}$ into (30), we can simplify the second term $-\frac{1}{\gamma} \mathbf{y}_k^\top \mathbf{K} \mathbf{y}_k$ as follows:

$$-\frac{1}{\gamma} \mathbf{y}_k^\top \mathbf{K} \mathbf{y}_k = -\gamma \overline{\mathbf{X}}_k^\top Q \left(\gamma Q + \Sigma_k^{-1} \right)^{-1} Q \overline{\mathbf{X}}_k, \quad (31)$$

assuming that Q is a diagonal matrix and Σ_k is a symmetric matrix. Afterward, we can utilize the Woodbury matrix identity to simplify the expression $\left(\gamma Q + \Sigma_k^{-1} \right)^{-1}$, where the Woodbury matrix identity is defined as [46]:

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}. \quad (32)$$

By applying the Woodbury identity expressed in (32) to the given expression with $A = \gamma Q$, $C = \Sigma_k^{-1}$, and $U = V = \mathbf{I}$ (where $\mathbf{I}$ denotes the identity matrix), we can rewrite $\left(\gamma Q + \Sigma_k^{-1} \right)^{-1}$ as follows:

$$\left(\gamma Q + \Sigma_k^{-1} \right)^{-1} = \frac{1}{\gamma} Q^{-1} \left[\mathbf{I} - Q_{\mathrm{rs}} Q^{-1} \right], \quad (33)$$

where $Q_{\mathrm{rs}} = \left(Q^{-1} + \gamma \Sigma_k \right)^{-1}$. By substituting (33) back into (31), we obtain:

$$-\frac{1}{\gamma} \mathbf{y}_k^\top \mathbf{K} \mathbf{y}_k = - \left\| \overline{\mathbf{X}}_k \right\|_Q^2 + \left\| \overline{\mathbf{X}}_k \right\|_{Q_{\mathrm{rs}}}^2. \quad (34)$$

Subsequently, by substituting (34) into (30) and further substituting the obtained results into (25), while replacing $\overline{\mathbf{X}}_k$ with $\bar{\mathbf{x}}_k - \mathbf{x}_f$, the desired result can be obtained as follows:

$$q_{\mathrm{rs}}(\mathbf{x}_k) = \frac{1}{\gamma} \log \det \left(\mathbf{I} + \gamma Q \Sigma_k \right) + \left\| \bar{\mathbf{x}}_k - \mathbf{x}_f \right\|_{Q_{\mathrm{rs}}}^2. \quad (35)$$

Remarkably, as γ approaches zero, the term $\frac{1}{\gamma} \log \det(\mathbf{I} + \gamma Q \Sigma_k)$ approximates $\mathrm{Tr}(Q \Sigma_k)$, avoiding the singularity that occurs if $\gamma = 0$. Therefore, as $\gamma \to 0$, $q_{\mathrm{rs}}(\mathbf{x}_k)$ simplifies to:

$$q_{\mathrm{rs}}(\mathbf{x}_k) = \mathrm{Tr}(Q \Sigma_k) + \left\| \bar{\mathbf{x}}_k - \mathbf{x}_f \right\|_{Q_{\mathrm{rs}}}^2,$$

where $Q_{\mathrm{rs}} = Q$ and $\mathrm{Tr}(\cdot)$ denotes the trace of a matrix. It is worth emphasizing that a similar form of (35), along with its derivation, can be found in [11]. However, the presented derivation process in [11] is brief and lacks detailed explanations and clarifications.