CoRL-MPPI: Enhancing MPPI With Learnable Behaviours For Efficent And Provably-Safe Multi-Robot Collision Avoidance

Stepan Dergachev, Artem Pshenitsyn, Aleksandr Panov, Alexey Skrynnik, Konstantin Yakovlev

***Abstract*— Decentralized collision avoidance remains a core challenge for scalable multi-robot systems. One of the promising approaches to tackle this problem is Model Predictive Path Integral (MPPI) – a framework that is naturally suited to handle any robot motion model and provides strong theoretical guarantees. Still, in practice MPPI-based controller may provide suboptimal trajectories as its performance relies heavily on uninformed random sampling. In this work, we introduce CoRL-MPPI, a novel fusion of Cooperative Reinforcement Learning and MPPI to address this limitation. We train an action policy (approximated as deep neural network) in simulation that learns local cooperative collision avoidance behaviors. This learned policy is then embedded into the MPPI framework to guide its sampling distribution, biasing it towards more intelligent and cooperative actions. Notably, CoRL-MPPI preserves all the theoretical guarantees of regular MPPI. We evaluate our approach in dense, dynamic simulation environments against state-of-the-art baselines, including ORCA, BVC, and a multi-agent MPPI implementation. Our results demonstrate that CoRL-MPPI significantly improves navigation efficiency (measured by success rate and makespan) and safety, enabling agile and robust multi-robot navigation.**

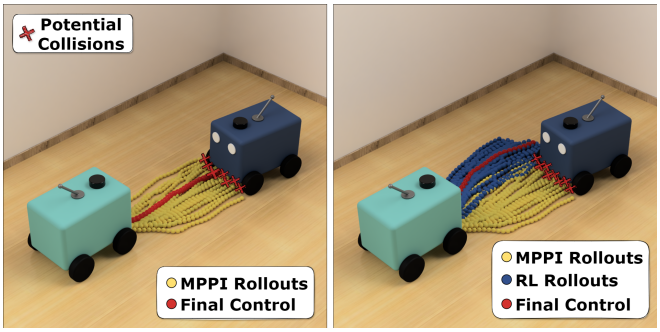

Fig. 1: The figure illustrates the core idea of our method for decentralized collision avoidance. The left panel shows the baseline MPPI controller, where random rollouts (yellow) lead to potential collisions (red crosses) and suboptimal control (red trajectory). The right panel depicts proposed fusion of RL and MPPI, where learned policy rollouts (blue) bias the sampling distribution toward more cooperative and collision-free behaviors, improving final control performance while retaining the theoretical guarantees of MPPI.

I. INTRODUCTION

The deployment of multi-robot systems in shared spaces promises significant boost in efficiency in such domains as warehouse logistics, search-and-rescue, disaster management etc. A fundamental problem in any multi-robot system is decentralized collision avoidance: each robot must navigate to its goal efficiently while ensuring safety by proactively avoiding conflicts with others, all without centralized coordination. This problem is inherently challenging due to the non-linear and dynamic nature of robot interactions, the curse of dimensionality as the number of agents increases, and the necessity for real-time computation under uncertainty.

Traditional approaches to this problem can be broadly categorized into reactive and planning-based methods. Reactive algorithms, such as Velocity Obstacles and its widely-used variant Optimal Reciprocal Collision Avoidance (ORCA) [1], compute collision-free velocities based on the current states of neighboring robots. While highly computationally efficient, these methods are inherently myopic. They operate on a one-step time horizon, which can lead to oscillatory behavior, deadlocks in congested scenarios, and a general lack of cooperation, as agents do not reason about the future intentions of their neighbors. Conversely, planning-based methods, like those employing Buffered Voronoi Cells (BVC) [2], define safe corridors for each agent. These approaches provide strong safety guarantees but can be overly conservative, often sacrificing optimality and agility for safety, leading to inefficient trajectories and longer travel times.

Model Predictive Control (MPC) frameworks offer a compelling middle ground by explicitly optimizing a short-term trajectory while accounting for future states. The Model Predictive Path Integral (MPPI) [3], a sampling-based variant of MPC, has gained a significant attention for its ability to handle non-linear systems and complex cost functions without the need for gradient computation. MPPI allows flexible formulation of both motion models and cost functions and has been extended in numerous works, including methods that incorporate safety guarantees in multi-agent scenarios [4], [5].

However, the performance of MPPI-based methods is critically dependent on the quality of its sampled trajectories. In its standard formulation, control sequences are drawn from a Gaussian distribution centered around a prior (often the previous solution). Such sampling may be very inefficient in complex multi-agent settings, where the vast majority of sampled trajectories may lead to uncooperative behavior. Consequently, even when the number of samples is high the resultant trajectories may be highly suboptimal leading to an overall degradation of the multi-robot system's performance. Generally, one may claim that MPPI in multi-robot navigation lacks a higher-level strategic understanding of multi-agent cooperation.

Meanwhile, recent advances in Reinforcement Learning

have demonstrated remarkable success in learning complex, cooperative behaviors directly from simulation. RL-based agents may learn implicit cooperation strategies, learning to anticipate the actions of other agents and negotiate passage efficiently. Yet, purely learned policies are often sensitive to distribution shifts and poor performance in new environments not seen during training. Moreover, they typically lack the guarantees provided by the model-based controllers.

In this work, we propose a novel hybrid approach to multi-robot collision avoidance, that leverages the complementary strengths of both paradigms. We introduce CoRL-MPPI – a framework where a pre-trained RL policy guides the sampling process of the MPPI controller. This policy encapsulates the high-level strategic knowledge of cooperative avoidance, and is intended to generate a sophisticated proposal distribution. Instead of sampling controls purely randomly, suggested method additionally uses a distribution of actions provided by it, thus biasing the search towards intelligent and cooperative trajectories – see Fig 1. Such a fusion creates a synergistic effect: the RL policy provides the strategic intuition of the cooperative behavior while MPPI provides the robust long-horizon planning with the full model dynamics with formal safety guarantees [5].

To summarize, the core contributions of this paper are:

- We introduce CoRL-MPPI, a novel hybrid architecture that integrates a learned RL policy into the MPPI control framework to guide its sampling distribution for decentralized multi-robot navigation.
- We provide theoretical justification that CoRL-MPPI preserves safety guarantees; moreover, when execution noise is present, control safety is maintained with a specified probability.
- A comprehensive empirical evaluation in simulation demonstrating that CoRL-MPPI outperforms classical approaches (ORCA, BVC) and MPPI in terms of success rate and navigation efficiency (makespan).

II. RELATED WORK

Next lines of research are most relevant to this work: *model predictive path integral*, *decentralized classical* and *learning-based multi-agent collision avoidance*.

a) ***Model Predictive Path Integral****:* The Model Predictive Path Integral (MPPI) algorithm was originally proposed as a sampling-based optimal control method [3] and later extended to Information-Theoretic MPC to handle general non-linear dynamics [6]. Since then, numerous extensions have been proposed to improve smoothness, robustness, and sampling efficiency, including SMPPI [7], CC-MPPI [8], Tube-MPPI [9], Robust-MPPI [10] and CBF-MPPI [11], [12].

Although most MPPI-based methods have been developed for single-agent systems, several studies have explored their adaptation to multi-agent scenarios. These works typically address dynamic collision avoidance, either assuming explicit inter-agent communication [13], [14], [15] or relying on motion prediction without formal safety guarantees [16]. A notable exception is the decentralized MPPI-ORCA framework [4], [5], which provides theoretical safety guarantees through the incorporation of ORCA-based constraints. However, this method still relies on the basic MPPI sampling scheme, which can lead to inefficiencies in complex interaction scenarios.

Recent research has also investigated the integration of reinforcement learning into the MPPI framework. For instance, TD-MPC [17] and TD-MPC2 [18]learn a latent dynamics model and a value function to perform short-horizon rollouts, achieving high sample efficiency but lacking formal safety guarantees. RL-driven MPPI [19] combines an offline RL policy with MPPI by using the learned policy for trajectories generation and the RL value function as a terminal cost. While this approach mitigates the reliance on random sampling, it remains restricted to single-agent settings, lacks safety guarantees, and heavily depends on the generalization capability of the underlying RL policy.

b) ***Multi-Agent Collision Avoidance****:* Multi-agent navigation with collision avoidance is an important problem in robotics. The main goal is to move several agents safely in a shared continuous space without collisions.

Velocity-based approaches define a set of admissible velocities that guarantee collision-free motion and then select the optimal velocity within this feasible region. One of the most established representatives of this class is the Optimal Reciprocal Collision Avoidance (ORCA) algorithm [1], which provides an efficient solution for reciprocal collision avoidance. Despite its popularity, ORCA does not take into account kinematic limitations of agents. To partially overcome these drawbacks, Snape et al. [20], [21] proposed modifying ORCA for differential-drive robots by expanding the effective agent radius, thereby indirectly modeling non-holonomic constraints. The Non-Holonomic ORCA (NH-ORCA) algorithm [22], [23] extends this concept by explicitly considering non-holonomic motion through the use of precomputed lookup tables that encode feasible velocities. Other variants, such as PRVO [24], CALU [25], and COCALU [26], incorporate uncertainty in localization and sensing into the velocity-obstacle framework.

Another family of methods, based on BVC [2], defines safe navigation zones by constructing buffered Voronoi regions around each agent. The PBVC algorithm [27] extends this principle to a decentralized formulation that accounts for perception uncertainty, but it neglects kinematic constraints. A more recent modification, B-UAVC [28], integrates positional uncertainty and can be adapted to different motion models. However, the general B-UAVC formulation relies on Model Predictive Control (MPC), requiring the development of a dedicated MPC controller consistent with the agent's dynamic model.

In addition, several learning-based methods [29], [30], [31], [32], [33] have been proposed to address the multi-agent collision avoidance problem. Some of them operate directly on raw sensory inputs (e.g., LiDAR) [30], [31], thereby eliminating the need for explicit state estimation. For instance, Han et al. [33] combine decentralized re-

inforcement learning with reciprocal velocity obstacles to reduce collisions. Nevertheless, such approaches lack formal safety guarantees and often rely purely on reactive behaviors learned from training data, which may fail to generalize to unseen scenarios.

III. PROBLEM STATEMENT

Consider a set of homogeneous robots (agents) denoted by $\mathscr{A} = \{1, 2, \ldots, N\}$, operating within a two-dimensional workspace $\mathscr{W} \subset \mathbb{R}^2$. Each robot is modeled as a disk with a safety radius r. Time is discretized, and at each discrete step, every robot selects a control input (action) $\mathbf{u_t} \in \mathbb{R}^m$ to update its state $\mathbf{x_t} \in \mathbb{R}^n$. However, the executed control is subject to stochastic perturbations that model actuation uncertainty:

$$\nu_t \sim \mathscr{N}(\mathbf{u}_t, \Sigma), \quad \Sigma = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_m^2) \tag{1}$$

where $\nu_\mathbf{t} \in \mathbb{R}^m$ represents the actual, randomly perturbed control signal.

The robot dynamics are described by a discrete-time, continuous-state nonlinear affine system:

$$\mathbf{x_{t+1}} = F(\mathbf{x_t}) + G(\mathbf{x_t})\nu_\mathbf{t}, \tag{2}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ and $G : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ are given functions. The control input is bounded as follows:

$$\nu_{\min}[k] \le \nu_\mathbf{t}[k] \le \nu_{\max}[k], \quad k = 1, \ldots, m, \tag{3}$$

where $[k]$ denotes the k-th element of a vector.

At each time step, robot i has perfect knowledge of its own state $\mathbf{x_t^i}$, including its position $\mathbf{p_t^i}$. Furthermore, it can perceive the relative positions $\mathbf{p_t^j}$ and velocities $\mathbf{v_t^j}$ of nearby robots within its sensing range of w.

Now, consider robot i in state $\mathbf{x_t^i}$ observing another robot j. A control $\mathbf{u_t^i}$ is defined as *probabilistically safe* (or simply *safe*) with respect to robot j if, after executing the perturbed control $\nu_\mathbf{t} \sim \mathscr{N}(\mathbf{u_t}, \Sigma)$ and transitioning to the next state $\mathbf{x_{t+1}^i}$, the probability that the inter-robot distance falls below $2r$ does not exceed a predefined safety threshold δ.

The objective is to compute, at each time step, a control input $\mathbf{u_t^i}$ for every robot $i \in \mathscr{A}$ such that:

1) it satisfies the control constraints given by (3);
2) it ensures progress toward the assigned goal state $\tau_\mathbf{i}$;
3) it remains probabilistically safe with respect to all observed neighboring robots.

IV. BACKGROUND

Our method is based on Model Predictive Path Integral (MPPI) control and Reinforcement Learning (RL). This we begin by providing a brief overview of these two frameworks.

A. Model-Predictive Path Integral

The MPPI algorithm addresses discrete-time stochastic optimal control problems formulated as:

$$u^* = \underset{u\in\mathscr{U}^H}{\arg\min}\, \mathbb{E}\left[\phi(\mathbf{x_H}) + \sum_{t=0}^{H-1}\left(q(\mathbf{x_t}) + \frac{\gamma}{2}\mathbf{u_t^T}\Sigma^{-1}\mathbf{u_t}\right)\right] \tag{4}$$

where $\mathscr{U}$ denotes the set of admissible controls, $u = (\mathbf{u_0}, \ldots, \mathbf{u_{H-1}})$ represents the control sequence, $x = (\mathbf{x_0}, \ldots, \mathbf{x_H})$ is the corresponding trajectory over a prediction horizon of length H, $\phi(\cdot)$ denotes the terminal cost, $q(\cdot)$ is the running cost, and $\gamma \in \mathbb{R}^+$ is the control cost weight parameter.

Let $\mathbf{x_0}$ denote the current system state and $u^{init} = (\mathbf{u_0^{init}}, \ldots, \mathbf{u_{H-1}^{init}})$ an initial control sequence. The MPPI framework generates K stochastic perturbations $\xi^k = (\varepsilon_\mathbf{0}^\mathbf{k}, \ldots, \varepsilon_\mathbf{H-1}^\mathbf{k})$, with each noise term sampled as $\varepsilon_\mathbf{t}^\mathbf{k} \sim \mathscr{N}(0, \Sigma^*)$. Here, the sampling covariance Σ^* is a scaled version of the nominal control noise covariance Σ (matrix Σ^* must remain diagonal).

Using these samples, a set of K candidate control sequences $\{u^k\}_{k=1}^K$ is obtained as: $u^k = (\mathbf{u_0^k}, \ldots, \mathbf{u_{H-1}^k})$, $\mathbf{u_t^k} = \mathbf{u_t^{init}} + \varepsilon_\mathbf{t}^\mathbf{k}$

Each control sequence u^k induces a corresponding state trajectory x^k, which is evaluated using the cost functional $S(x,u)$:

$$\begin{aligned} S(x,u) = \phi(\mathbf{x_H}) + \sum_{t=0}^{H-1}\Big[q(\mathbf{x_t},\mathbf{u_t}) + \frac{\gamma}{2}(\,\mathbf{u_t^T}\Sigma^{-1}\mathbf{u_t} + \\ 2\mathbf{u_t^T}\Sigma^{-1}\varepsilon_\mathbf{t}\,) + \frac{\lambda}{2}\varepsilon_\mathbf{t}^\mathbf{T}(I - \mathscr{K}^{-1})\Sigma^{-1}\varepsilon_\mathbf{t}\Big] \end{aligned} \tag{5}$$

where $\lambda \in \mathbb{R}^+$ is called the inverse temperature parameter, I is the identity matrix, and $\mathscr{K}$ scales the sampling variance such that $\Sigma^* = \mathscr{K}\Sigma$.

The weight associated with each trajectory is computed as:

$$\omega(x,u) = \frac{\exp\left(-\frac{1}{\lambda}(S(x,u) - \min_l S(x^l,u^l))\right)}{\sum_{k=1}^K \exp\left(-\frac{1}{\lambda}(S(x^k,u^k) - \min_l S(x^l,u^l))\right)} \tag{6}$$

The resulting optimal control sequence is then obtained as the weighted average. After executing the first control $\mathbf{u_0^*}$, the initial control sequence is updated $u^{init} = (\mathbf{u_1^*}, \ldots, \mathbf{u_{H-1}^*}, \mathbf{u^{init}})$.

B. Updating MPPI Distributions Parameters to Ensure Collision-Free Behavior

To ensure collision-free behavior during control sampling, we adopt an approach proposed in [4], [5].

During the MPPI sampling process, the parameters of the control distribution, denoted as $\hat{\mathbf{u}}_\mathbf{t}^\mathbf{init}$ and $\hat{\Sigma}^*$, are adjusted to remain close to the nominal ones $\mathbf{u_t^{init}}, \Sigma^*$, while simultaneously increasing the likelihood of satisfying predefined safety constraints. In this work, safety constraints are represented using the ORCA-based linear inequalities in the velocity space [1].

The determination of the adjusted parameters is formulated

as the following optimization problem:

$$
\begin{aligned}
\underset{\hat{\mathbf{u}}_\mathbf{t}^{\mathbf{init}},\hat{\Sigma}^*}{\arg\min} \quad & ||\hat{\mathbf{u}}_\mathbf{t}^{\mathbf{init}} - \mathbf{u}_\mathbf{t}^{\mathbf{init}}|| + ||diag(\hat{\Sigma}^*) - diag(\Sigma^*)|| \\
\text{s.t.} \quad & \mathbf{a_j'}^T \hat{\mathbf{u}}_\mathbf{t}^{\mathbf{init}} + \Phi^{-1}(\delta_u)\sqrt{\mathbf{a_j'}\hat{\Sigma}^*\mathbf{a_j'}^\mathbf{T}} \leq b'_j \ ... \\
& - \Phi^{-1}(\delta_v)\sqrt{\mathbf{a_j'}^T \Sigma \mathbf{a_j'}}, \ \forall j \in \mathscr{A}_i \\
& \hat{\mathbf{u}}_\mathbf{t}^{\mathbf{init}}[k] + \Phi^{-1}(\delta_u)\sqrt{\hat{\Sigma}^*_{k,k}} \leq \nu_{max}[k], \\
& \hat{\mathbf{u}}_\mathbf{t}^{\mathbf{init}}[k] - \Phi^{-1}(\delta_u)\sqrt{\hat{\Sigma}^*_{k,k}} \geq \nu_{min}[k], \\
& \hat{\Sigma}^*_{k,k} \geq 0 \,, \\
& k = 1, ..., m
\end{aligned}
\tag{7}
$$

where $\mathbf{a_j'}$ and b'_j are the coefficients derived from the ORCA linear constraints, $\Phi(\cdot)$ denotes the standard normal cumulative distribution function, and $\mathscr{A}_i$ represents the set of visible neighbors of agent i.

Solving this optimization problem yields updated parameters of the sampling distribution such that, with probability at least $\delta_c = |\mathscr{A}_i| \times \delta_v \times \delta_u$, the sampled control inputs satisfy all safety constraints.

The optimization problem (7) is convex and can be formulated as a Second-Order Cone Programming (SOCP) problem (or as Linear Programming (LP) for certain types of dynamic models). A detailed derivation of the constraint formulation, as well as the transformation of the problem into SOCP and LP forms with all theoretical justifications, is provided in [5].

C. Reinforcement Learning

The problem introduced in Section III, i.e. decentralized multi-agent navigation with collision avoidance problem, can be formulated as a decentralized partially observable Markov decision process (Dec-POMDP) [34], [35]. Formally, Dec-POMDP is defined as a tuple $(\mathbb{X}, \{\mathbb{U}^i\}_{i=1}^N, \mathbb{T}, \mathfrak{R}, \{\Omega^i\}_{i=1}^N, \gamma)$, where $\mathbb{X} \ni \mathbf{x}_t$ is the global state space, $\mathbb{U}^i \ni \mathbf{u}_t^i$ is the action space of agent i, $\mathbb{U} = \times^i \{\mathbb{U}^i\}_{i=1}^N \ni \mathbf{u}_t^\times$ is the joint action space, $\mathbb{T}(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t^\times)$ is the transition model, $\mathfrak{R}(\mathbf{x}_t, \mathbf{u}_t^\times, \mathbf{x}_{t+1})$ is the reward function, $\Omega^i : \mathbb{X} \to \mathbf{o}^i$ is the observation function of agent i, $\gamma \in [0,1)$ is the discount factor.

*a) **State space**:* $\mathbb{X}$ contains the states of all agents and environment properties that matter for navigation and collisions. For example, agent positions $\mathbf{p}_t^i$, velocities $\mathbf{v}_t^i$, safe radii r^i, obstacles in the workspace, etc.

*b) **Dynamic and Constraints**:* $\mathbb{T}$ describes how the system transitions form one state to the other based on the actions (controls) picked by the agents. In the consider case $\mathbb{T}$ is defined by Eq. 2.

*c) **Reward**:* There are several possible design choices for the reward function when training RL agents. To reflect the requirements of the collision-avoidance problem, one can penalize agents for collisions and reward them for successfully reaching their destinations.

*d) **Observations**:* Each agent has partial and local information. Ω^i returns the observation $\mathbf{o}^i$ for agent i. It can include its own state $\mathbf{x}_t^i$, and relative positions $\mathbf{p}_t^j$ and velocities $\mathbf{v}_t^j$ of nearby agents $j \neq i$.

*e) **Interaction Loop**:* At each timestep t, each agent i selects an action (control) $\mathbf{u}_t^i \in \mathbb{U}^i$ using its policy $\pi^i(\mathbf{o}^i)$. The joint action $\mathbf{u}_t^\times$ drives the next state via $\mathbb{T}$. Then each agent gets a new observation, and a team receives a scalar reward.

*f) **Optimization Goal**:* The aim is to learn decentralized policies that maximize the expected discounted return. A clear statement is

$$
\times^i \pi^i(\mathbf{o}^i) = \underset{\mathbf{u}_t^\times \sim \times^i \pi^i}{\arg\max} \sum_t \left[\gamma^t \mathfrak{R}(\mathbf{x}_t, \mathbf{u}_t^\times, \mathbf{x}_{t+1})\right] \tag{8}
$$

*g) **Policy Gradient and PPO**:* There are many ways to solve the optimization problem in eq. 8. One strong family is policy gradient methods that directly improve a parameterized policy by ascending the expected return. Another method is Proximal Policy Optimization (PPO) [36], strong actor-critic approach that uses a clipped surrogate loss to make updates stable and effective.

*h) **Indepedent PPO**:* Many MARL methods build on policy gradients, and several use PPO to handle multi-agent problems, including centralized critic PPO, parameter sharing PPO, etc. A simple yet effective choice is Independent PPO (IPPO) ([37]), where each agent learns its own policy and value from local observations, and treats other agents as part of the environment.

*i) **PPO Objective**:* Let agent i use policy $\pi_{\theta^i}(\mathbf{o}_t^i)$ and value $V_{\phi^i}(\mathbf{o}_t^i)$. The clipped surrogate loss, value loss, and total loss are:

$$
\begin{cases}
\mathbf{L}_\pi^i(\theta^i) = \mathbb{E}\left[\min\left(\rho_t^i A_t^i, \text{clip}\left(\rho_t^i, 1-\varepsilon, 1+\varepsilon\right) A_t^i\right)\right], \\
\mathbf{L}_V^i(\phi^i) = \mathbb{E}\left[\left(V_{\phi^i}(\mathbf{o}_t^i) - V_{\text{target}}(\mathbf{o}_t^i)\right)^2\right], \\
\mathbf{L}_{\text{total}} = \sum_{i=1}^N \left(\mathbf{L}_{\text{clipped}}^i(\theta^i) + \lambda_1 \mathbf{L}_V^i(\phi^i) + \lambda_2 \mathbf{H}(\pi_{\theta^i})\right)
\end{cases}
\tag{9}
$$

where $\rho_t^i = \frac{\pi_{\theta^i}}{\pi_{\theta^i}^{\text{old}}}$ denotes a probability ratio, A_t^i - a GAE-λ estimator of the advantage function [38], $\mathbf{H}(\pi_{\theta^i})$ - an entropy bonus for agent i, λ_1, λ_2 are coefficients.

V. CoRL-MPPI: COOPERATIVE RL-GUIDED MPPI FOR MULTI-ROBOT COLLISION AVOIDANCE

Learning-based methods demonstrate strong capability in handling complex multi-agent scenarios, particularly in situations requiring cooperative behaviors and implicit coordination among densely arranged agents. However, such approaches typically lack formal safety guarantees for the actions they produce, and their performance is highly dependent on the training data. In contrast, the MPPI control framework does not rely on training data and can be extended to incorporate theoretical safety guarantees. Nevertheless, due to its stochastic optimization process, MPPI may struggle to efficiently resolve complex local interactions or tightly coupled agent clusters. To overcome this limitation, we propose integrating a pre-trained RL policy into the MPPI sampling process, thereby combining the cooperative decision-making capabilities of RL with the robustness and theoretical properties of MPPI. I.e. we suggest to, first, learn

a decentralized navigation and collision avoidance policy via reinforcement learning and, second, use the distribution of actions provided by the trained policy in the MPPI framework.

A. Pre-trained RL-based Policy

a) ***Observation Space****:* Each agent's observation is represented as a vector encoding information about its goal and nearby agents. Specifically, each observation includes the relative distance and angular offset to the goal, as well as to the k nearest agents within a local sensing range of size w. All components of the observation vector are normalized to ensure numerical stability and invariance to environment scale.

b) ***Action Space****:* Each agent's action consists of continuous control variables corresponding to its motion commands. These actions are bounded within predefined intervals appropriate for the agent's dynamic model used during training.

c) ***Reward****:* At every simulation step, the reward for agent i is defined as a weighted sum of components encouraging goal reaching and smooth navigation while penalizing collisions:

$$\Re^i = \Re^i_{\text{on\_g}} - \Re^i_{\text{collision}} + \Re^i_{\text{g\_dist}};$$

This structure promotes continuous progress toward each agent's target while discouraging unsafe interactions with neighbors. The formulation is compatible with various dynamic models and control parameterizations.

d) ***RL Algorithm****:* A single policy shared among agents is trained with Independent Proximal Policy Optimization (IPPO), a multi-agent extension of PPO, as described in the Background section.

B. Multi-Step Safety-Constrained and RL-Guided Planning Step

Consider the planning step for agent i. To enable the algorithm, predicted positions of neighboring agents $\mathbf{p}_t^j$ in the set $\mathscr{A}_i$ must be available for the planning horizon $t = 0, \ldots, H$. Various prediction models can be employed, in this work, a simple constant-velocity model is used:

$$\mathbf{p^j_{t+1}} = \mathbf{p^j_t} + \mathbf{v^j_t}, \; \forall j \in \mathscr{A}_i \; \forall t = 0, \ldots, H \tag{10}$$

At each control iteration, two predictive trajectories, x^{mppi} and x^{π}, are constructed. The first trajectory, x^{mppi}, follows the previously optimized control sequence u^{init} (see Section IV-A). A corresponding sequence of covariance matrices Σ^{mppi} is also maintained:

$$u^{mppi} = u^{init}, \; \Sigma^{mppi} = \{\Sigma^*_t = \Sigma^*\}_{t=0}^{H-1} \tag{11}$$

The second trajectory, x^{π}, along with the control and variance sequences u^{π} and Σ^{π}, is generated using the pre-trained RL policy π. For this purpose, the predicted positions $\mathbf{p}^j_{t-1}$ of neighboring agents, the previous trajectory element $\mathbf{x}^{\pi}_{t-1}$, and the goal position τ are combined into an observation vector $\mathbf{o}_t$. The policy then outputs the parameters of the control distribution:

$$(u^{\pi}, \Sigma^{\pi}) = \{(\mathbf{u}^{\pi}_t, \Sigma^{\pi}_t) = \pi(\mathbf{x}^{\pi} t, \mathbf{o}_t)\}_{t=0}^{H-1}. \tag{12}$$

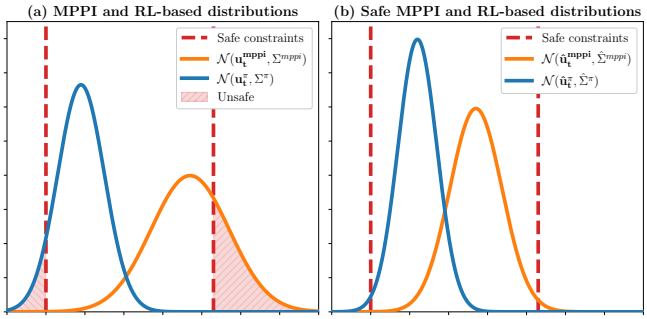

Fig. 2: Visualization of safety-constrained update of distribution parameters. The probability mass of unsafe controls (red region) is reduced to meet the required confidence level.

For each time step within the safety horizon H_{safe}, the mean controls $\hat{\mathbf{u}}^{mppi}_t$, $\hat{\mathbf{u}}^{\pi}_t$ and corresponding covariances $\hat{\Sigma}^{mppi}_t$, $\hat{\Sigma}^{\pi}_t$ for both sampling branches are refined by solving the optimization problem (7), which enforces probabilistic safety constraints to ensure collision-free behavior.

An illustration of this adjustment process is shown in Figure 2. Initially, when sampling from both distributions $\mathscr{N}(\mathbf{u}^{\pi}_t, \Sigma^{\pi}_t)$ and $\mathscr{N}(\mathbf{u}^{mppi}_t, \Sigma^{mppi}_t)$, there is a high probability of generating unsafe controls. After applying the optimization procedure, the updated distributions $\mathscr{N}(\hat{\mathbf{u}}^{\pi}_t, \hat{\Sigma}^{\pi}_t)$ and $N(\hat{\mathbf{u}}^{mppi}_t, \hat{\Sigma}^{mppi}_t)$ reduce the likelihood of unsafe samples to the specified safety level δ.

Both trajectories are propagated forward over the prediction horizon H using the robot's dynamic model (2):

$$\mathbf{x^{\pi}_t} = F(\mathbf{x^{\pi}_{t-1}}) + G(\mathbf{x^{\pi}_{t-1}})\hat{\mathbf{u}}^{\pi}_{\mathbf{t-1}} \tag{13}$$

$$\mathbf{x^{mppi}_t} = F(\mathbf{x^{mppi}_{t-1}}) + G(\mathbf{x^{mppi}_{t-1}})\hat{\mathbf{u}}^{\mathbf{mppi}}_{\mathbf{t-1}} \tag{14}$$

Subsequently, two sets of control rollouts, $\{u^k\}_{k=1}^{K^{\pi}}$ and $\{u^k\}_{k=K^{\pi}}^{K}$, are sampled: one around the RL-guided control sequence $\hat{u}^{\pi}$ with variances $\hat{\Sigma}^{\pi}_t$, and another around the MPPI-based sequence $\hat{u}^{mppi}$ with variances $\hat{\Sigma}^{mppi}_t$. Each sampled control sequence u^k induces a trajectory x^k, which is evaluated using the MPPI cost function $S(x^k, u^k)$. The resulting costs are transformed into trajectory weights $\omega(x^k, u^k)$, and the final control sequence u^* is obtained as a weighted average of all sampled controls. After executing the first control action $\mathbf{u}^*_0$, the optimized sequence is shifted forward and reused as the initialization for the next planning iteration. A detailed description of this procedure is presented in Algorithm 1.

VI. EXPERIMENTAL EVALUATION

a) ***Experimental Setup****:* In the experiments, a differential-drive robot model was employed. The agent's state, denoted as $\mathbf{x}$, was defined as $\mathbf{x} = (p_x, p_y, \theta)^T$, where p_x and p_y represent the robot's position (the center of the

Algorithm 1: CoRL-MPPI

Input: Current state $\mathbf{x_t^i}$; Goal state τ;
Control sequence from previous step u^{init};
Positions and velocities $(\mathbf{p^j}, \mathbf{v^j})$ of neighbors $j \in \mathscr{A}_i$;
MPPI parameters λ, γ, $\Sigma^*, \mathbf{u^{init}}$; Learned policy π;
Motion model $F(\cdot), G(\cdot)$; Cost function $S(\cdot,\cdot)$

$\mathbf{x_0^\pi}, \mathbf{x_0^{mppi}} \leftarrow \mathbf{x_t^i}$;
$\mathbf{p_0^j} \leftarrow \mathbf{p^j} \quad \forall j \in \mathscr{A}_i$;
for $t \in 1, ..., H$ **do**
  $\mathbf{p_t^j} \leftarrow \mathbf{p_{t-1}^j} + \mathbf{v^j} \quad \forall j \in \mathscr{A}_i$;
  Compute $\mathbf{o_{t-1}}$ using $\{\mathbf{x_{t-1}}, \tau, \{\mathbf{p_{t-1}^j}\}_{\mathbf{j} \in \mathscr{A}_i}\}$;
  $\mathbf{u_{t-1}^\pi}, \Sigma_{t-1}^\pi \leftarrow \pi(\mathbf{x_{t-1}^\pi}, \mathbf{o_{t-1}})$;
  $\mathbf{u_{t-1}^{mppi}}, \Sigma_{t-1}^{mppi} \leftarrow \mathbf{u_{t-1}^{init}}, \Sigma^*$;
  if $t \leq H_{safe}$ **then**
    Get $\hat{\mathbf{u}}_{\mathbf{t-1}}^\pi, \hat{\Sigma}_{t-1}^\pi$ by solving (7) using $\mathbf{x_{t-1}^\pi}, \{\mathbf{p_{t-1}^j}\}_{\mathbf{j} \in \mathscr{A}_i}$;
    Get $\hat{\mathbf{u}}_{\mathbf{t-1}}^{\mathbf{mppi}}, \hat{\Sigma}_{t-1}^{mppi}$ by solving (7) using $\mathbf{x_{t-1}^{mppi}}, \{\mathbf{p_{t-1}^j}\}_{\mathbf{j} \in \mathscr{A}_i}$;
  else
    $\hat{\mathbf{u}}_{\mathbf{t-1}}^\pi, \hat{\Sigma}_{t-1}^\pi \leftarrow \mathbf{u_{t-1}^\pi}, \Sigma_{t-1}^\pi$;
    $\hat{\mathbf{u}}_{\mathbf{t-1}}^{\mathbf{mppi}}, \hat{\Sigma}_{t-1}^{mppi} \leftarrow \mathbf{u_{t-1}^{mppi}}, \Sigma_{t-1}^{mppi}$
  $\mathbf{x_t^\pi} = F(\mathbf{x_{t-1}^\pi}) + G(\mathbf{x_{t-1}^\pi})\hat{\mathbf{u}}_{\mathbf{t-1}}^\pi$;
  $\mathbf{x_t^{mppi}} = F(\mathbf{x_{t-1}^{mppi}}) + G(\mathbf{x_{t-1}^{mppi}})\hat{\mathbf{u}}_{\mathbf{t-1}}^{\mathbf{mppi}}$;
Sample rollouts $\{u^k\}_{k=1}^{K_\pi}$ using $\hat{u}^\pi$ and $\{\hat{\Sigma}_t^\pi\}_{t=0}^{H-1}$;
Sample rollouts $\{u^k\}_{k=K_\pi}^{K}$ using $\hat{u}^{mppi}$ and $\{\hat{\Sigma}_t^{mppi}\}_{t=0}^{H-1}$;
for $k \in 1, ..., H$ **do**
  Predict trajectory x^k using $\mathbf{x_t^i}$, u^k and $F(\cdot), G(\cdot)$ using (2);
  Compute cost $S(x^k, u^k)$
$\omega(x,u) \leftarrow \frac{\exp\left(-\frac{1}{\lambda}(S(x,u) - \min_l S(x^l,u^l))\right)}{\sum_{k=1}^K \exp\left(-\frac{1}{\lambda}(S(x^k,u^k) - \min_l S(x^l,u^l))\right)}$;
$\mathbf{u}^* \leftarrow \sum_{k=1}^K \omega(x^k, u^k) \cdot \mathbf{u^k}$;
$u^{init} \leftarrow (\mathbf{u_1^*}, ..., \mathbf{u_{H-1}^*},)$;
return $\mathbf{u_0^*}$;

corresponding disk) in a two-dimensional workspace, and θ denotes the robot's heading angle. The control input was defined as $\mathbf{u} = (v, w)^T$, where v and w correspond to the linear and angular velocities, respectively.

At each time step, the selected control input was perturbed by zero-mean Gaussian noise $\varepsilon \sim \mathscr{N}(0, \Sigma)$. The resulting control commands were constrained according to the following bounds, consistent with (3):

$$v_{min} \leq (v + \varepsilon[0]) \leq v_{max}, \quad w_{min} \leq (w + \varepsilon[1]) \leq w_{max}. \quad (15)$$

The robot's motion was governed by the discrete-time kinematic model:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \begin{pmatrix} \cos\theta_t & 0 \\ \sin\theta_t & 0 \\ 0 & 1 \end{pmatrix} (\mathbf{u_t} + \varepsilon). \quad (16)$$

All robots in the experiments shared identical physical and control parameters: the robot size (radius of the disk) was set to $r = 0.3\,\text{m}$; linear velocity limits were $v_{min} = -1.0\,\text{m/s}$ and $v_{max} = 1.0\,\text{m/s}$; angular velocity limits were $w_{min} = -2.0\,\text{rad/s}$ and $w_{max} = 2.0\,\text{rad/s}$; and the control noise covariance was $\Sigma = \text{diag}(0.1\,\text{m/s}, 0.2\,\text{rad/s})^2$. The observation radius was assumed to be unbounded. The simulation time step was fixed at $0.1\,\text{s}$

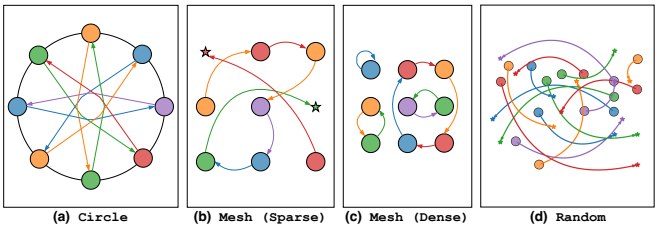

Fig. 3: Illustrative visualization of the experimental scenarios. Scales and proportions are adjusted for clarity

*b) **Experimental Environments**:* Four types of scenarios were used for training and/or evaluation: `Circle`, `Mesh` (`Sparse` and `Dense`), and `Random`. An illustrative depiction of these scenarios is presented in Figure 3, with scales and proportions adjusted for clarity.

In the `Circle` scenario, N agents were positioned equidistantly along the circumference of a circle, with their respective goals located at diametrically opposite points. Each agent's initial heading was directed toward its goal. This configuration challenges the algorithms to avoid deadlocks and manage dense interactions as agents converge toward the center.

The scenario was used both for training and evaluation. During training, circles with diameters of $14m$ and $20m$ were used, with 32 agents in each case. For evaluation, the diameter was fixed at $14m$, while the number of agents varied from 5 to 50 in increments of 5, with each instance repeated 10 times.

In the `Mesh` scenarios, agents were initially placed at the centers of cells in a uniform square grid, with goal locations assigned by a random permutation of these cells. All agents started with a zero heading. This setup was designed to assess the ability of algorithms to resolve cooperative conflicts in structured, densely populated environments.

The `Sparse` version, employed during training, used a 6×6 grid with a cell size of $2\,\text{m}$ and 32 agents, leaving some cells unoccupied. In contrast, the `Dense` version, used in evaluation, employed smaller cell sizes ($1.5\,\text{m}$) with all grid cells occupied. The number of agents was varied as 4 (2×2), 9 (3×3), 16 (4×4), and 25 (5×5). For each configuration, 10 random instances were generated and each instance was executed 10 times.

The `Random` scenario represented unstructured environments with sparse agent distributions. Here, N agents were uniformly randomly placed within a bounded $40m \times 40m$ area, and random goal positions were assigned independently. This environment type was not encountered during RL policy training and was used solely for evaluation to test generalization to unseen configurations. The number of

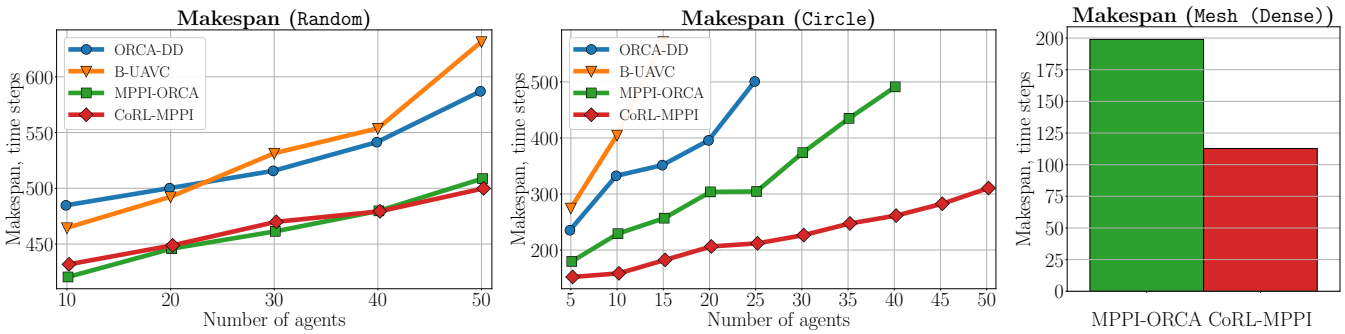

Fig. 4: The average makespan of the evaluated algorithms across the `Random`, `Circle`, and `Mesh (Dense)` scenarios. Only instances with a 100% successful runs are included. The lower is better

agents ranged from 10 to 50 in increments of 10, with 10 distinct random instances generated for each configuration. Each instance was executed 10 times.

*c) **RL Policy Training Details**:* To obtain the pre-trained policy used in our method, we trained decentralized agents in the CAMAR environment [39] using a differential-drive dynamic model. We employed the IPPO [37] algorithm implemented in the Sample Factory framework [40]. Agents shared network parameters but acted independently during training.

The observation space was modified compared to the default CAMAR setup, as described in V-A. Additionally, distances were normalized by dividing by the sensing range size, and all angular quantities were scaled to the interval $[-1;1]$.

Similarly, we used a slightly adapted reward function, as described in V-A.0.b, where the terms are defined as:

$$\begin{cases} \mathfrak{R}^i_{\text{on\_g}} & = 0.2, \text{ if } \|\mathbf{x}^i_{t+1} - \tau_i\| \leq r_\tau; \\ \mathfrak{R}^i_{\text{collision}} & = 1, \text{ if } \exists j \in N : \|\mathbf{x}^i_{t+1} - \mathbf{x}^j_{t+1}\| < r^i + r^j; \\ \mathfrak{R}^i_{\text{g\_dist}} & = 0.5 \cdot \left(\|\mathbf{x}^i_t - \tau_i\| - \|\mathbf{x}^i_{t+1} - \tau_i\|\right). \end{cases}$$

We trained a single RL policy on tasks with 32 agents for $60M$ environment steps ($\approx 1.9B$ individual agent steps). Each episode consisted of 1,500 simulation steps. The training utilized 128 parallel vectorized environments, running on a single NVIDIA H100 GPU. Owing to Sample Factory's high-throughput asynchronous architecture and the efficiency of the CAMAR simulator, the entire training process completed in under 2 hours.

*d) **Implementation Details**:* The CoRL-MPPI was implemented in C++[1]. The ONNX Runtime framework [41] was employed for executing the inference of the pre-trained policy.

The cost function $S(x,u)$ comprised several components: *(i)* a running cost penalizing the deviation of trajectory positions from the target, *(ii)* a running cost inversely proportional to the distance to the nearest predicted position of a neighboring agent, *(iii)* a collision penalty based on predicted neighboring trajectories, *(iv)* a running cost penalizing negative linear velocities, *(v)* a terminal cost penalizing the deviation of the final trajectory position from the target.

In the experiments, a time horizon of 10 steps (corresponding to 3 seconds) was used, with 1500 sampled rollouts per iteration. Among these, 30% of the trajectories were sampled from the RL-guided distribution.

*e) **Baselines**:* We compare our method with ORCA-DD [20], B-UAVC [28] and a multi-agent collision avoidance algorithm based on the MPPI framework MPPI-ORCA [5].

ORCA-DD is derived from the well-known ORCA [1] algorithm but employs an increased agent radius for control computations, accounting for kinematic constraints.

The B-UAVC method is based on the BVC approach and incorporates multiple enhancements over the base BVC algorithm, particularly the consideration of kinematic constraints for differential-drive robots.

The MPPI-ORCA algorithm was configured with a time horizon of 30 steps and 1500 sampled rollouts. The algorithm incorporated control execution noise, while the neighbors' positions were assumed to be known exactly.

ORCA-DD and B-UAVC methods have been modified in such a way that a small random value $\varepsilon_x, \varepsilon_y \sim \mathcal{N}(0, 0.3)$ is added to the goal direction vector. This was necessary to reduce the chance of getting into deadlocks in symmetric cases. In addition, for all the methods involved in the experiment, the radius of the agent used in the computations was increased by $\varepsilon_r = 0.01\ m$ relative to the real radius to minimize the chance of collision by creating additional safety buffer.

*f) **Experimental Results**:* The primary performance metrics evaluated in the experiments were the **success rate** and the **makespan**. The **success rate** represents the proportion of launches in which all agents successfully reached their respective goals without collisions, within a tolerance of $0.3,\text{m}$, and before reaching the predefined limit of 1000 simulation steps. After reaching their goals, agents were allowed to depart from them. The **makespan** denotes the total time required for all agents to reach their respective goals. In addition, the presence of collisions during execution was analyzed to assess both the safety of the compared approaches.

The resulting **success rates** are summarized in Table I,

[1]We will provide a link to a `Github` repo in case of acceptance

	Random		Circle		Mesh (Dense)	
Algorithm	**SR** ↑	**%Col.** ↓	**SR** ↑	**%Col.** ↓	**SR** ↑	**%Col.** ↓
ORCA-DD	99.8%	0%	66%	15%	46.5%	0%
B-UAVC	97.2%	0.8%	40%	57%	54.5%	27.75%
MPPI-ORCA	**100%**	**0%**	94%	6%	**100%**	**0%**
CoRL-MPPI	**100%**	**0%**	**100%**	**0%**	**99.25%**	**0.75%**

TABLE I: Success rate and percentage of runs terminated due to collisions for the evaluated algorithms. The arrows indicate preferred directions of improvement.

along with the proportion of runs terminated due to collisions. Among the tested environments, the `Random` scenario proved to be the least challenging for all algorithms. In contrast, the `Circle` and `Mesh (Dense)` scenarios, which involve complex deadlock resolution and dense agent interactions, were considerably more difficult.

The proposed method achieved flawless performance in both the `Random` and `Circle` scenarios, completing all tasks without any collisions. In the `Mesh (Dense)` scenario, it successfully solved over 99% of tasks, with only three launches resulting in collisions. These rare failures can be attributed to the probabilistic nature of the method's safety guarantees: the imposed safety threshold is close to, but not exactly, one. Its predecessor, MPPI-ORCA, demonstrated comparable performance, performing slightly worse in the `Circle` scenario while achieving a 100% success rate in `Mesh (Dense)`. In contrast, the ORCA-DD and B-UAVC algorithms exhibited substantially lower success rates and higher collision frequencies.

Figure 4 presents the average **makespan**. Each instance within a scenario is included only if 100% of runs were successfully completed. For the `Random` and `Circle` scenarios, the results are reported as a function of the number of agents, whereas for the `Mesh (Dense)` scenario, they are averaged over all trials. Due to the low success rates observed for ORCA-DD and B-UAVC in the `Mesh (Dense)` scenario, their corresponding makespan values were excluded from the analysis.

The figures clearly illustrate that MPPI-based approaches outperform the competing methods. In the `Random` scenario, CoRL-MPPI performance closely matches that of MPPI-ORCA. This similarity can be explained by the low density of agents in this environment, where cooperative behavior is less critical for avoiding collisions. It is also worth noting that large-scale sparse configurations similar to the `Random` scenario were not included in the RL policy's training process. Nevertheless, the proposed approach maintained comparable performance to the classical method, demonstrating strong generalization capability. In contrast, in densely populated scenarios requiring coordinated decision-making, the proposed method significantly outperforms all competitors, achieving nearly a twofold improvement over MPPI-ORCA.

The experimental evaluation demonstrates that CoRL-MPPI consistently outperforms the baseline algorithms across all tested scenarios. It achieves the highest success rates and the lowest makespan values, indicating both superior reliability and efficiency. In simpler environments such as `Random`, its performance remains comparable to that of MPPI-ORCA, confirming the method's robustness and generalization capability. However, in more complex and densely populated scenarios like `Circle` and `Mesh (Dense)`, the proposed approach exhibits a substantial advantage, successfully resolving challenging interactions and reducing task completion time.

VII. CONCLUSIONS

In this work we have introduced CoRL-MPPI – a novel hybrid framework that enhances the Model Predictive Path Integral controller with learnable, cooperative behaviors for decentralized multi-robot collision avoidance. Our approach successfully addresses a key limitation of vanilla MPPI – its reliance on uninformed random sampling – by leveraging a pre-trained reinforcement learning policy to intelligently bias the sampling distribution. Extensive simulation experiments in dense and dynamic scenarios confirm that the suggested method significantly outperforms state-of-the-art baselines, including ORCA, BVC, and a decentralized multi-agent MPPI implementation. Promising directions include deploying suggested method on physical robot swarms to bridge the sim-to-real gap and investigating the online adaptation of the policy to further improve generalization across diverse and evolving environments.

REFERENCES

[1] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*, 2011, pp. 3–19.

[2] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, online collision avoidance for dynamic vehicles using buffered voronoi cells," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1047–1054, 2017.

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

[4] S. Dergachev and K. Yakovlev, "Model predictive path integral for decentralized multi-agent collision avoidance," *PeerJ Computer Science*, vol. 10, p. e2220, 2024.

[5] ——, "Decentralized uncertainty-aware multi-agent collision avoidance with model predictive path integral," 2025. [Online]. Available: https://arxiv.org/abs/2507.20293

[6] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1714–1721.

[7] T. Kim, G. Park, K. Kwak, J. Bae, and W. Lee, "Smooth model predictive path integral control without smoothing," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 406–10 413, 2022.

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

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

[10] M. S. Gandhi, B. Vlahov, J. Gibson, G. Williams, and E. A. Theodorou, "Robust model predictive path integral control: Analysis and performance guarantees," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1423–1430, 2021.

[11] C. Tao, H. Kim, H. Yoon, N. Hovakimyan, and P. Voulgaris, "Control barrier function augmentation in sampling-based control algorithm for sample efficiency," in *2022 American Control Conference (ACC)*, 2022, pp. 3488–3493.

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

[13] Z. Wang, A. D. Saravanos, H. Almubarak, O. So, and E. A. Theodorou, "Sampling-based optimization for multi-agent model predictive control," *arXiv preprint arXiv:2211.11878*, 2022.

[14] L. Song, P. Zhao, N. Wan, and N. Hovakimyan, "Safety embedded stochastic optimal control of networked multi-agent systems via barrier states," in *2023 American Control Conference (ACC)*, 2023, pp. 2554–2559.

[15] I. S. Mohamed, M. Ali, and L. Liu, "Chance-constrained sampling-based mpc for collision avoidance in uncertain dynamic environments," *IEEE Robotics and Automation Letters*, vol. 10, no. 7, pp. 7492–7499, 2025.

[16] L. Streichenberg, E. Trevisan, J. J. Chung, R. Siegwart, and J. Alonso-Mora, "Multi-agent path integral control for interaction-aware motion planning in urban canals," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 1379–1385.

[17] N. Hansen, X. Wang, and H. Su, "Temporal difference learning for model predictive control," *arXiv preprint arXiv:2203.04955*, 2022.

[18] N. Hansen, H. Su, and X. Wang, "Td-mpc2: Scalable, robust world models for continuous control," *arXiv preprint arXiv:2310.16828*, 2023.

[19] 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," vol. 9, no. 2, pp. 3605–3616.

[20] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha, "Smooth and collision-free navigation for multiple robots under differential-drive constraints," in *2010 IEEE/RSJ international conference on intelligent robots and systems*, 2010, pp. 4584–4589.

[21] J. Snape, S. J. Guy, J. Van Den Berg, and D. Manocha, "Smooth coordination and navigation for multiple differential-drive robots," in *Experimental Robotics: The 12th International Symposium on Experimental Robotics*, 2014, pp. 601–613.

[22] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed autonomous robotic systems: The 10th international symposium*, 2013, pp. 203–216.

[23] J. Alonso-Mora, P. Beardsley, and R. Siegwart, "Cooperative collision avoidance for nonholonomic robots," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 404–420, 2018.

[24] B. Gopalakrishnan, A. K. Singh, M. Kaushik, K. M. Krishna, and D. Manocha, "Prvo: Probabilistic reciprocal velocity obstacle for multi robot navigation under uncertainty," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1089–1096.

[25] D. Hennes, D. Claes, W. Meeussen, and K. Tuyls, "Multi-robot collision avoidance with localization uncertainty," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 2012, pp. 147–154.

[26] D. Claes, D. Hennes, K. Tuyls, and W. Meeussen, "Collision avoidance under bounded localization uncertainty," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 1192–1198.

[27] M. Wang and M. Schwager, "Distributed collision avoidance of multiple robots with probabilistic buffered voronoi cells," in *2019 international symposium on multi-robot and multi-agent systems (MRS)*, 2019, pp. 169–175.

[28] H. Zhu, B. Brito, and J. Alonso-Mora, "Decentralized probabilistic multi-robot collision avoidance using buffered uncertainty-aware voronoi cells," *Autonomous Robots*, vol. 46, no. 2, pp. 401–420, 2022.

[29] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *Proceedings of the 2017 IEEE International Conference on Robotics and Automation ({ICRA} 2017)*, pp. 285–292.

[30] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *2018 IEEE international conference on robotics and automation (ICRA)*, 2018, pp. 6252–6259.

[31] T. Fan, P. Long, W. Liu, and J. Pan, "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 856–892, 2020.

[32] S. Asayesh, M. Chen, M. Mehrandezh, and K. Gupta, "Least-restrictive multi-agent collision avoidance via deep meta reinforcement learning and optimal control," in *International Conference on Robot Intelligence Technology and Applications*, 2022, pp. 213–225.

[33] R. Han, S. Chen, S. Wang, Z. Zhang, R. Gao, Q. Hao, and J. Pan, "Reinforcement learned distributed multi-robot navigation with reciprocal velocity obstacle shaped rewards," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 5896–5903, 2022.

[34] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.

[35] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[37] C. S. De Witt, T. Gupta, D. Makoviichuk, V. Makoviychuk, P. H. Torr, M. Sun, and S. Whiteson, "Is independent learning all you need in the starcraft multi-agent challenge?" *arXiv preprint arXiv:2011.09533*, 2020.

[38] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[39] A. Pshenitsyn, A. Panov, and A. Skrynnik, "Camar: Continuous actions multi-agent routing," *arXiv preprint arXiv:2508.12845*, 2025.

[40] A. Petrenko, Z. Huang, T. Kumar, G. Sukhatme, and V. Koltun, "Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7652–7662.

[41] O. R. developers, "Onnx runtime," https://onnxruntime.ai/, 2021, version: x.y.z.