2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Bird's-Eye-View Trajectory Planning of Multiple Robots using Continuous Deep Reinforcement Learning and Model Predictive Control

Kristian Ceder[1*], Ze Zhang[1*], Adam Burman, Ilya Kuangaliyev,
Krister Mattsson, Gabriel Nyman, Arvid Petersén, Lukas Wisell and Knut Åkesson[1]

Abstract— **Efficient motion planning and control for multiple mobile robots in industrial automation and indoor logistics face challenges such as trajectory generation and collision avoidance in complex environments. We propose a hybrid, sequential method combining Bird's-Eye-View vision-based continuous Deep Reinforcement Learning (DRL) with Model Predictive Control (MPC). DRL generates candidate trajectories in complex environments, while MPC refines these trajectories to ensure adherence to kinematic and dynamic constraints of the robot, as well as constraints modeling humans' current and predicted future positions. In this study, the DRL utilizes a Deep Deterministic Policy Gradient model for trajectory generation, demonstrating its capability to navigate non-convex obstacles, a task that might pose challenges for MPC. We demonstrate that the proposed hybrid DRL-MPC model performs favorably in handling new scenarios, computational efficiency, time to destination, and adaptability to complex multi-robot situations when compared to pure DRL or pure MPC approaches.**

I. INTRODUCTION

In industrial indoor logistics, Autonomous Mobile Robots (AMRs) can operate alongside humans and other manually operated vehicles. Though tasks and routes are typically pre-scheduled, AMRs may encounter unforeseen situations during operation and have to quickly adapt to them. In this work, we aim to enable AMRs to navigate their paths precisely and efficiently while adapting to avoid static and dynamic obstacles. We assume a permanent ceiling-mounted vision system [1] is given to provide real-time updates of the current state of the workspace, which can be further processed as Bird's-Eye-View (BEV) occupancy grid images.

Model Predictive Control (MPC) is a recognized strategy for planning, tracking, and controlling the movement of mobile robots with references spanning from single robot applications [2], [3] to fleets of robots [4], [5], [6]. In MPC, trajectory planning, tracking, and control are treated as an optimization problem employing a receding horizon approach, with paths and obstacles serving as various constraints. However, there are still challenges for stable real-time implementation of MPC, such as significant computational requirements, variability in time for solution convergence, and the difficulty of identifying feasible paths around non-convex obstacles.

As a combination of deep learning and reinforcement learning, Deep Reinforcement Learning (DRL) [7] presents a trajectory-planning alternative with constant, predictable computational costs, processing images directly to generate trajectories in complex, dynamic environments. Despite potential robustness issues [8], [9] and the need for extensive offline training, DRL, particularly the Deep Deterministic Policy Gradient (DDPG) algorithm [10], offers advantages in handling continuous action spaces for control tasks, showing promise across various applications in trajectory planning, robotics, and control [11], [12], [13].

In our previous work [14], a hybrid DRL-MPC method is introduced to address the limitations of standalone techniques, thereby enhancing performance beyond that of pure DRL or MPC solutions. This paper advances that approach by integrating it with a DDPG model featuring a continuous action space, aiming to improve the hybrid strategy's reliability and robustness. We focus on planning improved reference trajectories by transitioning to a continuous action space and refining the training process for more effective use of the BEV vision input, which the previous hybrid approach struggled with. Additionally, we extend its application to multi-agent scenarios through the distributed execution of MPC controllers, achieving promising results without specifically training the DDPG model for multi-robot trajectory planning. The main contributions of the study are threefold:

- Designing a BEV vision-based DDPG model for trajectory planning and collision avoidance in dynamic environments.
- Enhancing the hybrid approach by combining DDPG and MPC for both single and multi-robot scenarios.
- Evaluating the improved method across various scenarios involving both single and multiple agents.

II. PRELIMINARIES

A. Model Predictive Control

To solve a constrained Optimal Control Problem (OCP), Model Predictive Control (MPC) [15] estimates the future behavior of the control target within a predictive horizon and makes informed decisions regarding an objective, while complying with given constraints. In particular, given a discrete motion model $\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k)$, where $\boldsymbol{x}_k$ and $\boldsymbol{u}_k$ are the state and control input at time step k, assuming $\boldsymbol{z}_k = [x_k, y_k]^\top$ is part of $\boldsymbol{x}_k$ indicating the planar position of the robot, $\tilde{\boldsymbol{x}}_k$ and $\tilde{\boldsymbol{u}}_k$ are the references for $\boldsymbol{x}$ and $\boldsymbol{u}$, $\bar{\boldsymbol{x}}$ is the initial state, U_k is the constraint on the control input, a general trajectory tracking OCP with horizon N can be

*Denotes equal contribution.

[1]Chalmers University of Technology, 41296 Gothenburg, Sweden {cederk, zhze, knut}@chalmers.se

This work is supported by the Vinnova project AIHURO (Intelligent human-robot collaboration) and the ITEA project ArtWork.

formed as,

$$\min_{\boldsymbol{u}_{0:N-1}} J_N + \sum_{k=0}^{N-1} \left(||\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_k||^2_{\boldsymbol{Q}_x} + ||\boldsymbol{u}_k - \tilde{\boldsymbol{u}}_k||^2_{\boldsymbol{Q}_u} \right), \quad (1)$$

$$\text{s.t.} \quad \boldsymbol{x}_0 = \bar{\boldsymbol{x}}, \quad (2)$$

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k),\ k \in \mathbb{N}_{[0,N-1]}, \quad (3)$$

$$\boldsymbol{u}_k \in U_k,\ k \in \mathbb{N}_{[0,N-1]}, \quad (4)$$

$$\boldsymbol{z}_k \notin \mathcal{O} \cup \mathcal{D}_k,\ k \in \mathbb{N}_{[0,N]}, \quad (5)$$

where J_N is the terminal cost, Q_x and Q_u are the penalty weights, and $\mathcal{O}$ and $\mathcal{D}_k$ are the static and dynamic obstacle areas respectively. For multi-robot cases, other robots can be perceived as dynamic obstacles for the ego one. The notation $\mathbb{N}_{[a,b]}$ means the integer set from a to b.

For real-time tasks, only the first action $\boldsymbol{u}_0$ is applied to the robot. The OCP is reconsidered with a new set of parameters at each time step, which is challenging and high-demanding in runtime if the environment is complex.

B. Deep Reinforcement Learning

Markov Decision Processes (MDPs), as the foundation of Reinforcement Learning (RL) [16], are discrete models for sequential decision-making. An MDP is defined by a set of states S, actions A, the transition space P describing the transitions among states, the reward R of taking an action at a state, and a discount factor γ balancing the immediate versus distant rewards. A policy π is a strategy of action selection given states and the goal in RL is to find the optimal policy π^* maximizing the cumulative discounted reward, or *Return*, $G_k = \sum_{i=0}^{\infty} \gamma^k R_{k+i+1}$ at time k. The agent learns through exploring the state space in a trial-and-error fashion. Q-learning [17] is a well-known RL algorithm, where an action-value function is approximated

$$q_\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_\pi \left[G_k \mid S_k = \boldsymbol{s}, A_k = \boldsymbol{a} \right], \quad (6)$$

and the values $Q(\boldsymbol{s}, \boldsymbol{a})$ are stored in a look-up Q-table. The agent explores the environment by following a $\epsilon-$greedy policy per step, i.e., taking the action with the largest G_k with probability $(1-\epsilon)$ and taking a random action with probability ϵ. Meanwhile, the Q-values are updated:

$$Q(\boldsymbol{s},\boldsymbol{a}) \leftarrow Q(\boldsymbol{s},\boldsymbol{a}) + \eta \left[R(\boldsymbol{s},\boldsymbol{a}) + \gamma \max_{\boldsymbol{a}' \in A} Q(\boldsymbol{s}',\boldsymbol{a}') - Q(\boldsymbol{s},\boldsymbol{a}) \right], \quad (7)$$

where $\boldsymbol{s}'$ is the state at the next time step, $R(\boldsymbol{s}, \boldsymbol{a})$ is the reward at $\boldsymbol{s}'$ from taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$, and η is the learning rate. After convergence of Q, the optimal policy can be found by

$$\pi^*(\boldsymbol{s}_k) = \underset{\boldsymbol{a}}{\operatorname{argmax}}\ Q_\pi(\boldsymbol{s}_k, \boldsymbol{a}). \quad (8)$$

Q-learning's reliance on Q-tables limits it to discrete state and action spaces. DRL combines RL with neural networks for function approximation, exemplified by DQN [18], which extends Q-learning to continuous state spaces.

In continuous action spaces, Q-learning needs to solve (8) via iterative optimization or discretize the action domain, which is unsuitable in most control applications [10]. Alternatively, *policy gradient methods* [16] aim to learn a policy $\pi(\boldsymbol{s}|\boldsymbol{\theta}^\pi)$ parameterized by $\boldsymbol{\theta}^\pi$, which is updated by taking a gradient ascent step with step length α:

$$\boldsymbol{\theta}^\pi_{k+1} = \boldsymbol{\theta}^\pi_k + \alpha \widehat{\nabla_{\boldsymbol{\theta}^\pi_k} L}, \quad (9)$$

where $\widehat{\nabla_{\boldsymbol{\theta}^\pi_k} L}$ is an approximation of the gradient of the performance measure L with respect to $\boldsymbol{\theta}^\pi_k$. Within policy gradient methods, the *actor-critic* approach [16] learns both a policy (actor) and a value function (critic). Deep Deterministic Policy Gradient (DDPG) uses a traditional actor-critic structure with neural networks for both the actor and the critic with parameters $\boldsymbol{\theta}^\pi$ and $\boldsymbol{\theta}^Q$. The actor is updated by

$$\nabla_{\boldsymbol{\theta}^\pi} L = \mathbb{E}\left[\nabla_a Q(\boldsymbol{s}_k, \boldsymbol{a}|\boldsymbol{\theta}^Q)|_{\boldsymbol{a}=\pi(\boldsymbol{s}_k)} \nabla_{\boldsymbol{\theta}^\pi} \pi(\boldsymbol{s}_k|\boldsymbol{\theta}^\pi) \right] \quad (10)$$

and the critic is updated by minimizing the loss

$$\mathcal{L}(\boldsymbol{\theta}^Q) = \mathbb{E}\left[\left(Q(\boldsymbol{s}_k, \boldsymbol{a}_k|\boldsymbol{\theta}^Q) - \bar{G}_k \right)^2 \right], \quad (11)$$

$$\bar{G}_k = R(\boldsymbol{s}_k, \boldsymbol{a}_k) + \gamma Q(\boldsymbol{s}_{k+1}, \pi(\boldsymbol{s}_{k+1}|\boldsymbol{\theta}^\pi)|\boldsymbol{\theta}^Q). \quad (12)$$

DDPG uses target networks [18] for both the actor and critic to increase convergence properties. The parameters $\boldsymbol{\theta}^{'}$ of target networks are updated using a soft update $\boldsymbol{\theta}^{'} \leftarrow (1-\tau)\boldsymbol{\theta}^{'} + \tau\boldsymbol{\theta}$ with $\tau \ll 1$.

III. Obstacle Avoidance with Optimal Control

In this section, we define the problem of collision-free navigation for a fleet of AMRs and present the optimal control formulation for this problem.

A. Problem Formulation

Consider the navigation problem of a fleet of AMRs with configurations $\mathcal{R}_k = \{R^{(1)}_k, R^{(2)}_k, \ldots, R^{(n_r)}_k\}$, where $R^{(i)}_k$ is the configuration of the i-th AMR at time step k adhering to the discrete-time motion model $\boldsymbol{x}^{(i)}_{k+1} = f(\boldsymbol{x}^{(i)}_k, \boldsymbol{u}^{(i)}_k)$, with $\boldsymbol{x}^{(i)}$ being the state vector, and $\boldsymbol{u}^{(i)}$ being the control input. For any AMR in the fleet, the goal while not idle is to follow a predefined path $\mathcal{P}^{\text{ref}} = \{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_M\}$ composed of M waypoints, where $\boldsymbol{p}_j \in \mathbb{R}^2$ represents the j-th waypoint, with $\boldsymbol{p}_0$ as the start and $\boldsymbol{p}_M$ as the target positions. The workspace of AMRs contains n_o static obstacles $\mathcal{O} = \{O^{(1)}, O^{(2)}, \ldots, O^{(n_o)}\}$ and n_d dynamic obstacles $\mathcal{D}_k = \{D^{(1)}_k, D^{(2)}_k, \ldots, D^{(n_d)}_k\}$ at time step k. Each AMR should navigate from the start to the target position at a predefined reference speed v_{ref}, whilst ensuring collision-free movement. Collision-free navigation for any robot i requires that its position $\boldsymbol{z}^{(i)}_k \in \mathbb{R}^2$ does not intersect with the spaces of obstacles or other AMRs at any time step k, i.e., $\boldsymbol{z}^{(i)}_k \notin \left(\mathcal{O} \cup \mathcal{D}_k \cup (\mathcal{R}_k - \{R^{(i)}_k\}) \right), \forall k$.

B. Modelling of Obstacles

To be added to the constraints of the MPC problem, obstacles are modeled mathematically as described in [19]. A static obstacle $O^{(i)}$ is modeled as a convex polygon defined as an intersection of half-spaces. Thus, the static obstacle area $\mathcal{O} = \cup_n O^{(n)}$ is the union of all static obstacles.

For dynamic obstacles, such as pedestrians, ellipses serve as their representation to capture their motion patterns as in [1]. To judge whether a point $\boldsymbol{p} = [p_x, p_y]^\top$ resides within an ellipse with the center $\boldsymbol{\mu} = [\mu_x, \mu_y]^\top$ and axes $\boldsymbol{\sigma} = [\sigma_x, \sigma_y]^\top$, an indicator $\iota \geq 0$ is employed as in (13). The notation $[x]_+$ is the compact form of $\max(0, x)$.

$$\iota(\boldsymbol{p} \mid \boldsymbol{\mu}, \boldsymbol{\sigma}) = \left[1 - (\frac{p_x - \mu_x}{\sigma_x})^2 - (\frac{p_y - \mu_y}{\sigma_y})^2\right]_+ \tag{13}$$

Based on the indicator, the dynamic obstacle area composed of N_d obstacles can be defined as

$$\mathcal{D} = \{\boldsymbol{p} \in \mathbb{R}^2 \mid \exists\, i \in \mathbb{N}_{[1,N_d]},\, \iota(\boldsymbol{p}|\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i) > 0\}. \tag{14}$$

Since robots are simplified as points during the computation of control actions, all obstacles are padded with the size of the robot plus an extra margin for safety.

C. Optimal Control Formulation

The complete MPC formulation is similar to [14] with an extension of fleet collision avoidance. To safely interact with other robots, the ego robot can regard them as dynamic obstacles. However, this results in the allocation of computational resources potentially taking precedence over other uncontrolled dynamic obstacles. In industrial environments, mobile robots in a fleet have known size, whose actions and predicted states are determined by MPC. Therefore, fleet collision avoidance is implemented by adding a cost term in the objective function preventing any two robots from being closer than a safe distance. The total formulation of the MPC problem with the sampling time Δt is

$$\begin{aligned}
\min_{\boldsymbol{u}_{0:N-1}} \quad & ||\boldsymbol{x}_N - \tilde{\boldsymbol{x}}_N||^2_{\boldsymbol{Q}_N} \\
& + \sum_{k=0}^{N-1} \left[J_R(k) + \sum_{j=1}^{n_r} J_F(\boldsymbol{z}_k, \hat{\boldsymbol{z}}_k^{(j)}) \right] && (15) \\
\text{s.t.} \quad & \boldsymbol{x}_0 = \bar{\boldsymbol{x}}, && (16) \\
& \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k),\, k \in \mathbb{N}_{[0,N-1]}, && (17) \\
& \boldsymbol{u}_k \in [\boldsymbol{u}_{min}, \boldsymbol{u}_{max}],\, k \in \mathbb{N}_{[0,N-1]}, && (18) \\
& \frac{\boldsymbol{u}_k - \boldsymbol{u}_{k-1}}{\Delta t} \in [\dot{\boldsymbol{u}}_{min}, \dot{\boldsymbol{u}}_{max}],\, k \in \mathbb{N}_{[0,N-1]}, && (19) \\
& \boldsymbol{z}_k \notin \mathcal{O} \cup \mathcal{D}_k,\, k \in \mathbb{N}_{[0,N]}, && (20)
\end{aligned}$$

where all $\boldsymbol{Q}$s are penalty parameters; $\boldsymbol{u}_{-1}$ is the action from the last step if it exists, otherwise it is a zero vector. Velocity and acceleration are limited by box constraints as in (18) and (19). In the objective function, $J_R(k) = ||\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_k||^2_{\boldsymbol{Q}_x} + ||\boldsymbol{u}_k - \tilde{\boldsymbol{u}}_k||^2_{\boldsymbol{Q}_u} + ||\boldsymbol{u}_k - \boldsymbol{u}_{k-1}||^2_{\boldsymbol{Q}_a}$ is the reference deviation cost for the robot, and $J_F(\boldsymbol{z}_k, \hat{\boldsymbol{z}}_k^{(j)})$ as defined in (21) is the fleet collision cost between the ego robot and robot j.

$$J_F(\boldsymbol{z}_k^{(i)}, \hat{\boldsymbol{z}}_k^{(j)}) = \left[Q_f \cdot \left(d^{\text{fleet}} - ||\boldsymbol{z}_k^{(i)} - \hat{\boldsymbol{z}}_k^{(j)}||_2\right)\right]_+. \tag{21}$$

Note that in (21), d^{fleet} is the safe distance between any two robots and the penalty weight Q_f is zero if $i = j$.

IV. Reference trajectory from Deep Reinforcement Learning

In this section, the components for the reference trajectory generation of DDPG are presented. The approach is similar to [14] with some modifications to accommodate moving from a discrete to a continuous action space.

A. State Observation

The state observation $\boldsymbol{s}_k$ at time step k is divided into the *internal* observations $\boldsymbol{s}_k^{\text{int}}$, and *external* observation $\boldsymbol{s}_k^{\text{ext}}$. The internal observation can be seen in (22),

$$\boldsymbol{s}_k^{\text{int}} = [v_{k-1},\, \omega_{k-1},\, \boldsymbol{s}_k^{\text{close}},\, \boldsymbol{s}_k^{\text{path}}] \in \mathbb{R}^{14}. \tag{22}$$

Specifically, v_{k-1} and ω_{k-1} are speed and angular velocity at $k-1$, and they are normalized in the range $[-1, 1]$. The next component $\boldsymbol{s}_k^{\text{close}} = [\cos(\beta^{\text{close}}),\, \sin(\beta^{\text{close}}),\, \bar{d}^{\text{close}}]$ where β^{close} is the relative angle and $\bar{d}^{\text{close}}$ is the normalized distance to the closest point on the reference path relative to the agent. Finally, $\boldsymbol{s}_k^{\text{path}} \in \mathbb{R}^9$ consists of three closest future waypoints on the reference path defined similarly as in $\boldsymbol{s}_k^{\text{close}}$.

The external observation $\boldsymbol{s}^{\text{ext}} \in \mathbb{R}^{3\times 54\times 54}$ is modeled as an image with three channels, where two channels consist of occupancy grids and one channel represents a distance field to locate the agent in the image. The premise is that the external ceiling-mounted camera system can produce occupancy grids through semantic segmentation and object tracking. One occupancy grid of the current time step k and one grid from $\Delta k = 5$ time steps ago to provide temporal information of dynamic obstacles. The occupancy grids are cropped images from the global camera system, rotated, and translated such that the agent's heading is consistent.

B. Action space

The actions generated by the DDPG model are continuous acceleration and angular acceleration $a_k, \dot{\omega}_k \in [-1, 1]$. To comply with the AMR kinematics defined in [4], $\dot{\omega}_k$ is rescaled to be in the range of $[-3, 3]$.

C. Reward Function

One of the hurdles of DRL is the definition of reward functions, in terms of design and tuning parameters to promote a specific desired behavior for the problem at hand [20]. In this work, the reward function is similar to [14]:

$$R = R^{\text{goal}} + R^{\text{collision}} + R^{\text{path}} + R^{\text{speed}} + R^{\text{dev}}, \tag{23}$$

where

$$R_k^{\text{goal}} = \begin{cases} \lambda^{\text{goal}}, & \text{goal reached at time-step k} \\ 0, & \text{otherwise} \end{cases} \tag{24a}$$

$$R_k^{\text{col}} = \begin{cases} -\lambda^{\text{col}}, & \text{collision at time-step k} \\ 0, & \text{otherwise} \end{cases} \tag{24b}$$

$$R_k^{\text{path}} = \lambda^{\text{path}}(l_k - l_{k-1}), \tag{24c}$$

$$R_k^{\text{speed}} = -\lambda^{\text{speed}}\text{max}(0, (v_k - v_k^{\text{ref}}))\Delta t, \tag{24d}$$

$$R_k^{\text{dev}} = -\lambda^{\text{dev}}(d_k^{\text{close}})^2\Delta t. \tag{24e}$$

All terms in (23) are weighted with positive scalars λ, and l_k is the traversed distance of the AMR projected perpendicularly on the reference path at k.

D. Reference Trajectory Generation

A new local reference trajectory $\mathcal{T}'$ is generated by the DDPG agent, to cope with scenarios when an object is blocking the MPCs original reference path $\mathcal{T}^{\text{ref}}$. The actions from the agent's learned policy, a_k and $\dot{\omega}$, is integrated using the motion model:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ v_{k+1} \\ \varphi_{k+1} \\ \omega_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + v_k \cos \varphi_k \Delta t \\ y_k + v_k \sin \varphi_k \Delta t \\ v_k + a_k \Delta t \\ \varphi_k + \omega_k \Delta t \\ \omega_k + \dot{\omega}_k \Delta t \end{bmatrix}. \tag{25}$$

Trajectory $\mathcal{T}'$ is generated by simulating the DDPG model for N steps and integrating the actions according to (25). To increase stability of $\mathcal{T}'$ only the first control input $\boldsymbol{u}_0^{\text{DDPG}}$ is used in the first time step. The consecutive control inputs use an additional velocity reference v'_{ref} and a decay constant λ^{decay} such that $\boldsymbol{u}_k^{\text{DDPG}} = [v'_{\text{ref}}, (\lambda^{\text{decay}})^k \omega^{\text{DDPG}}]^T$, $\forall k > 0$.

When and where to use $\mathcal{T}^{\text{ref}}$ or $\mathcal{T}'$ is determined by a switch, provided in the vision system [14], with parameter adjustments to handle multiple robots. To concisely introduce the switch, $\mathcal{T}^{\text{ref}}$ is used when no object is blocking $\mathcal{P}^{\text{ref}}$ (as defined in III-A) in the vicinity of the AMR according to the lookahead parameter Δd, which defines at which distance from the object blocking $\mathcal{P}^{\text{ref}}$ the switch from $\mathcal{T}^{\text{ref}}$ and $\mathcal{T}'$ should occur. The switch is activated as long as there are objects on the section of $\mathcal{P}^{\text{ref}}$ between the closest point on $\mathcal{P}^{\text{ref}}$ relative to the AMR and Δd along the path. For the multi-robot case, Δd has to be adjusted to a longer horizon to be able to handle the added complexity due to the simultaneous navigation of multiple AMRs.

V. Implementation

The system is evaluated using an Intel i7-13700H with NVIDIA RTX 3060 and is documented online[1]. Built on [14], we use the OpEn Engine [21] for MPC and Stable Baselines3 [22] for DRL, enhancing DDPG with a Prioritized Experience Replay Buffer [23]. The architecture includes dual feature extractors for actor and critic networks to process internal ($\boldsymbol{s}^{\text{int}}$) and external ($\boldsymbol{s}^{\text{ext}}$) states, employing the convolutional neural network [18] for images. Images are converted into a 256-length vector, merged with $\boldsymbol{s}^{\text{int}}$ for the policy network, which diverges from [10].

The training environment is similar to [14] with data split between random and twelve manually designed fixed scenes for evaluation. Fixed scenes incorporate randomization in start positions, obstacle sizes, and speed of dynamic obstacles, while the random scenes generate varying start and goal points, obstacles, and L- or U-shaped non-convex barriers. The initial reference paths are determined via visibility graphs and the A^* algorithm.

The model is trained using Ornstein-Uhlenbeck action noise [10], soft update factor $\tau = 0.01$, discount factor $\gamma = 0.98$, and learning rate $\alpha = 10^{-4}$. A delayed update of the target network is used as in [18]. The target network is updated every fourth episode and the actor and critic networks are trained using as many gradient steps as in the environment over an episode. To evaluate and checkpoint the model during training, the model is deterministically evaluated for 32 episodes at a fixed interval during training to save the best model.

[1]https://github.com/kristianceder/DRL-Traj-Planner

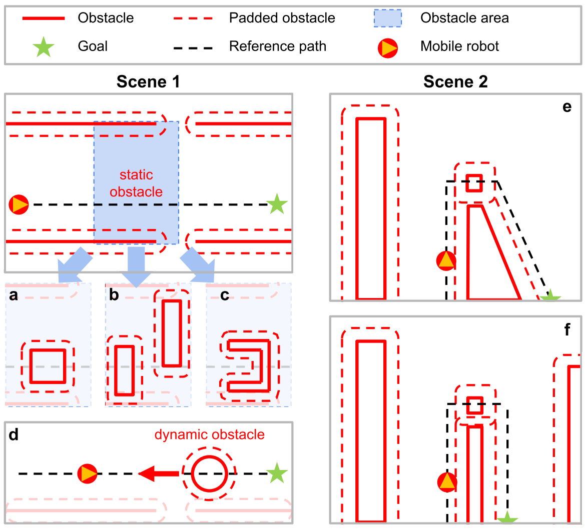

Fig. 1. Scenes for single-agent evaluation [14]. Scene 1 includes three types of static obstacles and one dynamic obstacle scenario. Scene 2 assesses the turning performance of AMRs with unexpected obstacles.

Training multiple agents operating in a shared environment is complex and resource-intensive. Centralizing training necessitates a shared experience buffer [24], complicating the model and hindering scalability. Furthermore, a fluctuating number of agents during runtime complicates setting up the training environment. To address these challenges, our training is distributed, mirroring single-agent training setups; agents are trained independently and other agents are treated as dynamic obstacles concerning each other during operation. This simplification reduces complexity at the expense of potential optimality in decisions. Despite this, evaluations show emergent cooperative behaviors among agents, notably with MPC support, demonstrating the approach's effectiveness despite its simplifications.

VI. Evaluation

As this work aims to extend into a multi-robot setting, the experiment is divided into single- and multi-robot cases. Scenes 1 and 2 are for single-agent evaluation, while Scenes 3-5 are for multi-agent evaluation. Links to videos of evaluations are available in the code repository.

A. Single-agent

To compare the DDPG model with the previous work and DQN implementation, it is evaluated on the same scenes as in [14], illustrated in Fig. 1.

1) Use Cases: The first scene examines the agent's ability to avoid unexpected obstacles along a straight path, featuring static objects of varying size and shape (a, b, and c) and a dynamic obstacle (d) in a collision course. The second scene

TABLE I

EVALUATION RESULTS FOR A SINGLE AGENT (AVERAGE OVER 50 RUNS). RESULTS WITH SUCCESS RATES UNDER 30% HAVE BEEN OMITTED FOR BREVITY. NOTABLY, THE PROPOSED METHOD, **HYB-DDPG** , ACHIEVES A 100% SUCCESS RATE ACROSS ALL SCENARIOS.

Scene	Obstacle type	Method	Computation time (ms/step)			Deviation (m)		Action smoothness		Other	
			mean	max	std	mean	max	speed	angular speed	finish time step	success rate (%)
Scene 1	(a) Rectangular obstacle	MPC	29.03	94.11	24.51	0.54	1.94	0.04	0.03	76	100
		DDPG	0.77	10.41	7.78	1.02	1.98	0.08	0.61	73	100
		HYB-DQN-V	23.07	240.46	45.22	0.91	2.11	0.03	0.1	81	100
		HYB-DDPG	11.94	49.74	8.54	0.9	2.23	0.02	0.06	76	100
	(b) Two obstacles	HYB-DQN-V	20.18	109.7	18.39	1.0	2.68	0.04	0.06	98	100
		HYB-DDPG	17.38	86.21	14.31	0.96	2.62	0.03	0.05	97	100
	(c) U-shape obstacle	DDPG	0.79	10.65	7.85	0.98	1.91	0.08	0.58	74	100
		HYB-DQN-V	93.49	1019.9	254.09	0.84	2.93	0.09	0.16	118	40
		HYB-DDPG	15.74	99.04	17.95	0.87	2.18	0.02	0.07	75	100
	(d) Dynamic obstacle (face-to-face)	MPC	16.07	129.18	25.02	0.44	1.69	0.03	0.04	69	100
		DDPG	0.82	10.96	7.95	0.74	1.94	0.08	0.48	77	98
		HYB-DDPG	11.13	69.7	13.01	0.9	2.85	0.02	0.05	79	100
Scene 2	(e) Sharp turn with an obstacle	DQN-V	0.69	10.74	5.49	0.87	1.44	0.19	0.7	150	78
		DDPG	0.74	10.61	5.35	0.53	1.37	0.09	0.56	158	96
		HYB-DQN-V	11.8	135.32	15.95	0.49	1.8	0.02	0.04	149	100
		HYB-DDPG	14.82	98.26	15.47	0.43	1.63	0.01	0.03	151	100
	(f) U-turn with an obstacle	DQN-V	0.75	10.48	7.80	0.42	0.87	0.17	0.67	149	40
		DDPG	0.73	10.48	5.00	0.63	1.05	0.13	0.51	169	74
		HYB-DQN-V	14.19	99.66	15.10	0.43	1.67	0.02	0.03	148	100
		HYB-DDPG	11.91	75.17	11.80	0.4	1.42	0.02	0.02	147	100

focuses on maneuverability through a sharp turn (e) and a U-turn (f), each with a static obstacle on the course.

2) Results: The model's performance is gauged via these metrics:

- **Computation time:** The mean, maximum, and standard deviation (std) of computation times across models are recorded, aiming for lower values.
- **Deviation:** The mean and maximum distances from the reference path are measured, with smaller deviations indicating better accuracy.
- **Action smoothness:** Smoothness is calculated as the mean of the approximated jerk of the AMR's speed and angular speed. Lower values indicate smoother actions, which are preferable for practical AMR applications.
- **Finish time step:** The number of steps needed to complete the evaluation. Consistency across models is desired, as large variances suggest significant deviations from the intended speed.
- **Success rate:** The percentage of successful runs (which is defined as the robot reaching the goal without any collision) and total runs of a given scene.

In the evaluation, pure MPC and DDPG methods are the baseline, and we also compare with the image-based DQN (DQN-V) and hybrid MPC-DQN (HYB-DQN-V) approaches [14]. For a fair comparison, all methods are tested across 50 episodes on identical hardware, with outcomes detailed in Table I (cases with success rates under 30% are discarded). The evaluation of the dynamic obstacle was performed using a larger Δd compared to the other test cases to enable the DDPG model to generate a reference trajectory for the hybrid model more consistent with the DDPG model's trajectory. Table I highlights that the image-based DDPG hybrid model (HYB-DDPG) achieved 100% in all scenarios, significantly outperforming the baseline and DQN-based algorithms. Despite challenges in certain tests, the DDPG model effectively supported HYB-DDPG, offering viable reference trajectories where both MPC and DDPG individually falter. The occasional shortfall of the DDPG model could be attributed to training data gaps and low image resolution [14]. Nonetheless, HYB-DDPG reduces computation times versus MPC, albeit with an increase in completion times attributed to DDPG's cautious approach, which resulted in an improved success rate.

B. Multi-agent

In the multiple-robot simulation, three scenarios are devised to examine the potential safety risk and inefficiency caused by unexpected non-convexity or circular wait. The circular wait appears because each agent is independently adapting to the others, which is a natural drawback of distributed training. However, as mentioned, this trade-off earns simplicity in modeling, training, and inferencing. Meanwhile, by combining DRL with MPC which has the information of decisions of other robots, more stable cooperative behaviors emerge among robots.

1) Use Cases: The following scenarios are designed to showcase the superiority of the proposed approach, which are visualized in Fig. 2:

- **Scene 3** - Alternative for encounter avoidance: Two robots proceed in opposite directions through a narrow corridor. For each robot, the other one and the environment together form a nonconvex barrier. Initial conditions prevent

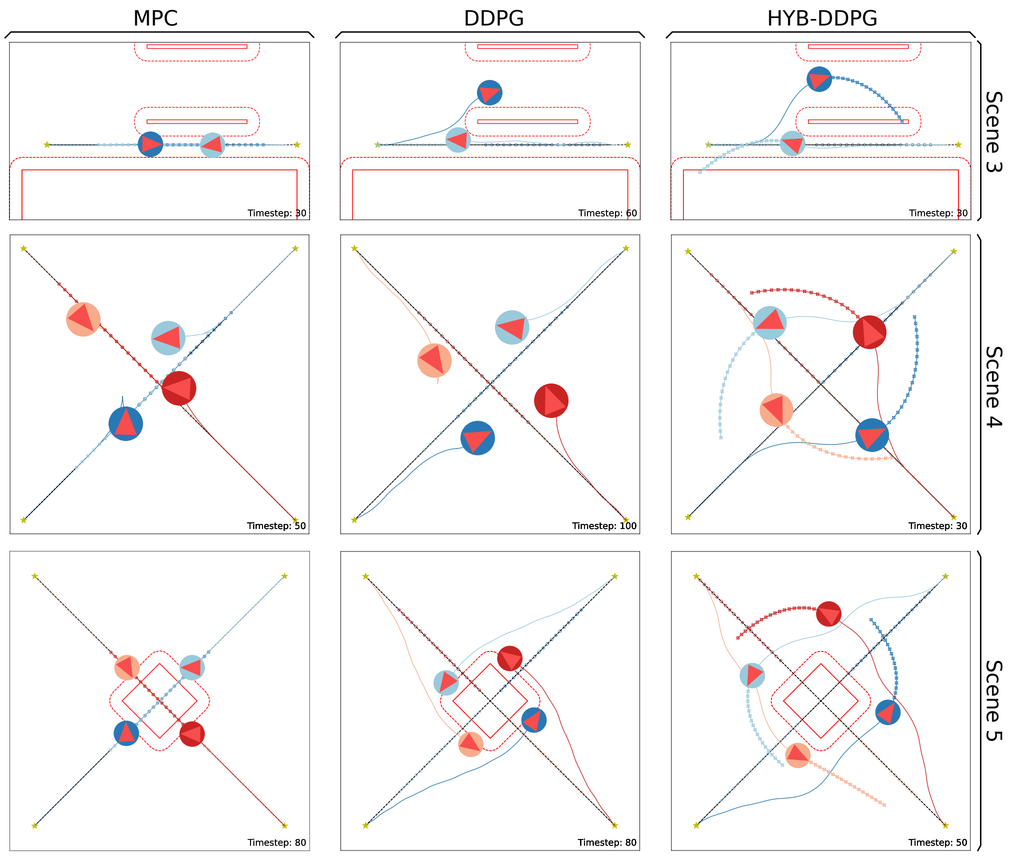

Fig. 2. Scenes for multi-agent evaluation. Circles filled with different colors stand for different robots. Polygons represented by red lines are the original obstacles while red dashed lines are padded obstacles. Black dashed lines are reference paths. Colored lines are traversed trajectories of corresponding robots. Small hollow circles are original reference trajectories while hollow squares in the HYB-DDPG case are RL reference. Crosses mark the actual reference followed by the robot in the HYB-DDPG case.

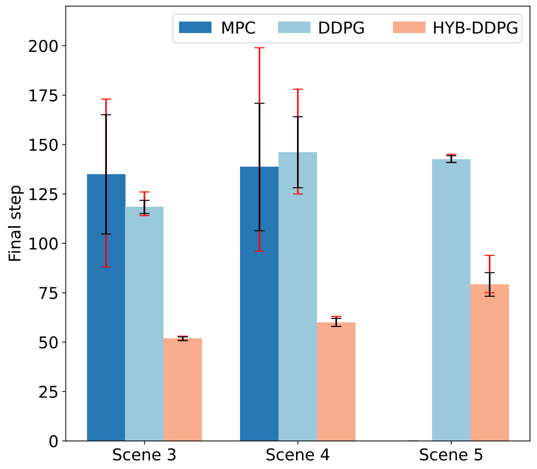

Fig. 3. Evaluation of the multi-agent scenarios. Each bar shows the average time step to reach the destinations of the corresponding category. Black error bars indicate the standard deviations of instances in the corresponding category. Red error bars show the maximum and minimum values. MPC fails in Scene 5.

foreseeing this conflict. With the corridor's size beyond its predictive horizon, MPC fails to navigate this non-convexity. In contrast, DDPG anticipates the encounter, guiding a detour to prevent it. The hybrid approach, informed by DDPG, not only avoids the bottleneck but also outperforms DDPG alone in finish time.

- **Scene 4** - Crossing of intersection: In this scenario, four robots are initialized at the corners of an empty map and move to the opposite corners. As a classic circular wait example, each robot makes its decision according to the decisions of the others and is besieged by them. MPC struggles with this scenario, either failing or delaying due to excessive calculation. DDPG performs better, likely due to being trained to avoid collisions with significant penalties, promoting obstacle avoidance behaviors. What's more, since each agent is equipped with the same DRL model that is trained in a distributed way, they all tend to present the same reaction (e.g. in Fig. 2, moving to the right side is the mutual option), which is beneficial in fixing the deadlock. Once again, the hybrid approach excels by combining the uniform detour clue from DRL and the decisive action from MPC.
- **Scene 5** - Traffic at the roundabout: In this scenario, MPC fails against nonconvex obstacles including a central static barrier and other robots. Similar to Scene 4, DDPG offers a more organized strategy in which each robot makes a comparable turn,

In the multi-agent simulation, distributed training showcases intelligent agent behaviors, underscoring the benefits of integrating MPC with DRL. In Scene 3, DDPG provides an alternative to avoid potential deadlock between agents, while MPC accelerates and smoothes the trajectory tracking process. Scenes 4 and 5 display the emerging harmonious behaviors of the DRL agents, which is reasonable since all agents are equipped with the same decision-making kernel.

2) Results: The multi-agent evaluation focuses on the success rate and finish time step. The hybrid method excels in mean finish time across all scenes, as Fig. 3 illustrates, also demonstrating stability through low variance. Both DDPG and HYB-DDPG maintain 100% success rates across all scenes, whereas MPC faces one failure in Scene 3 (90% success rate) and fails in Scene 5 (0% success rate). The evaluation demonstrates that the proposed approach successfully integrates the strengths of MPC and DRL, enabling scalability from single-agent to multi-agent scenarios.

VII. CONCLUSION

In this study, we developed a DDPG model for safe trajectory planning in dynamic settings using BEV vision data and integrated it with MPC for detailed control. The evaluations across various use cases show that this DDPG-MPC hybrid outperforms previous DQN models, particularly in success rate. We also tested the model in multi-robot scenarios, finding that DDPG and MPC not only boost individual robot performance but also enhance cooperative behavior in robot fleets. Future work will focus on refining the DDPG model's planning efficiency and exploring how varying lookahead distances impact performance, alongside optimizing the trajectory generation from the DRL model.

REFERENCES

[1] Z. Zhang, H. Hajieghrary, E. Dean, and K. Åkesson, "Prescient collision-free navigation of mobile robots with iterative multimodal motion prediction of dynamic obstacles," *Robotics and Automation Letters*, vol. 8, no. 9, pp. 5488–5495, 2023.

[2] M. Kokot, D. Miklić, and T. Petrović, "A unified MPC design approach for AGV path following," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 4789–4796.

[3] A. Dahlin and Y. Karayiannidis, "Obstacle avoidance in dynamic environments via tunnel-following MPC with adaptive guiding vector fields," in *Conference on Decision and Control (CDC)*. IEEE, 2023, pp. 5784–5789.

[4] F. Bertilsson, M. Gordon, J. Hansson, D. Möller, D. Söderberg, Z. Zhang, and K. Åkesson, "Centralized versus distributed nonlinear model predictive control for online robot fleet trajectory planning," in *International Conference on Automation Science and Engineering (CASE)*. IEEE, 2022, pp. 701–706.

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

[6] Ángel Madridano, A. Al-Kaff, D. Martín, and A. de la Escalera, "Trajectory planning for multi-robot systems: Methods and applications," *Expert Systems with Applications*, vol. 173, p. 114660, 2021.

[7] H. Sun, W. Zhang, R. Yu, and Y. Zhang, "Motion planning for mobile robots — focusing on deep reinforcement learning: A systematic review," *IEEE Access*, vol. 9, pp. 69 061–69 081, 2021.

[8] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.

[9] T.-W. Weng, K. D. Dvijotham, J. Uesato, K. Xiao, S. Gowal, R. Stanforth, and P. Kohli, "Toward evaluating robustness of deep reinforcement learning with continuous control," in *International Conference on Learning Representations (ICLR)*, 2020.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2016.

[11] H. Sang and S. Wang, "Motion planning of space robot obstacle avoidance based on DDPG algorithm," in *International Conference on Service Robotics (ICoSR)*, 2022, pp. 175–181.

[12] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, I. G. Moreno, and P. Campoy, "A deep reinforcement learning technique for vision-based autonomous multirotor landing on a moving platform," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1010–1017.

[13] C. Do, C. Gordillo, and W. Burgard, "Learning to pour using deep deterministic policy gradients," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3074–3079.

[14] Z. Zhang, Y. Cai, K. Ceder, A. Enliden, O. Eriksson, S. Kylander, R. Sridhara, and K. Åkesson, "Collision-free trajectory planning of mobile robots by integrating deep reinforcement learning and model predictive control," in *International Conference on Automation Science and Engineering (CASE)*. IEEE, 2023, pp. 1–7.

[15] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design (2nd Edition)*. Nob Hill Publishing, LLC, 2020.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[17] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, p. 279–292, 1992.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529–533, 2015.

[19] A. Sathya, P. Sopasakis, R. Van Parys, A. Themelis, G. Pipeleers, and P. Patrinos, "Embedded nonlinear model predictive control for obstacle avoidance using PANOC," in *European Control Conference (ECC)*. IEEE, 2018, pp. 1523–1528.

[20] J. Eschmann, *Reinforcement Learning Algorithms: Analysis and Applications*. Springer International Publishing, 2021, pp. 25–33.

[21] P. Sopasakis, E. Fresk, and P. Patrinos, "OpEn: Code generation for embedded nonconvex optimization," in *IFAC World Congress*, 2020.

[22] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[23] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations (ICLR)*, 2016.

[24] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, vol. 55, p. 895–943, 2022.