Collision-Free Trajectory Planning of Mobile Robots by Integrating Deep Reinforcement Learning and Model Predictive Control

Ze Zhang[1], Yao Cai[1],
Kristian Ceder, Arvid Enliden, Ossian Eriksson, Soleil Kylander, Rajath Sridhara, and Knut Åkesson[1]

***Abstract*— In this paper, we present an efficient approach to real-time collision-free navigation for mobile robots. By integrating deep reinforcement learning with model predictive control, our aim is to achieve both collision avoidance and computational efficiency. The methodology begins with training a preliminary agent using deep Q-learning, enabling it to generate actions for next time steps. Instead of executing these actions, a reference trajectory is generated based on them, which avoids obstacles present on the original reference path. Subsequently, this local trajectory is employed within an MPC trajectory-tracking framework to provide collision-free guidance for the mobile robot. Experimental results demonstrate that the proposed DQN-MPC hybrid approach outperforms pure MPC in terms of time efficiency and solution quality.**

I. INTRODUCTION

Mobile Robots (MRs) have become increasingly significant in various aspects of daily life and industrial applications. In particular, MRs have been widely utilized in industrial settings to alleviate human labor and enhance operational efficiency. Automatic Guided Vehicles (AGVs) [1] represent a class of MRs that adhere to predetermined paths and employ sensors to passively evade collisions, meaning that AGVs decelerate and halt if their pre-defined paths are obstructed. In contrast, Autonomous Mobile Robots (AMRs) exhibit greater adaptability in terms of navigational strategies and decision-making capabilities [2]. A key advantage of AMRs is their proactive collision avoidance, allowing them to deviate from the original paths to circumvent obstacles. This online obstacle avoidance necessitates a sophisticated controller that accounts for obstacles. In this work, we propose a novel approach for online obstacle avoidance.

The classic motion planning and control for a mobile robot encompasses three steps [3], [4]: path planning, trajectory generation, and trajectory tracking control. Path planning [3], [4] aims to identify a collision-free route connecting the starting state to the goal state of the mobile robot. Utilizing the reference path, a reference trajectory comprises a sequence of path nodes, coupled with the corresponding time when the robot should reach each node. The tracking and control problem entails generating actions, such as the velocity or acceleration, to ensure the robot remains in close proximity to the reference trajectory while avoiding collisions. Path planning and trajectory generation can be performed online or offline, while the control typically occurs in real time. Online motion planning is crucial for addressing environmental changes, including unexpected obstacles. The term *unexpected* refers to obstacles that were not present during the reference path planning. In this study, we assume that a high-level job scheduler [5] provides the reference path. The focus of this work lies in the online trajectory generation and control of a mobile robot.

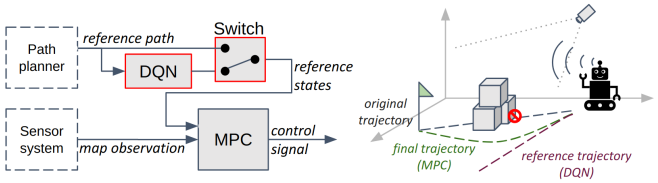

Fig. 1. Blocks outlined by dashed lines are assumed to be given in this work, while all other blocks are implemented. The blocks with red borders indicate the main contributions. On the right side, there are two types of sensors shown: a vision system, which is a camera installed in the ceiling, and a scanner such as a lidar, which is attached to the robot. The DQN generates an alternative reference trajectory, diverging from the original one if it's obstructed. Lastly, the MPC solver makes the final decision regarding the control signals.

Model Predictive Control (MPC) has emerged as a prominent technique for ensuring collision-free trajectory tracking and has demonstrated efficacy in numerous applications [6]–[9]. Owing to its ability to treat obstacles as constraints and its receding horizon control strategy, MPC can predict potential collisions within the horizon and propose an action for collision-free trajectory tracking. Nevertheless, complex optimization problems often necessitate time-consuming numerical and iterative solvers [10], and the computational time is difficult to predict due to the uncertain number of iterations required, which may result in a breach of real-time constraints. Another issue with MPC solvers pertains to their susceptibility to getting stuck in local optima [11]. Recently, Reinforcement Learning (RL), particularly Deep Reinforcement Learning (DRL), emerges as an alternative for motion planning and control problems [12]. In comparison to traditional optimization solvers, the computational time is essentially constant and predictable for DRL methods. The initial motion-planning task assigned to RL is in two-dimensional grid maps with simple action spaces, such as the Frozen Lake task [13]. A different strategy for estimating feasible and flexible reference trajectories involves generating physical commands, such as velocity [14]–[17], which allows MRs to be directly controlled by RL algorithms. However, a DRL controller cannot guarantee the success rate of collision-free navigation, as it is not interpretable and often produces unstable and fluctuating actions even when there are no obstacles blocking the mobile robot [14], [17].

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

*This work is supported by the AIMCoR project and the Vinnova project, AIHURO (Intelligent human-robot collaboration).

In this paper, we propose a hybrid approach integrating MPC and DQNs for collision-free navigation of AMRs, as illustrated in Fig. 1. This combination addresses the separate limitations of MPC and DQNs. Specifically, MPC generates speed and angular velocity commands for the MR based on the heuristic reference trajectory from the DQN. The MR can smoothly reach its goal without colliding with unexpected obstacles in numerous use cases, and the proposed method outperforms both MPC and DQN comprehensively. We also employ two alternatives of sensors: a laser or lidar scanner [14] on the robot or a vision system [9] consisting of multiple cameras mounted in the ceiling to provide object detection and semantic segmentation. The main contributions are threefold:

- Training DQNs for collision-free navigation of MRs following given reference path and reference speed.
- Combining the DQN and MPC to conduct stabler and smoother (compared to the pure DQN solution) collision-free reference path tracking with better real-time performance and obstacle-avoidance ability (compared to the pure MPC solution).
- Designing a switch between the pure MPC solver and the hybrid solver to improve the overall performance.

An extensive evaluation is provided with multiple metrics covering different aspects of the trajectory tracking problem.

II. Preliminary

To dispel possible confusion, the word *agent* is used to indicate the controlled target, i.e., the AMR. The output of MPC is the *control signal/input* to the agent, while the output of the DQN is its suggested *action* to the agent. To distinguish the different *states* in the MPC and the DQN, $\boldsymbol{x}$ represents the state in MPC and $\boldsymbol{s}$ represents the state in the DQN. For convenience, a set of non-negative integers in the closed interval $[a, b]$, s.t. $b > a \geq 0$, is written as $\mathbb{N}_{[a,b]}$.

A. Model Predicted Control

Model Predictive Control (MPC) [18] is a constrained-optimization-based control approach with a receding horizon fashion. It considers the future states of the agent within a horizon and computes a series of optimal control inputs in terms of an objective, which do not violate given constraints. Given a discrete motion model $\boldsymbol{x}_{k+1} = f^{\text{MPC}}(\boldsymbol{x}_k, \boldsymbol{u}_k)$ with k being the time step, where $\boldsymbol{x}$ is the state and $\boldsymbol{u}$ is the control input, assuming $\boldsymbol{z}_k = [x_k, y_k]^T$ is part of $\boldsymbol{x}_k$ indicating the position of the robot, a general form of MPC problems on collision-free trajectory tracking with horizon N is,

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

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

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

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

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

where J_N is the terminal cost, Q_x and Q_u are the penalty weights for the state and the control input, $\tilde{\boldsymbol{x}}_k$ and $\tilde{\boldsymbol{u}}_k$ are the reference state and control input, $\bar{\boldsymbol{x}}$ is the given initial state, U_k is the constraint on the control input (normally a box constraint with min-max limits), and $\mathcal{O}$ and $\mathcal{D}_k$ are the static and dynamic obstacle areas respectively. In practice, only the first control input $\boldsymbol{u}_0$ is used at every time step, which means an optimization question needs to be solved frequently. If the optimization space is complex or non-convex, the solving time can be prolonged and barely predictable.

B. Temporal Difference Reinforcement Learning

An RL task [19] is a decision-making process for an agent based on rewards received through interaction with an environment. The goal is to learn a policy that maximizes the cumulative rewards the agent receives over time. During the training, the agent explores the environment until the maximal time is reached or terminal conditions are triggered, which is called an *episode*.

A Markov Decision Process (MDP) [19] is a discrete-time stochastic process to model RL tasks that satisfies the Markov property. An MDP is defined by (S, A, P, R, γ), where S is the state set of the environment; A is the action set of the agent; P is the transition function determining the probability from one state to another state given an action; R is the reward function that specifies the reward from one state to another state by taking an action; and $\gamma \in [0, 1]$ is the discount factor determining the importance of future rewards. Define the discounted cumulative rewards as *Return* $G_k = \sum_{i=0}^{\infty} \gamma^k R_{k+i+1}$. A stochastic policy $\pi : \boldsymbol{s} \to \boldsymbol{a}$ is a mapping representing the probability from the state space to the action space. The optimal policy at a state is the one with the maximum return, i.e.,

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \underbrace{\mathbb{E}_\pi \left[G_k \mid S_k = \boldsymbol{s} \right]}_{:= V_\pi(s)}, \quad (6)$$

where the expectation is called value V of state $\boldsymbol{s}$ following policy π. Similarly, the state-action value, also known as the Q-value is defined as

$$q_\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_\pi \left[G_k \mid S_k = \boldsymbol{s}, A_k = \boldsymbol{a} \right], \quad (7)$$

where the value is also determined by the action $\boldsymbol{a}$ at time step k. According to the Bellman Expectation Equation for MDPs, the q-value can be written recursively,

$$q_\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_\pi \left[R_{k+1} + \gamma q_\pi(S_{k+1}, A_{k+1}) \mid S_k = \boldsymbol{s}, A_k = \boldsymbol{a} \right]. \quad (8)$$

Temporal Difference (TD) learning [19] is a type of RL algorithm that learns by updating the estimate of the value function based on the difference between expected and actual rewards received at each time step. Q-learning [19] is a variant of TD learning, which iteratively maximizes the Q-value instead of the state value V. Q-learning stores estimated Q-values $Q(\boldsymbol{s}, \boldsymbol{a})$ in a look-up table called the Q-table, and updates the table according to the following rule

$$Q(\boldsymbol{s}, \boldsymbol{a}) \leftarrow Q(\boldsymbol{s}, \boldsymbol{a}) + \alpha \left[r + \gamma \max_{\boldsymbol{a}' \in A} Q\left(\boldsymbol{s}', \boldsymbol{a}'\right) - Q(\boldsymbol{s}, \boldsymbol{a}) \right], \quad (9)$$

where $\boldsymbol{s}'$ is the state at the next time step, r is the reward at $\boldsymbol{s}'$, and α is the learning rate.

C. Deep Q-Learning

Deep Q-learning [20] is the combination of Q-learning and neural networks. In some environments, there can be too many states leading to an enormous Q-table, which is impossible for Q-learning to converge due to time and hardware limitations. To cope with these high-dimensional scenarios, neural networks are introduced to replace the Q-table, which are known as Deep Q-Networks (DQNs). DQNs have the advantage of being highly efficient since no enumerating is needed. Deep Q-learning uses the experience replay technique which is a technique randomly sampling experiences from a replay buffer to break the correlation between consecutive samples and improve the stability of the training. It also uses a copy of the DQN, called the target network which is periodically updated with the weights of the current DQN. The target network is used to generate the target Q-values in the Q-learning update equation.

Deep Q-learning aims to learn the optimal Q-values for all state-action pairs by minimizing the mean squared error between the estimated Q-values from the DQN and the target Q-values from the target network. The loss function to update the DQN is

$$\mathcal{L}_\theta = \left[r + \gamma \max_{\boldsymbol{a}' \in A} Q\left(\boldsymbol{s}', \boldsymbol{a}'; \theta^{\text{target}}\right) - Q(\boldsymbol{s}, \boldsymbol{a}; \theta) \right]^2, \quad (10)$$

similarly to (9), where $\boldsymbol{s}'$ is the state at the next time step, r is the reward at $\boldsymbol{s}'$. Moreover, θ represents the parameter for the DQN and θ^{target} is the parameter of the target network.

D. Modeling of Obstacles

In this work, static obstacles are modeled as convex polygons and represented by sets of linear inequalities, as described in [7]. Let $\mathcal{O} = \cup_n O_n$ be the space of n static obstacles, assuming that a static obstacle O has E edges, $O = \{\boldsymbol{p} \in \mathbb{R}^2 \,|\, \boldsymbol{b}_i - \boldsymbol{a}_i^T \boldsymbol{p} > 0, \forall i \in \mathbb{N}_{[1,E]}\}$, where $\boldsymbol{a}_i$, $\boldsymbol{b}_i$ are coefficients defining a half-space.

Dynamic obstacles, such as humans, are modeled by two-dimensional ellipses and assumed to have constant velocities. For an ellipse centered at $\boldsymbol{\mu} = [\mu_x, \mu_y]^T$ with axes $\boldsymbol{\sigma} = [\sigma_x, \sigma_y]^T$, a variable ι indicating if a point $\boldsymbol{p} = [p_x, p_y]^T$ is inside the ellipse can be defined as (11). The time-step k is omitted here for brevity. The indicator is zero outside and positive inside the ellipse, and its value increases for points closer to the center $\boldsymbol{\mu}$ with a maximum of 1. Formally

$$\iota(\boldsymbol{p} \,|\, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \max\left\{0, \left[1 - (\frac{p_x - \mu_x}{\sigma_x})^2 - (\frac{p_y - \mu_y}{\sigma_y})^2\right]\right\}. \quad (11)$$

With the help of the indicator, the area $\mathcal{D}$ occupied by N_d ellipses can be defined as:

$$\mathcal{D} = \{\boldsymbol{p} \in \mathbb{R}^2 \,|\, \exists\, i \in \mathbb{N}_{[1,N_d]},\, \iota(\boldsymbol{p}|\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i) > 0\}. \quad (12)$$

To consider the size of the agent, all obstacles are inflated with a margin equal to the size of the agent plus an extra safety margin, as shown in Fig. 2.

III. Problem Formulation

For a mobile robot with a discrete motion model $\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k)$ and k is the time step, where $\boldsymbol{x} \in \mathbb{R}^{n_x}$ is the state and $\boldsymbol{u} \in \mathbb{R}^{n_u}$ is the control input, $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$, given a reference speed v_{ref} and a reference path with k path nodes, where each node $\boldsymbol{p}_i = [x_i, y_i]^T$ represents a two-dimensional position and the first node $\boldsymbol{p}_0$ represents the current location of the robot,

$$\mathcal{P}_{\text{ref}} = \langle \boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_k \rangle, \quad (13)$$

and a set of n_o static obstacles $\mathcal{O} = \{O^1, O^2, \ldots, O^{n_o}\}$ and n_d dynamic obstacles $\mathcal{D}_k = \{D_k^1, D_k^2, \ldots, D_k^{n_d}\}$ at time step k, the task is to conduct collision-free navigation for the mobile robot to follow the reference path. Assuming $\boldsymbol{z}_k = [x_k, y_k]^T$ is part of $\boldsymbol{x}_k$ indicating the position of the robot, the collision-free condition at time step k is $\boldsymbol{z}_k \notin (\mathcal{O} \cup \mathcal{D}_k)$.

Note that the set of static obstacles $\mathcal{O}$ and the reference speed v_{ref} are assumed to be time-invariant, but they can update over time. In this paper, we try to solve the problem using model predictive control for trajectory tracking with the aid of deep Q-learning which provides a heuristic reference trajectory to avoid collisions.

IV. Obstacle Avoidance with Model Predictive Control

As mentioned in (1), MPC needs reference states and control inputs to evaluate the objective. In the case of trajectory tracking, the reference control signal can be the reference speed v_{ref}. The reference states at time step k can be sampled from the reference path $\mathcal{P}_{\text{ref}}$, called the local reference trajectory $\mathcal{T}_k^{\text{ref}}$. A straightforward way to sample a reference trajectory is to start from the current location of the agent and sample N steps with the speed v_{ref} along $\mathcal{P}_{\text{ref}}$. Note that $\mathcal{T}_k^{\text{ref}}$ doesn't consider unexpected obstacles. To improve the smoothness of control inputs, we also penalize the acceleration of the agent. The overall objective of the agent's states and control signals is $J_R(k) = ||\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_k||^2_{\boldsymbol{Q}_x} + ||\boldsymbol{u}_k - \tilde{\boldsymbol{u}}_k||^2_{\boldsymbol{Q}_u} + ||\boldsymbol{u}_k - \boldsymbol{u}_{k-1}||^2_{\boldsymbol{Q}_a}$. Finally, as suggested in [10], a soft constraint for dynamic obstacle avoidance is added,

$$J_{\mathcal{D}}(\boldsymbol{x}_k) = \sum_{n=1}^{N_d} ||\iota(\boldsymbol{z}_k \,|\, \boldsymbol{\mu}_{n,k}, \boldsymbol{\sigma}_{n,k})||^2_{\boldsymbol{Q}_{\mathcal{D}}}. \quad (14)$$

The full MPC formation with the sampling time Δt, is

$$\min_{\boldsymbol{u}_{0:N-1}} \quad ||\boldsymbol{x}_N - \tilde{\boldsymbol{x}}_N||^2_{\boldsymbol{Q}_N} + \sum_{k=0}^{N-1} \left[J_R(k) + J_{\mathcal{D}}(\boldsymbol{x}_k)\right] \quad (15)$$

$$\text{s.t.} \quad \boldsymbol{x}_0 = \bar{\boldsymbol{x}}, \quad (16)$$

$$\boldsymbol{x}_{k+1} = f^{\text{MPC}}(\boldsymbol{x}_k, \boldsymbol{u}_k),\, k \in \mathbb{N}_{[0,N-1]}, \quad (17)$$

$$\boldsymbol{u}_k \in [\boldsymbol{u}_{min}, \boldsymbol{u}_{max}],\, k \in \mathbb{N}_{[0,N-1]}, \quad (18)$$

$$\frac{\Delta \boldsymbol{u}_k}{\Delta t} \in [\dot{\boldsymbol{u}}_{min}, \dot{\boldsymbol{u}}_{max}],\, k \in \mathbb{N}_{[0,N-1]}, \quad (19)$$

$$\boldsymbol{z}_k \notin \mathcal{O} \cup \mathcal{D}_k,\, k \in \mathbb{N}_{[0,N]}, \quad (20)$$

where $\Delta \boldsymbol{u}_k = \boldsymbol{u}_k - \boldsymbol{u}_{k-1}$, $\boldsymbol{u}_k$ and $\frac{\Delta \boldsymbol{u}_k}{\Delta t}$ are bounded by the minimal and maximal limitations. Compared to (1), the penalty on the acceleration and the soft constraint of dynamic obstacle avoidance are augmented in (15).

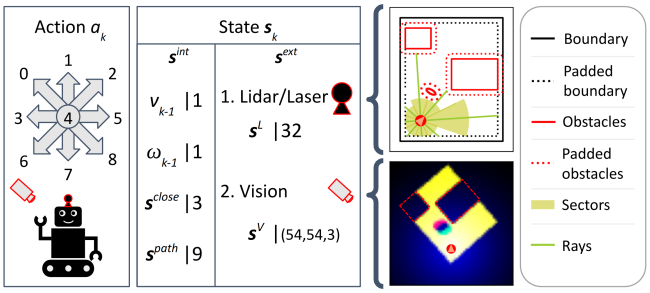

Fig. 2. The action and state spaces of the DQN agent. Subscript k is omitted for detailed state variables. There are eight alternative actions and each action is an acceleration in that direction. The state includes two parts. The internal state $\boldsymbol{s}^{\text{int}}$ is composed of 1-dimensional speed, 1-dimensional angular velocity, 3-dimensional closest-point-on-path indication, and 9-dimensional upcoming-path-nodes indication. The external state $\boldsymbol{s}^{\text{ext}}$ can be either the laser/lidar detection or the visual detection from cameras.

V. Reference Trajectory from Deep Q-Learning

The MPC method defined in (15)-(20) can handle most situations when the reference path is not blocked. However, if the path is blocked, the solver may need a longer time to iteratively calculate a feasible numeric solution or fails totally. This is because the local reference trajectory is not flexible to dynamic environments and the burden to find a feasible detour lies on MPC. In this section, we introduce a method to generate new reference trajectories via deep Q-learning. The action, state, and reward in the deep Q-learning method are defined in the following subsections (the details of the action and state spaces are also shown in Fig. 2).

A. State Observation

The DQN state $\boldsymbol{s}_k$ at time step k includes the internal observation $\boldsymbol{s}_k^{\text{int}}$ and the external observation $\boldsymbol{s}_k^{\text{ext}}$. The subscript k is omitted if there is no ambiguity. The internal observation $\boldsymbol{s}^{\text{int}} \in \mathbb{R}^{14}$ contains the previous speed v_{k-1} and angular velocity ω_{k-1} of the agent, the closest-point-on-path indication $\boldsymbol{s}^{\text{close}}$ to tell the agent its relative location to the reference path, and the upcoming-path-nodes indication $\boldsymbol{s}^{\text{path}}$ to tell the agent the relative locations of upcoming path nodes. Specifically, $\boldsymbol{s}^{\text{close}} = [\cos(\beta^{\text{close}}), \sin(\beta^{\text{close}}), \bar{d}^{\text{close}}]^T$, where β^{close} is the angle between the agent's heading and the direction from the agent to the closest point on the reference path, and $\bar{d}^{\text{close}}$ is the normalized distance [14] of the distance d^{close} from the agent to the closest point on the reference path, as in (21) where $d_{\max}$ is a saturation constant functioning as a soft distance cut-off. Similarly, $\boldsymbol{s}^{\text{path}}$ is angles and distances from the agent to the upcoming path nodes. We choose three upcoming path nodes thus $\boldsymbol{s}^{\text{path}} \in \mathbb{R}^9$.

$$\bar{d} = 2 \cdot (1 + e^{-2d/d_{\max}})^{-1} - 1. \tag{21}$$

The external observation $\boldsymbol{s}^{\text{ext}}$ can be either rays-and-sectors observation (from laser/lidar scanners) or image observation (from cameras). The rays-and-sectors observation is an improvement based on the ray observation. A contradiction of the ray observation [14] is that if the detecting rays are not sufficiently dense, small obstacles may be invisible to the agent; if the number of rays is too large, the training of DQNs may be unstable [15]. The rays-and-sectors observation [15] clusters several rays as a sector whose range is determined by the closest distance of rays belonging to this sector. To consider dynamic obstacles, the detection is concatenated with the previous detection from Δk time steps ago.

The other option is to use the cameras in the ceiling. The vision system can provide semantic segmentation of accessible areas and static obstacles, and also detect moving objects including the agent and other dynamic obstacles. As shown in Fig. 2, similar to [21], an image centered at the agent's position is cropped from the camera detection. The image has three channels: a distance map to the agent where each pixel is the distance to the agent, and two occupancy grid maps where one map is in the current time and the other one is from Δk time steps back. In Fig. 2, the image from the camera detection is such a three-channel image, where the two static obstacles are marked by red dashed blocks, the green shadow represents the current position of the elliptical dynamic obstacle, the red shadow represents the position of the elliptical dynamic obstacle $\Delta k = 5$ time steps ago.

B. Action Space

Since DQNs have finite discrete action spaces, one choice is to preset some action modes [15], [17]. In this work, to have more velocity options, the action $a \in \{0, 1, \ldots, 8\}$ is the combination of the linear acceleration $\{-1, 0, 1\}$ and angular acceleration $\{-1, 0, 1\}$, as described in Fig. 2. In options of linear acceleration, -1 means the maximal deceleration, 0 means constant velocity, and 1 means the maximal acceleration. In options of angular acceleration, -1 means turning left with the maximal angular acceleration, 0 means keeping the heading, and 1 means turning right with the maximal angular acceleration.

C. Reward Function

The selection of the reward function is crucial in RL, which should ensure the agent follows the given path as much as possible and reach the goal without collisions. As in [14], we also select the path-progress reward R^{path}, the reach-goal reward R^{goal}, and the negative collision reward R^{col}. In addition, a path-deviation penalty R^{dev} and a speeding penalty R^{speed} are used. The terminal condition for an episode is that the maximal time is reached or a collision happens. The overall reward composition at time step k is:

$$R_k = R_k^{\text{col}} + R_k^{\text{goal}} + R_k^{\text{path}} + R_k^{\text{speed}} + R_k^{\text{dev}}. \tag{22}$$

Among the reward terms,

$$R_k^{\text{col}} = \begin{cases} -\lambda^{\text{col}}, \text{ if collision happens} \\ 0, \text{ otherwise} \end{cases}, \tag{23}$$

$$R_k^{\text{goal}} = \begin{cases} \lambda^{\text{goal}}, \text{ if the agent reaches the goal} \\ 0, \text{ otherwise} \end{cases}, \tag{24}$$

$$R_k^{\text{path}} = \lambda^{\text{path}}(l_k - l_{k-1}), \tag{25}$$

$$R_k^{\text{speed}} = \lambda^{\text{speed}} \max(0, (v_k - v_{\text{ref}})), \tag{26}$$

$$R_k^{\text{dev}} = -\lambda^{\text{dev}}(d_k^{\text{close}})^2, \tag{27}$$

where all λs are corresponding positive reward weights, l_k the progress measured by the length from the start point to

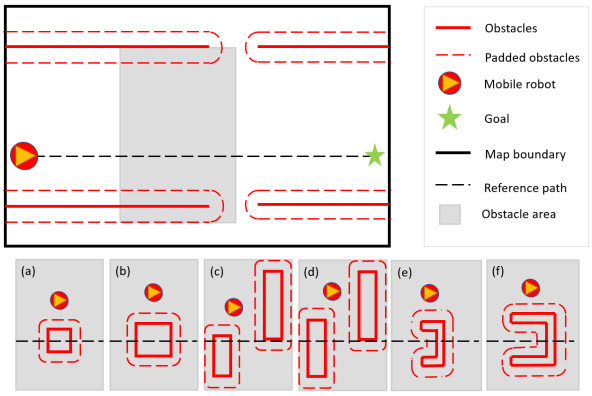

Fig. 3. Static obstacles tested in Scene 1. The gray area is the area where different types of obstacles are placed. Under the map, six obstacles are shown with the agent beside them to compare the size.

the current closest point to the agent on the reference path, v_k is the current speed, d_k^{close} is the closest distance from the agent to the reference path as previously introduced.

D. Reference Trajectory Generation and Switch Trick

The output of DQNs is the Q-values of all action indices. During inference, the action with the largest Q-value is selected, which means a deterministic policy. An action index represents a pair of linear and angular accelerations. The corresponding speed v^{DQN} and angular velocity ω^{DQN} of the agent can be obtained by integrating the action from the DQN. The produced action can be interpreted as a tendency that the agent should follow to maximize the reward, i.e., progressing to the goal and following the reference path without collisions. This can be a reference to MPC when the agent approaches an obstacle by indicating a local trajectory to evade collisions. To generate a local trajectory, the agent is simulated to run N steps. In the first step, the original control signal $\boldsymbol{u}_0^{\text{DQN}} = [v^{\text{DQN}}, \omega^{\text{DQN}}]^T$ generated by the DQN is used. For the rest steps, to improve stability, an extra reference speed v'_{ref} is used instead. As for the angular velocity, a decay constant λ^{decay} is multiplied at every step. The overall control inputs are $\boldsymbol{u}_0^{\text{DQN}}$ at time step 0 and $\boldsymbol{u}_k^{\text{DQN}} = [v'_{\text{ref}}, (\lambda^{\text{decay}})^k \omega^{\text{DQN}}]^T$ at time step $k > 0$. With this control sequence, a new local reference trajectory $\mathcal{T}'_k$ is obtained through the agent's motion model.

For MPC, there are now two reference trajectories to choose from, the original $\mathcal{T}^{\text{ref}}$ by sampling the reference path and the new $\mathcal{T}'$ from the DQN. The original one is obviously stable but does not handle changes in the environment; the new one considers new obstacles in the environment but is unstable. For better comprehensive performance, we combine them in a way that MPC can switch between them. When the reference path is not blocked, MPC uses the original reference trajectory $\mathcal{T}^{\text{ref}}$ until any positions of $\mathcal{T}^{\text{ref}}$ enter an occupied area and MPC switches to the new reference $\mathcal{T}'$. When MPC uses $\mathcal{T}'$, the condition to detach and reuse $\mathcal{T}^{\text{ref}}$ is that no position of $\mathcal{T}^{\text{ref}}$ is in occupied areas. In reality, the switch condition can be easily detected by the vision system. If only the laser/lidar scanner is available, this needs to be done by analyzing beams pointing to the reference path. In this work, we assume the vision system is provided to aid.

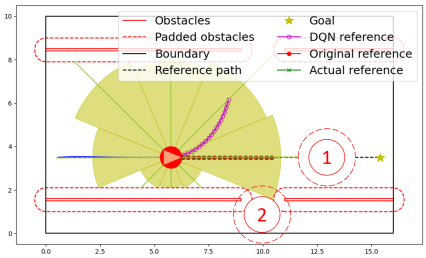

Fig. 4. Scene 1 with a dynamic obstacle: (1) coming to the agent face-to-face (from the right side to the left side, which is the opposite of the agent); (2) crossing the lane from one side (from the bottom to the top).

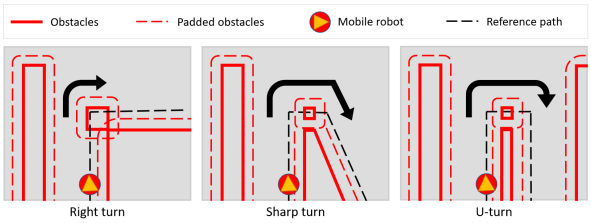

Fig. 5. Turning cases of Scene 2. The black arrows indicate desired turning directions. A small obstacle is placed at the turning corner in each case.

VI. Implementation

All experiments are conducted on Intel i7-9750H, while the neural network inferences are on NVIDIA GTX 1650 Max-Q. The code of this work is available online [1]. The MPC part is implemented by the real-time OpEn optimization engine [22]. The obstacle avoidance is solved by the penalty methods [10]. The DQN training is based on the Stable-Baseline3 library [23] with the default DQN architectures. For the laser/lidar scanner DQN, since the input state $\boldsymbol{s} = [(\boldsymbol{s}^{\text{int}})^T, (\boldsymbol{s}^L)^T]^T$ is a vector, the architecture is a two-layers fully-connected net with 16 neurons in each layer. For the vision DQN, the input image (cropped from 18×18 unit area, compressed to 54×54 pixels) is converted into a latent vector of size 256 by a convolutional neural network with the architecture described in [24]; the latent vector is concatenated with $\boldsymbol{s}^{\text{int}}$ and the entire vector is fed into a two-layers fully-connected net with 64 neurons in each layer.

To train the reinforcement learning agent, two kinds of training scenes are designed to increase the generality of the agent, which are fixed scenes and random scenes. Fixed scenes are manually created with static and dynamic obstacles, which are available in our provided code. The random scenes contain three static obstacles with reasonable random varying sizes at random locations and a random number of dynamic obstacles moving back and forth on randomly generated straight paths. In each random scene, the agent is given a random start point from one side of the map to the other side at a random goal point. The reference path in each random scene is generated through the visibility graph and A* algorithm as in [25]. For the hyperparameters used during the training, the discount rate is $\gamma = 0.98$, the learning rate $\alpha = 0.0001$, the exploration fraction which means the probability of taking a random action instead of the action suggested by the policy during training is 0.2, the number

[1] https://github.com/Woodenonez/TrajTrack_MPCnDQN_RLBoost

of gradient steps is the same as the number of steps done in the environment during the rollout, and the others are default values in the Stable-Baseline3 library [23].

VII. Use Cases and Evaluation

To evaluate the proposed approach, multiple use cases are designed. Some of the videos are available online [2], and the others are available in the provided code. Generally, two scenes are presented. The first scene is a straight path in a vehicle lane with different static obstacles, as in Fig. 3, or dynamic obstacles, as in Fig. 4. The other scene is a group of different turning cases with an additional static obstacle at the turning corner. Three turning scenarios, as shown in Fig. 5, are evaluated: right turn, sharp turn, and U-turn.

A. Use Cases

In the first scene, four kinds of obstacles are tested. The first three are static obstacles corresponding to Fig. 3. The last one is a dynamic obstacle as illustrated in Fig. 4.

1) Single rectangular obstacle of (a) medium size (comparable size to the agent) and (b) large size (about four times larger than the agent) respectively.
2) Two rectangular obstacles in stagger position with the first one being (c) small (slightly blocking the path) and (d) large (extensively blocking the path) respectively.
3) Single non-convex (U-shape) obstacle of (e) small size (shallow) and (f) big size (deep) respectively.
4) Single dynamic obstacle, as shown in Fig. 4, coming from the front and side directions respectively.

In the second scene, three types of turns are considered as in Fig. 5: (1) orthogonal right turn; (2) sharp turn; (3) U-turn.

B. Evaluation

To evaluate use cases, a comprehensive analysis is made to compare different approaches. Six metrics are included:

- Computation time. The computation time at each step reflects the real-time property. The mean, maximum, and medium computation time in a run is recorded. The computation time should be as small as possible.
- Deviation (distance to the reference path). The deviation from the reference path should be as small as possible if the path is free. When there are obstacles blocking the path, it is difficult to tell if this metric should be large or not since it depends on the application and policy.
- Action smoothness. The smoothness calculates the second derivative of the action. The mean value is taken for a run. In general, we hope the action is as smooth as possible, which means small values in this metric.
- Clearance (closest distance to obstacles). The agent should keep a reasonable distance from the obstacle, depending on the application and policy.
- Finish time. The agent should finish the journey as quickly as possible while not violating other conditions.
- Success rate (of 50 runs). The success rate should be as high as possible. In the evaluation, we do not show results if the success rate is below 30%.

[2] https://www.shorturl.at/sHOV7

The evaluations of Scene 1 and 2 are shown in TABLE I, where L means the laser/lidar version of the DQN and V means the vision version of the DQN. *HYB* means the hybrid of the MPC and the DQN. We can observe that the *HYB-L* always gives a much lower computation time, especially for the worst-case computation time. Except for the right turn case in Scene 2, *HYB-L* always has a 100% success rate even in the cases with non-convex obstacles, where the pure MPC solver is stuck at the local minimum. *HYB-V* is less stable, which we believe is due to the limited resolution of the image observation and the unstable training process of the convolutional neural network. Compared to the pure DQN versions, the hybrid versions give much better performance in terms of action smoothness and success rate. One advantage of *HYB-V* is reflected in the right turn case, where *HYB-V* has a better success rate compared to *HYB-L*. This is because laser/lidar scanners cannot provide environmental information after the turn, while visual detection can.

VIII. Conclusion

In this study, we present a novel approach that combines deep reinforcement learning with model predictive control for trajectory tracking in mobile robots. We train two distinct deep Q-network variants to accommodate either laser/lidar observations or visual observations. A series of use cases are designed to evaluate the performance of different methodologies across various dimensions. The experimental results show that the integration of DQN and MPC outperforms both the standalone MPC and standalone DQN strategies. We conclude that this research offers valuable insights into the synergy between machine learning techniques and traditional control methods. Future work will focus on improving the stability of the vision-based DQN and implementing the proposed approach in real-world systems.

References

[1] M. De Ryck, M. Versteyhe, and F. Debrouwere, "Automated guided vehicle systems, state-of-the-art control algorithms and techniques," *Journal of Manufacturing Systems*, vol. 54, pp. 152–173, 2020.

[2] G. Fragapane, R. de Koster, F. Sgarbossa, and J. O. Strandhagen, "Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda," *European Journal of Operational Research*, vol. 294, no. 2, pp. 405–426, 2021.

[3] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots (2nd Edition)*. MIT Press, 2011.

[4] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control (First Edition)*. Cambridge University Press, 2017.

[5] S. F. Roselli, P.-L. Götvall, M. Fabian, and K. Åkesson, "A compositional algorithm for the conflict-free electric vehicle routing problem," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 1405–1421, 2022.

[6] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, "Model predictive contouring control for collision avoidance in unstructured dynamic environments," *RA-L*, vol. 4, no. 4, pp. 4459–4466, 2019.

[7] 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 *ECC*. IEEE, 2018, pp. 1523–1528.

[8] B. Lindqvist, S. S. Mansouri, A.-a. Agha-mohammadi, and G. Nikolakopoulos, "Nonlinear MPC for collision avoidance and control of UAVs with dynamic obstacles," *RA-L*, vol. 5, no. 4, pp. 6001–6008, 2020.

[9] Z. Zhang, E. Dean, Y. Karayiannidis, and K. Åkesson, "Motion prediction based on multiple futures for dynamic obstacle avoidance of mobile robots," in *CASE*. IEEE, 2021, pp. 475–481.

TABLE I

EVALUATION RESULTS ON SCENE 1 (AVERAGE OVER 50 RUNS).

Scene	Obstacle type	Method	Computation time (ms/step)			Deviation (m)		Action smoothness		Other		
			mean	max	median	mean	max	speed	angular speed	clearance (m)	finish time step	success rate (%)
Scene 1	Rectangular obstacle (a-medium)	MPC	38.51	181.73	22.62	0.42	1.61	0.03	0.03	0.76	70	100
		DQN-L	0.35	0.61	0.34	0.88	1.61	0.23	0.35	0.71	84	84
		HYB-L	23.27	79.13	16.27	0.80	1.97	0.02	0.06	0.93	77	100
		HYB-V	33.95	160.51	23.21	0.80	1.72	0.03	0.10	0.81	78	100
	Rectangular obstacle (b-large)	MPC	41.9	151.71	27.71	0.54	1.92	0.04	0.03	0.68	76	100
		DQN-L	0.40	0.68	0.39	1.01	1.82	0.24	0.35	0.62	82	100
		HYB-L	29.27	80.18	19.34	0.89	2.04	0.03	0.06	0.80	77	100
		HYB-V	40.52	262.91	31.02	0.92	2.10	0.04	0.10	0.78	81	100
	Two obstacles (c-small stagger)	MPC	48.54	134.25	44.75	0.44	1.31	0.05	0.03	0.67	71	100
		HYB-L	41.45	122.80	32.65	0.86	1.97	0.03	0.04	0.70	100	100
		HYB-V	40.68	152.05	32.21	0.64	1.45	0.03	0.07	0.73	85	100
	Two obstacles (d-large stagger)	**HYB-L**	42.2	132.44	37.81	1.02	2.59	0.04	0.06	0.70	109	100
		HYB-V	53.59	247.83	45.85	1.01	2.66	0.04	0.06	0.70	98	100
	U-shape obstacle (e-small)	DQN-L	0.36	0.68	0.36	0.99	1.84	0.24	0.36	0.56	87	94
		HYB-L	27.31	189.15	18.25	0.89	2.03	0.03	0.06	0.84	78	100
		HYB-V	87.99	961.82	38.75	0.90	2.78	0.08	0.12	0.69	128	60
	U-shape obstacle (f-large)	**HYB-L**	25.69	131.83	17.63	1.17	2.94	0.02	0.04	0.81	82	100
		HYB-V	256.16	>1000	59.90	0.6	1.52	0.15	0.21	0.63	85	38
	Dynamic obstacle (face-to-face)	MPC	33.63	262.75	13.65	0.44	1.68	0.03	0.03	1.28	70	100
		DQN-L	0.39	0.69	0.38	0.71	1.43	0.24	0.39	1.71	78	78
		HYB-L	20.12	97.51	15.62	0.43	1.67	0.04	0.04	1.29	71	100
Scene 2	Right turn with an obstacle	**HYB-L**	34.19	195.17	21.94	0.32	1.27	0.03	0.04	0.67	133	64
		HYB-V	28.15	254.82	15.12	0.53	1.86	0.02	0.05	0.8	132	100
	Sharp turn with an obstacle	MPC	28.94	183.12	15.7	0.15	0.61	0.04	0.05	0.53	146	30
		DQN-L	0.34	0.67	0.34	0.60	1.02	0.23	0.33	0.58	161	46
		DQN-V	0.88	6.45	0.79	0.86	1.43	0.19	0.69	0.58	150	78
		HYB-L	22.51	99.35	13.55	0.37	1.42	0.01	0.02	0.87	152	100
		HYB-V	23.34	137.93	14.00	0.46	1.76	0.02	0.03	0.84	150	100
	U-turn with an obstacle	DQN-L	0.42	0.66	0.42	0.44	0.85	0.18	0.29	0.52	163	34
		DQN-V	1.17	6.56	0.96	0.51	1.07	0.17	0.67	0.55	148	58
		HYB-L	26.55	137.20	18.06	0.39	1.38	0.02	0.02	0.76	148	100
		HYB-V	32.80	197.30	16.24	0.43	1.67	0.02	0.03	0.69	149	100

[10] B. Hermans, P. Patrinos, and G. Pipeleers, "A penalty method based approach for autonomous navigation using nonlinear model predictive control," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 234–240, 2018.

[11] A. Jain, L. Chan, D. S. Brown, and A. D. Dragan, "Optimal cost design for model predictive control," in *Conference on Learning for Dynamics and Control*, vol. 144. PMLR, 2021, pp. 1205–1217.

[12] K. Zhu and T. Zhang, "Deep reinforcement learning based mobile robot navigation: A review," *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021.

[13] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv:1606.01540*, 2016.

[14] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *IROS*. IEEE/RSJ, 2017, pp. 31–36.

[15] Y. Kato, K. Kamiyama, and K. Morioka, "Autonomous robot navigation system with learning based on deep Q-network and topological maps," in *SII*. IEEE/SICE, 2017, pp. 1040–1046.

[16] R. Chai, H. Niu, J. Carrasco, F. Arvin, H. Yin, and B. Lennox, "Design and experimental validation of deep reinforcement learning-based fast trajectory planning and control for mobile robot in unknown environment," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2022.

[17] E. Marchesini and A. Farinelli, "Discrete deep reinforcement learning for mapless navigation," in *ICRA*. IEEE, 2020, pp. 10 688–10 694.

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

[19] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[20] 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, pp. 529–533, 2015.

[21] P. Regier, L. Gesing, and M. Bennewitz, "Deep reinforcement learning for navigation in cluttered environments," in *CMLA*, 2020.

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

[23] 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.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[25] J. Berlin, G. Hess, A. Karlsson, W. Ljungbergh, Z. Zhang, K. Åkesson, and P.-L. Götvall, "Trajectory generation for mobile robots in a dynamic environment using nonlinear model predictive control," in *CASE*. IEEE, 2021, pp. 942–947.