

PathRL: An End-to-End Path Generation Method for Collision Avoidance via Deep Reinforcement Learning

Wenhao Yu¹, Jie Peng², Quecheng Qiu³, Hanyu Wang², Lu Zhang⁴ and Jianmin Ji^{3,*}

Abstract—Robot navigation using deep reinforcement learning (DRL) has shown great potential in improving the performance of mobile robots. Nevertheless, most existing DRL-based navigation methods primarily focus on training a policy that directly commands the robot with low-level controls, like linear and angular velocities, which leads to unstable speeds and unsmooth trajectories of the robot during the long-term execution. An alternative method is to train a DRL policy that outputs the navigation path directly. Then the robot can follow the generated path smoothly using sophisticated velocity-planning and path-following controllers, whose parameters are specified according to the hardware platform. However, two roadblocks arise for training a DRL policy that outputs paths: (1) The action space for potential paths often involves higher dimensions comparing to low-level commands, which increases the difficulties of training; (2) It takes multiple time steps to track a path instead of a single time step, which requires the path to predicate the interactions of the robot w.r.t. the dynamic environment in multiple time steps. This, in turn, amplifies the challenges associated with training. In response to these challenges, we propose *PathRL*, a novel DRL method that trains the policy to generate the navigation path for the robot. Specifically, we employ specific action space discretization techniques and tailored state space representation methods to address the associated challenges. Curriculum learning is employed to expedite the training process, while the reward function also takes into account the smooth transition between adjacent paths. In our experiments, *PathRL* achieves better success rates and reduces angular rotation variability compared to other DRL navigation methods, facilitating stable and smooth robot movement. We demonstrate the competitive edge of *PathRL* in both real-world scenarios and multiple challenging simulation environments.

I. INTRODUCTION

DRL navigation methods have shown great potential in improving the flexibility and adaptability of mobile robots in complex and changing scenarios [1]. Most existing DRL navigation methods intend to train a policy that directly commands the robot with single-step low-level controls, like linear and angular velocities. However, it is challenging for most DRL algorithms to memorize long histories of low-level controls, which may lead to unstable speeds and unsmooth trajectories of the robot during the long-term execution and cannot achieve typical driving maneuvers [2].

¹ Institute of Advanced Technology, University of Science and Technology of China (USTC), Hefei 230026, China wenhaoyu@mail.ustc.edu.cn

² School of Data Science, USTC, Hefei 230026, China {pengjie, wangyuu}@mail.ustc.edu.cn

³ School of Computer Science and Technology, USTC, Hefei 230026, China qiuqc@mail.ustc.edu.cn

⁴ Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, 230088, China luzha@ustc.edu.cn

* Corresponding author. jianmin@ustc.edu.cn

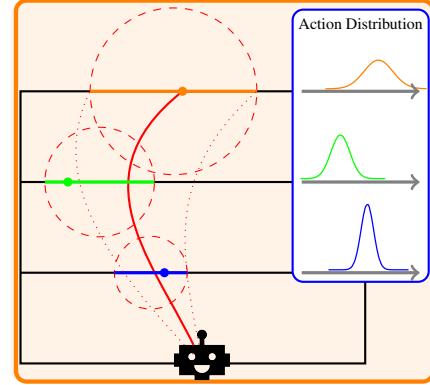


Fig. 1. Illustration of the path generation. The area surrounded by two red dotted lines denotes the potential path area. The control points in different colors are sampled by the DRL policy, and the final sampling path (red solid line) is formed through path continuation (Bézier curve).

It is also hard to train a DRL policy converging to the almost zero exploration from the balance of *exploration and exploitation* [3], which makes the outputs of the policy easy to drift. On the other hand, a human-friendly mobile robot is expected to drive stably and smoothly.

The decoupling of the control module from the planning module enables the robot to *follow fluctuating path outputs using stable control actions*. In this paper, we propose *PathRL*, a DRL-based navigation method that outputs navigation paths, which can be followed smoothly by the robot using sophisticated path-tracking algorithms [4]–[6]. This form of DRL network makes the robot action predictable due to the low-level controls that are got by the control module, which is predictable in multiple time steps. Moreover, we also benefit from the flexibility and adaptability of the DRL policy. The behavior policy limits the path generated by the imitation learning-based method, and our proposed *PathRL* as a policy-based DRL algorithm employs more flexibility. In *PathRL*, the trained policy focuses on generating proper paths for the robot in various dynamic environments, which is more robust for various robots with different motion models and can also benefit human-robot interaction [7], [8].

It is challenging to train a navigation policy that outputs paths as the paths often involve the higher-dimensional policy space. As shown in Fig. 1, in *PathRL*, we use Bézier curve to fit the navigation path via the current position of the robot and other three control points generated by the DRL policy. Notice that, each pair of adjacent control points has a fixed interval in the longitudinal direction. Then these three control points can be specified by the DRL policy via three continuous numbers for corresponding values in the horizontal direction. To facilitate subsequent citation and

description, we name the above method of setting control point coordinates as semi-discrete operations. In this way, the DRL policy can specify a path via these three continuous numbers, which is slightly more complex than the DRL method in [9] where the policy outputs two continuous numbers corresponding to linear and angular velocities, respectively. In *PathRL*, we employ three consecutive path-based frames of sensor information and relative targets as inputs of the navigation policy, as shown in Fig. 3. The state of each frame is extracted from the state space upon the termination of semi-MDP options [10] (once a path has been completely followed). We also apply curriculum learning [11] to speed up the training process.

In our experiments, we employ the soft actor-critic (SAC) algorithm [12] as the DRL method for the training. During the training, the robot is driven by a path-following algorithm that follows the paths generated by the trained policy while given a fixed linear velocity. The robot will follow the new path only after it has completed the following of the previous one, which follows a typical semi-MDP process [10] with temporal abstraction [13], [14]. During the execution, this path-following process can be interrupted when the robot is approaching obstacles to achieve more safety navigation. Our experiments in both real-world scenarios and multiple challenging simulation environments show that *PathRL* enables an explicable and predictable DRL navigation policy while maintaining the strategic diversity of DRL agents.

The main contributions of our work are summarized in the following:

- We propose a novel end-to-end DRL-based method, *PathRL*, that directly outputs navigation paths without relying on the supervised learning paradigm and is competent for a variety of complex scenarios.
- We implement semi-discrete operations on the action space to reduce the difficulty of policy training. We utilize three consecutive path-based frames as network inputs to enhance the performance of navigation strategies in dynamic and complex environments. We design a reward function to ensure that the output path is smoother and satisfies the smooth transition between adjacent paths.
- Extensive experiments demonstrate the promise of the *PathRL* in achieving superior yet smoother driving trajectories.

II. RELATED WORK

Classical navigation methods, like Time Elastic Band (TEB) algorithm [15], Optimal Reciprocal Collision Avoidance (ORCA) [16] and Social Force Model (SFM) [17], may take a lot of hard work to find proper parameters in various scenarios. Supervised learning based navigation methods, like [18] and [19], rely heavily on a large amount of labeled data, while the quality of which directly affects the performance of the trained policy.

Recently, there have been many DRL-based works on robot motion planning and collision avoidance. [20] used the asynchronous deep deterministic policy gradients (ADDPG)

algorithm with low-dimensional LiDAR data (10 returns) as input to create a navigation system with good obstacle avoidance capabilities. In the field of social navigation, [21] provided a crowd navigation method where both the pedestrian map and the sensor map were used to allow pedestrians to follow different collision avoidance strategies. In these works, the DRL network directly replaces the entire navigation stack and outputs end-to-end navigation instructions. However, these end-to-end methods try to predict low-level control commands for the robot, which may lead to unstable actions and unsmooth driving trajectories.

APF-RL [22] combined the strengths of Artificial Potential Functions (APF) with DRL, employing the SAC algorithm to dynamically adjust the two input parameters of the APF controller: the intermediate goal and the k-parameter. PRM-RL [23] used PRM as a global planner and DRL as a local planner to complete long-range navigation tasks. The majority of these methods incorporate elements of traditional approaches, with some serving as learning subsystems within classic architectures, while others focus on training individual components of traditional methods. The emphasis here lies in highlighting the distinction of our approach from previous methods. We aim to leverage the flexibility and adaptability inherent in DRL to train an end-to-end policy network capable of directly generating paths.

The following works are more similar to our method in the idea of path generation. [2] leveraged expert prior knowledge to learn high-level motion skills instead of low-level control skills. In contrast, our method focuses on searching feasible paths directly from the policy space in an end-to-end manner without any expert prior knowledge. [24] addressed the problem of predicting proper paths for robots. In this work, the navigation problem is modeled as a deep Markov model, the driving distance and rotation angle of the robot are output at each step, and the final navigation path is formed by combining the multi-step outputs of the algorithm. The method in [24] can achieve good performance in static scenarios, however, it does not perform well in dynamic scenarios.

III. METHOD

A. Reinforcement Learning Components

In DRL, we formulate robot’s navigation as a Markov Decision Process (MDP) problem. Specifically, an MDP is a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$, where \mathcal{S} is the state space, \mathcal{A} is the action space, \mathcal{P} presents the transition probability between states, and γ is the discount factor in $(0, 1)$.

1) *State Space*: In our formulation, a state $s_t = (m_t, g_t)$ at time step t consists of two parts: an egocentric costmap m_t and a relative target pose g_t .

The egocentric costmap¹ is a 84×84 grid map that is generated by a 3D laser sensor with 180° field of view and presents information about the environment around the robot, including the shape of the robot and the observable appearances of various obstacles.

¹http://wiki.ros.org/costmap_2d

The relative target pose $g_t = (x_t^g, y_t^g, \alpha_t^g)$ of the robot includes the relative position of the navigation target point (x_t^g, y_t^g) and the relative target orientation α_t^g of the robot w.r.t. the current global pose (x_t, y_t, α_t) of the robot at time t , i.e., this relative target pose g_t is the local pose for the local coordinate system when the pose of the robot is identified as the origin.

In pathRL, the input of the DRL network is three consecutive path-based frames. These three frames contain the sensor information and relative target poses of the robot at the poses of $(x_t^{-2}, y_t^{-2}, \alpha_t^{-2})$, $(x_t^{-1}, y_t^{-1}, \alpha_t^{-1})$, and (x_t, y_t, α_t) , respectively. Notice that, we define the two nearest consecutive paths that the robot must follow to reach its current pose (x_t, y_t, α_t) as P_t^{-1} and P_t^{-2} , respectively. We use $(x_t^{-1}, y_t^{-1}, \alpha_t^{-1})$ and $(x_t^{-2}, y_t^{-2}, \alpha_t^{-2})$ to denote the starting poses of the robot for these two paths, respectively.

As shown in Fig. 2, we can specify the input of the DRL policy, i.e., three consecutive path-based frames, as $\vec{s}_t = (s_{t-(\tau_1+\tau_2)}, s_{t-\tau_1}, s_t)$, where $s_{t-(\tau_1+\tau_2)} = (m_t^{-2}, g_t^{-2})$ denotes an egocentric costmap and a relative target pose w.r.t. the pose $(x_t^{-2}, y_t^{-2}, \alpha_t^{-2})$ of the robot, $s_{t-\tau_1} = (m_t^{-1}, g_t^{-1})$ w.r.t. the pose $(x_t^{-1}, y_t^{-1}, \alpha_t^{-1})$, and $s_t = (m_t, g_t)$ w.r.t. the pose (x_t, y_t, α_t) . We use τ_1 to denote the amount of the time steps for following the path P_t^{-1} and τ_2 to denote the amount of the time steps for following the path P_t^{-2} .

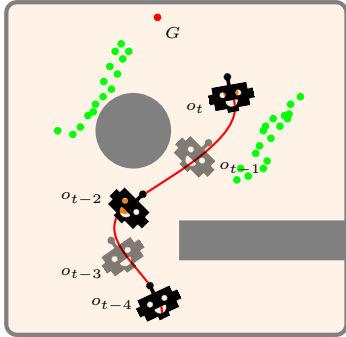


Fig. 2. Consecutive path-based frames for the DRL-based navigation policy. The gray circular and rectangular block denote static obstacles, green dots denote the trajectories of pedestrians, and G denotes the target point. In the current scenario, the three consecutive path-based frames are expressed as $\vec{s}_t = (s_{t-4}, s_{t-2}, s_t)$, as the robot takes 2 time steps to follow each of the two previous paths, and sensor-based frames are expressed as $\vec{s}_t = (s_{t-2}, s_{t-1}, s_t)$.

2) *Action Space*: In this paper, we intend to train a DRL policy that outputs navigation paths directly. In our formulation, we use Bézier curve to fit the navigation path via the current position of the robot and other n control points generated by the DRL policy. As shown in Fig. 1, the action space is the possible control points that can be selected in the rectangular area in front of the robot, which is high-dimensional. Therefore, to reduce the dimension of the action space and the training difficulty, we discretize the selection of longitudinal coordinate values for these n control points.

In specific, the n control points are identified as (l_t^1, h_t^1) , (l_t^2, h_t^2) , \dots , (l_t^n, h_t^n) , where the point (l, h) denotes the local longitudinal coordinate l and the local horizontal coordinate h when the current pose (x_t, y_t, α_t) of the robot is considered

as the origin of the local coordinates. We assign a fixed interval between two adjacent longitudinal coordinates, i.e., $l_t^{i+1} - l_t^i = l_t^1 = d$ for $1 \leq i < n$. In our experiments, we set $n = 3$ and $d = 0.4/3m$, then l_t^1 , l_t^2 , and l_t^3 are assigned to be $0.4/3m$, $0.8/3m$, and $0.4m$, respectively.

Notice that, the previous path P_t^{-1} is specified by the starting pose $(x_{t-\tau}, y_{t-\tau}, \alpha_{t-\tau})$ of the robot and n local control points $(l_{t-\tau}^1, h_{t-\tau}^1)$, \dots , $(l_{t-\tau}^n, h_{t-\tau}^n)$, where τ is the amount of the time steps for following the path P_t^{-1} . Particularly, the current position (x_t, y_t) of the robot at time t is the ending position of the robot for the previous path P_t^{-1} , then (x_t, y_t) is equivalent to the global position for the last control point $(l_{t-\tau}^n, h_{t-\tau}^n)$ of P_t^{-1} , i.e.,

$$\begin{aligned} x_t &= l_{t-\tau}^n \cos(\alpha_{t-\tau}) - h_{t-\tau}^n \sin(\alpha_{t-\tau}) + x_{t-\tau}, \\ y_t &= l_{t-\tau}^n \sin(\alpha_{t-\tau}) + h_{t-\tau}^n \cos(\alpha_{t-\tau}) + y_{t-\tau}. \end{aligned}$$

3) *Reward Function*: The DRL policy aims to maximize the cumulative reward. In robot navigation tasks, the primary goal of the robot is to reach the target without collision within a limited time. Based on this goal, the robot's trajectory is further required to be smoother. Therefore, the reward function of the algorithm is defined as follows:

$$r_t = r_t^{goal} + r_t^{safe} + r_t^{curvature} + r_t^{straight},$$

where r_t is the sum of four parts, r_t^{goal} , r_t^{safe} , $r_t^{curvature}$ and $r_t^{straight}$.

In particular, r_t^{goal} is the reward when the robot reaches the local target:

$$r_t^{goal} = \begin{cases} r_{arr}, & \text{if target is reached,} \\ 0, & \text{otherwise.} \end{cases}$$

r_t^{safe} specifies the penalty when the robot encounters a collision:

$$r_t^{safe} = \begin{cases} r_{col}, & \text{if collision,} \\ 0, & \text{otherwise.} \end{cases}$$

We use the $r_t^{curvature}$ to encourage the DRL algorithm to output paths with less curvature:

$$r_t^{curvature} = -\varepsilon_1 \left(\sum_{i=0}^N k(x) \right), x = \frac{i}{N} \in [0, 1],$$

where N is the total number of points after the discretization of the path, ε_1 is a hyper-parameter and $k(x)$ is the curvature of the path at the i th point. The following formula defines $k(x)$:

$$k(x) = \frac{\|B'(x) \times B''(x)\|}{\|B'(x)\|^3},$$

where $B(x)$ means Quadratic Bézier curve function. The intuition behind the definition of $r_t^{curvature}$ is that the larger the curvature of points along the path, the more unstable the robot's motion becomes, and thus the penalty should be greater.

The purpose of $r_t^{straight}$ is to encourage the DRL algorithm to generate paths that maximize alignment with the direction of the vehicle body. This strategic approach ensures the

smooth transition between adjacent paths, thereby enhancing the smoothness of the robot’s movement trajectory.

$$r_t^{straight} = \varepsilon_2(L_{min} - L),$$

where ε_2 is a hyper-parameter, L_{min} is the length of the shortest path, which is related to the action space dimension and the distance between the path points, and L is the length of the path output by the DRL algorithm. The intuition behind the definition of $r_t^{straight}$ is to avoid sharp angles between adjacent paths. In our experiments, we set $r_{arr} = 500$, $r_{col} = -700$, $N = 100$, $\varepsilon_1 = 0.4$ and $\varepsilon_2 = 200$.

Algorithm 1: SAC for the generation of paths

- 1 Initialize the policy network π , the value network, and various hyperparameters;
 - 2 Clear the experience buffer Buffer \mathcal{D} ;
 - 3 for epoch = 1, ..., E do
 - 4 for step $t = 1, \dots, T_{ep}$ do
 - 5 $a_t = \pi(\vec{s}_t)$;
 - 6 $\vec{a}_t = \text{Bézier}(a_t)$;
 - 7 if A path has been followed in its entirety
 - 8 then
 - 9 $a = \vec{a}_t$;
 - 10 $s_{t+1}, r_t = \text{path_follow}(a)$;
 - 11 if A path has been followed in its entirety
 - 12 then
 - 13 dequeue \vec{s}_{t+1} append s_{t+1} ;
 - 14 $\mathcal{D} \leftarrow \mathcal{D} \cup \{\vec{s}_t, a_t, r_t, \vec{s}_{t+1}\}$;
 - 15 Sample batch $\mathcal{B} \sim \mathcal{D}$ to update SAC network;
 - 16 if robot has stopped then
 - 17 $\vec{s}_t = \text{reset}()$;
-

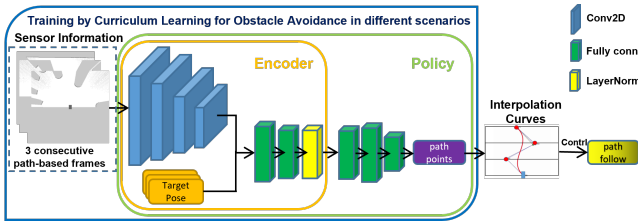


Fig. 3. An overview of the entire process and the structure of the policy network. The egocentric costmap is fed into a fully convolutional network for the feature extraction. The extracted features are then concatenated with the relative target pose of the robot and input into a fully connected network to output the final coordinates of the path points.

B. Training by Curriculum Learning for Obstacle Avoidance

It remains challenging to achieve an optimal policy through the above method stably and efficiently. Therefore, we employ curriculum learning [11] to assist and speed up the training process. Overall, we design a two-stage training process for curriculum learning in our experiments. In the first stage, we perform the DRL agent in two kinds of environments for the training, both of which were conducted in a static scenario with gradually increased complexity (4 and 16 static obstacles, respectively). At this stage, we intent

to enable the DRL agent to learn basic navigation abilities and obstacle avoidance capabilities. In the second stage, we continuously train the DRL agent obtained from the first stage in environments with much more complex dynamic scenarios, which enable the DRL agent to learn a more robust navigation strategy to interact with dynamic obstacles.

IV. EXPERIMENTS

A. Experiments on simulation scenarios

Here we consider three different types of scenarios, i.e., static, pedestrian, and multi-agent scenarios, to generate random environments in our customized simulator [9] to collect experiences for the training of the navigation policy. Fig. 4 illustrates environments for these three scenarios, respectively.

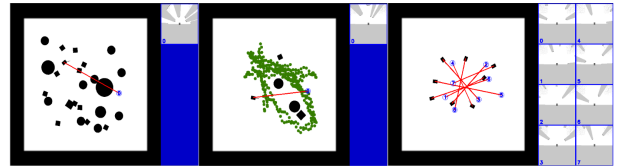


Fig. 4. Illustration of the three scenarios for the training. The blue digital circles indicate the target position of the corresponding robots, red lines represent the straight paths from the starting point to the target point for robots, black circle and rectangle blocks specify static obstacles, green dots denote pedestrian trajectories, and blue boxes on the right show sensor maps of each robot.

We compare the performance of *PathRL* with three other DRL-based navigation methods, i.e., PPO [9], SAC [12] and PSD [21]. All methods are evaluate in both static and dynamic simulation environments using an Ackermann steering robot² whose length by width is 0.3×0.2 . The action space for both PPO and SAC methods is the linear velocity and the angle of the front wheel, i.e., $v \in [0, 0.6]$ and $\theta \in [-0.785, 0.785]$. Notice that, both PPO and SAC methods share the similar network structure as *PathRL* as shown in Fig. 3. PSD is a crowd navigation method that enables efficient and smooth motion planning within highly crowded and dynamic environments. It’s important to mention that we have modified the network output of PSD to support Ackermann steering robots. We have also well justified the training details for all methods to achieve good performance as shown in our previous work [9], [25]. In this paper, we show that DRL policies as trained by *PathRL* that directly output paths can greatly improve the speed stability and trajectory smoothness of the robot.

We introduce four metrics to evaluate the performance of navigation policies trained by different DRL methods in multiple scenarios as the following:

- *Success rate (SUCC)*: the ratio of episodes in which the robot reaches the target pose without collision.
- *Average curvature (CUR)*: the average curvature of the robot’s driving path at each episode, which evaluates

²Note that, it is more challenging for DRL methods to train a navigation policy for a robot with additional kinematic constraints like an Ackermann steering robot comparing with a differential robot. We use Bézier curves to guarantee kinematic feasibility. Dynamic constraints (acceleration, curvature) are satisfied by forcibly limiting planning parameters within reasonable ranges.

the stability of the (angular) speeds and the smoothness of the trajectories for the robot.

- *Average length (LEN)*: the average length of the robot’s driving path at each success episode.
- *Average time (TIME)*: the average cost time at each success episode.

In the following, all experimental results for different methods were obtained from the average results for 1000 randomly generated environments of each scenario. Pedestrians in the scenario are driven by OCRA and SFM and multi-agents adopt the self-play strategy without communicating with each other as shown in [9]. For a fair comparison, we trained 3 versions of the models for each DRL method and summary the average performance of each method for multiple scenarios in following tables.

TABLE I
PERFORMANCE OF DIFFERENT NAVIGATION METHODS

Scenarios	Methods	SUCC(%) \uparrow	CUR \downarrow	LEN(m) \downarrow	TIME(s) \downarrow
Static Scenario (16 Obstacles)	PPO	0.979	1.237	5.209	9.177
	SAC	0.972	1.418	5.486	9.474
	PSD	0.975	1.219	5.136	9.164
	PathRL	0.985	0.691	4.804	7.522
Static Scenario (24 Obstacles)	PPO	0.960	1.454	5.292	9.113
	SAC	0.941	1.571	5.761	10.004
	PSD	0.958	1.405	5.307	9.154
	PathRL	0.968	0.772	5.281	8.249
Dynamic Scenario (4 Obstacles & 4 Pedestrians)	PPO	0.850	2.519	6.921	11.850
	SAC	0.889	1.538	6.858	11.533
	PSD	0.911	1.584	7.483	12.988
	PathRL	0.889	0.904	8.431	12.632
Dynamic Scenario (6 Pedestrians)	PPO	0.769	2.027	7.587	14.103
	SAC	0.894	1.462	8.452	14.224
	PSD	0.926	1.387	9.094	15.867
	PathRL	0.875	0.836	14.41	20.268
Dynamic Scenario (8 Agents)	PPO	0.879	2.197	6.171	11.832
	SAC	0.947	3.869	7.004	12.702
	PSD	0.909	1.928	7.409	12.364
	PathRL	0.998	0.705	7.567	11.642

1) *Comparative experiments*: Table I shows that *PathRL* outperforms the other DRL methods by a considerable improvement of “Average curvature (CUR)” without sacrificing “Success rate (SUCC)”. In dynamic environments, we find out that both “Average length (LEN)” and “Average time (TIME)” of *PathRL* are slightly larger than that of other DRL methods, as *PathRL* is required to drive the robot to avoid dynamic obstacles stably and smoothly, which is preferred for most real-world applications. Notice that, compared to *PathRL*, PSD has additional pedestrian map input, resulting in a higher success rate in pedestrian scenarios.

Notice that, additional smoothing requirements are preferred for an Ackermann steering robot. We use the average changes of the steering angle at each step, i.e. $\Delta\theta = \theta_{t+1} - \theta_t$, as a metric to assess the smoothness and comfort of the robot’s movement.

Table II summarizes the results of the $\Delta\theta$ metric for three reinforcement learning methods. Since *PathRL* directly outputs the path and decouples the motion control module from the DRL policy, it achieves a dramatic improvement of the stability of the steering angle for the robot comparing with other DRL methods.

In our setting, the DRL navigation policies are trained using the experiences collected in various simulation environments. Then it is crucial for these policies to be robust in the presence of the differences between the simulation model

TABLE II

AVERAGE CHANGES OF STEERING ANGLE FOR DIFFERENT METHODS

Scenarios	Methods	$\Delta\theta(rad)\downarrow$
Static Scenario (24 Obstacles)	PPO	0.5527
	SAC	0.5584
	PSD	0.5498
	PathRL	0.0093
Dynamic Scenario (4 Obstacles & 4 Pedestrians)	PPO	0.8983
	SAC	0.5047
	PSD	0.4763
	PathRL	0.0114
Dynamic Scenario (8 Agents)	PPO	1.2490
	SAC	1.1156
	PSD	1.0271
	PathRL	0.0023

and the real-world robot platform. Notice that, DRL-based navigation methods that directly commands the robot with low-level controls are sensitive to parameters of the robot platform, which limits the generality of methods. Meanwhile, *PathRL* directly outputs the path and decouples the motion control module from the DRL policy. Then, sophisticated velocity-planning and path-following controllers, whose parameters are specified according to the real-world robot platform, can be applied for the robot to follow the paths generated by the DRL policy. In the following, we consider the impact for the navigation performance for different DRL-based navigation methods when varying the maximum driving speed of the robot platform.

In our experiment, we respectively apply the four DRL methods to train corresponding navigation policies in the same simulation environments, where the maximum moving speed of the robot is 0.2 m/s. Then we test the performance of these navigation policies in both static and multi-agent scenarios when the maximum moving speed is increased to 0.4 m/s, 0.6 m/s, and 0.8 m/s, respectively. The experimental results are summarized in Figure 5, where “x-static” and “x-magent” denote the performance of the trained navigation policy using the DRL method “x” tested in static and multi-agent scenarios, respectively. Comparing with other DRL methods, the performance of *PathRL* is more robust when the maximum moving speed is increased, which maintains the stable success rate and average curvature of moving trajectories.

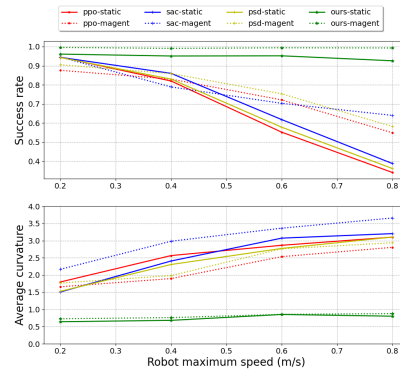


Fig. 5. The performance of DRL methods after increasing the maximum moving speed of the robot in different scenarios.

Additionally, we compare the performance of *PathRL* with TEB algorithm in 3D gazebo environments. Notice that, the DRL navigation policy is trained in previous training environments using *PathRL*, which is only tested in gazebo

environments here and We make every effort to fine-tune the parameters of the TEB algorithm to achieve improved performance. We test *PathRL* and TEB in environments of two scenarios, i.e., static scenario and dynamic scenario. As illustrated in Fig. 6, the testing environments of the static scenario is equipped with 8 possible routes (i.e., global paths) with different origins and destinations surrounding with static obstacles. The testing environments of the dynamic scenario contains 4 different routes surrounding with 25–35 pedestrians which are guided by the social forces model [17], [26], [27]. Table III summarizes the performance of *PathRL* and TEB in testing gazebo environments, where each method is tested in the same environment for five times. In addition, we use “NCOLL” to denote the average number of collisions during the execution of the policy or algorithm in testing environments.



Fig. 6. 3D Simulation test environments and test routes.

TABLE III
COMPARING *PathRL* AND TEB IN 3D GAZEBO ENVIRONMENTS

Scenarios	Method	SUCC(%) \uparrow	NCOLL \downarrow	TIME(s) \downarrow
Static Scenario	<i>PathRL</i>	0.900	3	65.326
	TEB	0.825	5	77.368
Dynamic Scenario	<i>PathRL</i>	0.850	5	76.331
	TEB	0.750	8	79.606

2) *Ablation study*: The input of DRL navigation position consists three consecutive state frames. In *PathRL*, there are two alternative considerations of “consecutive”, i.e., *i*) sensor-based three consecutive frames, where $\vec{s}_t = (s_{t-2}, s_{t-1}, s_t)$; *ii*) path-based three consecutive frames, where $\vec{s}_t = (s_{t-(\tau_1+\tau_2)}, s_{t-\tau_1}, s_t)$. Notice that, path-based consecutive frames have greater time intervals. The path-based frames introduces information for a longer span of time, which can benefit robot collision avoidance in dynamic scenarios and ensure the smooth transition between adjacent paths. To evaluate the advantage of path-based consecutive frames, we consider following ablation experiments. Specifically, we compare the performance of our design with sensor-based three consecutive frames in a dynamic scenario. As shown in Table IV, the design of path-based three consecutive frames achieves a higher success rate and smoother driving trajectory than the design of sensor-based three consecutive frames, with slight larger values of “LEN” and “TIME”.

TABLE IV
PERFORMANCE OF DIFFERENT STATE INPUT

Setting	SUCC(%) \uparrow	CUR \downarrow	LEN(m) \downarrow	TIME(s) \downarrow
path-based frames	0.889	0.993	8.758	12.632
sensor-based frames	0.837	1.469	7.905	11.590

Our experiments show that the $r_{straight}$ term in the reward function plays an important role in improving the

experimental results and guaranteeing the smooth transition. On the one hand, without the $r_{straight}$ constraint, the robot would twist during its movement, resulting in an uneven driving trajectory. On the other hand, the inclusion of the $r_{straight}$ term also results a significant improvement of the navigation performance. Table V summarizes the ablation study.

TABLE V
PERFORMANCE OF DIFFERENT REWARD FUNCTION SETTINGS

Scenarios	Rewards	SUCC(%) \uparrow	CUR \downarrow	LEN(m) \downarrow	TIME(s) \downarrow
Static Scenario (24 Obstacles)	<i>PathRL</i>	0.968	0.772	5.281	8.249
	<i>PathRL</i> w.o. $r_{straight}$	0.936	2.014	7.227	11.460
Dynamic Scenario (4 Obstacles & 4 Pedestrians)	<i>PathRL</i>	0.889	0.904	8.758	12.632
	<i>PathRL</i> w.o. $r_{straight}$	0.847	2.461	9.408	15.079

B. Deploy to real-world Ackermann steering robot

We deploy the trained DRL policy to a real-world Ackermann steering robot to test its performance in real-world static and dynamic environments for collision avoidance. We use Simultaneous Localization and Mapping (SLAM) [28] technology for mapping and localization.

As shown in Fig. 7, the platform for the Ackermann steering robot is based on Agilex hunter2.0 chassis and uses a 32-line 3D laser sensor. In addition, the robot is equipped with an RTX 3090 as a computing unit. The size of the robot, length \times width \times height, is $0.95m \times 0.75m \times 1.45m$. Our experiments show that the robot can successfully avoid static and dynamic obstacles and complete navigation tasks. More illustrations of the performance of the robot are shown in our demonstration video.

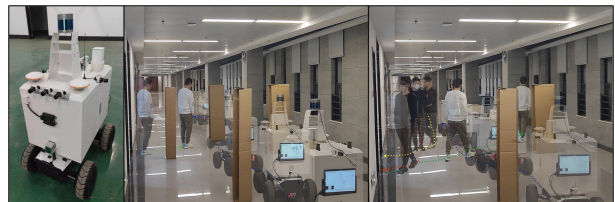


Fig. 7. The robot and trajectories of the robot in the test environments.

V. CONCLUSIONS

In this paper, we present an end-to-end path generation method without any expert prior knowledge, *PathRL*, for robot collision avoidance using DRL. We diminish the dimension of the policy space by decreasing the selection of longitudinal coordinate values for control points. By utilizing interpolation curves, we generate smoother paths. By employing three consecutive path-based frames and reward function constraints, we guarantee a smooth transition. Additionally, we utilize curriculum learning to expedite the training process and acquire a more robust navigation policy.

Comparing with DRL-based navigation methods that directly output low-level control commands, *PathRL* achieves fewer changes of driving trajectory curvature and less variation in angular rotation without sacrificing the success rate. Moreover, *PathRL* is more robust for the differences between the simulation model and the real-world robot platform, as it provides the decoupling between the planning and the control modules. In a variety of challenging scenarios in both simulation and the real world, *PathRL* achieves the nice navigation performance.

REFERENCES

- [1] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion planning and control for mobile robot navigation using machine learning: A survey," *Autonomous Robots*, vol. 46, no. 5, pp. 569–597, June 2022.
- [2] L. Wang, J. Liu, H. Shao, W. Wang, R. Chen, Y. Liu, and S. L. Waslander, "Efficient reinforcement learning for autonomous driving with parameterized skills and priors," *arXiv preprint arXiv:2305.04412*, 2023.
- [3] M. Coggan, "Exploration and exploitation in reinforcement learning," *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University*, 2004.
- [4] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.
- [5] T. Hellstrom and O. Ringdahl, "Follow the past: a path-tracking algorithm for autonomous vehicles," *International journal of vehicle autonomous systems*, vol. 4, no. 2-4, pp. 216–224, 2006.
- [6] K. C. Koh and H. S. Cho, "A smooth path tracking algorithm for wheeled mobile robots with dynamic constraints," *Journal of Intelligent and Robotic Systems*, vol. 24, pp. 367–385, 1999.
- [7] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey, *Human-computer interaction*. Addison-Wesley Longman Ltd., 1994.
- [8] I. S. MacKenzie, "Human-computer interaction: An empirical research perspective," 2012.
- [9] G. Chen, S. Yao, J. Ma, L. Pan, Y. Chen, P. Xu, J. Ji, and X. Chen, "Distributed non-communicating multi-robot collision avoidance via map-based deep reinforcement learning," *Sensors*, vol. 20, no. 17, p. 4836, 2020.
- [10] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.
- [11] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.
- [12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.
- [13] J. Merel, L. Hasenclever, A. Galashov, A. Ahuja, V. Pham, G. Wayne, Y. W. Teh, and N. Heess, "Neural probabilistic motor primitives for humanoid control," *arXiv preprint arXiv:1811.11711*, 2018.
- [14] J. Merel, S. Tunyasuvunakool, A. Ahuja, Y. Tassa, L. Hasenclever, V. Pham, T. Erez, G. Wayne, and N. Heess, "Catch & carry: reusable neural controllers for vision-guided whole-body tasks," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 39–1, 2020.
- [15] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *ROBOTIK 2012; 7th German Conference on Robotics*. VDE, 2012, pp. 1–6.
- [16] J. v. d. Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*. Springer, 2011, pp. 3–19.
- [17] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, p. 4282, 1995.
- [18] A. Amini, G. Rosman, S. Karaman, and D. Rus, "Variational end-to-end navigation and localization," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 8958–8964.
- [19] H. Ma, Y. Wang, L. Tang, S. Kodagoda, and R. Xiong, "Towards navigation without precise localization: Weakly supervised learning of goal-directed navigation cost map," June 2019.
- [20] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 31–36.
- [21] S. Yao, G. Chen, Q. Qiu, J. Ma, X. Chen, and J. Ji, "Crowd-aware robot navigation for pedestrians with multiple collision avoidance strategies via map-based deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 8144–8150.
- [22] K. Bektaş and H. I. Bozma, "Apf-rl: Safe mapless navigation in unknown environments," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 7299–7305.
- [23] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 5113–5120.
- [24] L. Zhang, Z. Hou, J. Wang, Z. Liu, and W. Li, "Robot navigation with reinforcement learned path generation and fine-tuned motion control," *arXiv preprint arXiv:2210.10639*, 2022.
- [25] Q. Qiu, S. Yao, J. Wang, J. Ma, G. Chen, and J. Ji, "Learning to socially navigate in pedestrian-rich environments with interaction capacity," *arXiv preprint arXiv:2203.16154*, 2022.
- [26] M. Moussaïd, D. Helbing, S. Garnier, A. Johansson, M. Combe, and G. Theraulaz, "Experimental study of the behavioural mechanisms underlying self-organization in human crowds," *Proceedings of the Royal Society B: Biological Sciences*, vol. 276, no. 1668, pp. 2755–2762, 2009.
- [27] Z. Xie and P. Dames, "Drl-vo: Learning to navigate through crowded dynamic scenes using velocity obstacles," *IEEE Transactions on Robotics*, pp. 1–20, 2023.
- [28] Y. Duan, J. Peng, Y. Zhang, J. Ji, and Y. Zhang, "Pfilter: Building persistent maps through feature filtering for fast and accurate lidar-based slam," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 11 087–11 093.