2025 IEEE International Conference on Robotics and Automation (ICRA)
May 19-23, 2025. Atlanta, USA

MARF: Cooperative Multi-Agent Path Finding with Reinforcement Learning and Frenet Lattice in Dynamic Environments

Tianyang Hu[1], Zhen Zhang[1], Chengrui Zhu[1], Gang Xu[1], Yuchen Wu[1,3], Huifeng Wu[2], Yong Liu[1,*]

***Abstract*— Multi-agent path finding (MAPF) in dynamic and complex environments is a highly challenging task. Recent research has focused on the scalability of agent numbers or the complexity of the environment. Usually, they disregard the agents' physical constraints or use a differential-driven model. However, this approach fails to adequately capture the kinematic and dynamic constraints of real-world vehicles, particularly those equipped with Ackermann steering. This paper presents a novel algorithm named MARF that combines multi-agent reinforcement learning (MARL) with a Frenet lattice planner. The MARL foundation endows the algorithm with enhanced generalization capabilities while preserving computational efficiency. By incorporating Frenet lattice trajectories into the action space of the MARL framework, agents are capable of generating smooth and feasible trajectories that respect the kinematic and dynamic constraints. In addition, we adopt a centralized training and decentralized execution (CTDE) framework, where a network of shared value functions enables efficient cooperation among agents during decision-making. Simulation results and real-world experiments in different scenarios demonstrate that our method achieves superior performance in terms of success rate, average speed, extra distance of trajectory, and computing time.**

I. INTRODUCTION

The application of multi-agent path finding has become increasingly widespread, spanning fields such as warehousing, drone swarms, and video games. As the number of agents increases and environments become more complex, ensuring collision avoidance while maintaining computational efficiency remains a significant challenge.

Algorithms designed to address this problem can be broadly categorized into two types: centralized planning and distributed execution. Centralized approaches are typically offline methods that require adequate knowledge of the states of all agents and obstacles, with a central controller generating a unified solution. Some well-known algorithms in this category include CBS [1], SIPP [2] and CL-MAPF [3].While these methods can typically provide optimal solutions, they struggle with dynamic obstacles due to their offline nature. Additionally, as the number of agents increases, the computational complexity of centralized algorithms rises sharply, resulting in a significant loss of efficiency. Another limitation is that these algorithms only generate paths, rather than time-dependent trajectories.

* Corresponding author. E-mail: yongliu@iipc.zju.edu.cn.
[1] are with the Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou 310027, China.
[2] is with the Hangzhou Dianzi University, Hangzhou 310018, China.
[3] is with the Polytechnic Institute of Zhejiang University, Hangzhou 310015, China.

This work was supported by National Natural Science Foundation of China (No.U21A20484).

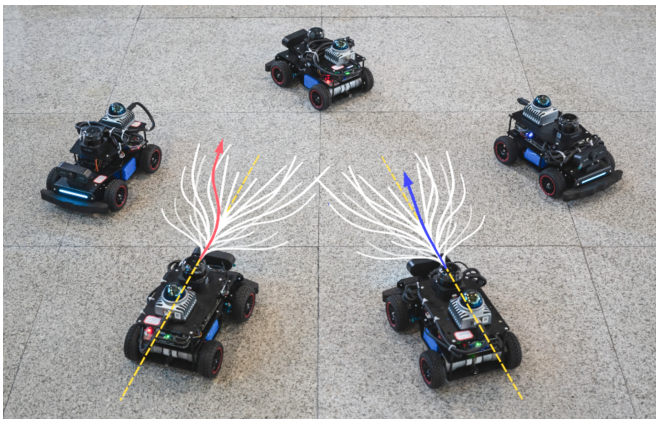


Fig. 1: The autonomous robots are performing navigation tasks. The white lines represent candidate trajectories generated by the Frenet lattice planner. A specific trajectory is selected based on our MARF algorithm. During navigation, they demonstrate effective collaboration and obstacle avoidance, enabling them to complete the task successfully.

In contrast, online distributed algorithms generally consume fewer computational resources and exhibit superior performance when handling dynamic obstacles. These algorithms can be further divided into velocity-space-based methods and RL-based methods. The most well-known method among the former is the Velocity Obstacle (VO) algorithm [4], which projects obstacles and other agents into the velocity space to create velocity obstacles. Agents must choose velocities outside these velocity obstacles to avoid collisions. However, this approach can lead to local minima when multiple agents are involved. To address this issue, the Reciprocal Velocity Obstacle (RVO) algorithm [5] extends the traditional VO approach by considering the reciprocal nature of agent interactions, ensuring that each agent adjusts its velocity to avoid collisions with others. However, RVO incurs higher computational costs and is still susceptible to local optima. Based on the RVO framework, the Optimal Reciprocal Collision Avoidance (ORCA) algorithm [6] is currently the most widely used. ORCA predicts the future trajectories of other agents and solves a linear programming problem to determine the optimal velocity, resulting in smoother trajectories and improved computational efficiency. Xu *et al.* [7] further incorporates vehicle posture constraints into this framework, allowing the algorithm to be applied to Ackermann steering robots.

Another major category of distributed algorithms is based on deep reinforcement learning (DRL). Some DRL ap-

2025 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3315-4139-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICRA55743.2025.11128009

proaches use grid maps, where a specific region of the map is provided as observations, and discrete directions (e.g., up, down, left, right) are used as the action space [8], [9], [10], [11]. Other approaches utilize sensor data as observations and employ a differential-driven robot. Chen *et al.* [12] uses ORCA to generate trajectories as training data, in order to restrict linear and angular velocities. Later, they [13] propose a socially aware path generation rule to describe and imitate human behaviors. Long *et al.* [14] introduces a hybrid strategy that combines PID controller and RL, allowing robots to complete tasks in large, complex environments. However, these algorithms fail to directly generate feasible trajectories that satisfy real-world Ackermann steering vehicles, which require a minimum turning radius.

Inspired by Cola-HRL [15], we propose our MARF (**M**ulti-**A**gent path finding with **R**einforcement learning and **F**renet lattice) algorithm. MARF generates trajectories that meet acceleration and curvature requirements, thereby complying with the Ackermann steering model, as shown in Fig. 2. Specifically, the main contributions of our work are as follows:

1) We parameterize the Frenet lattice planner's trajectories as the action space of MARL, enabling the agent to meet the kinematic and dynamic constraints of the Ackermann model.

2) By sharing the output of a value network across all agents, we facilitate effective coordination among agents, such as accelerating to overtake or decelerating to give way.

3) We conduct both simulations and real-world experiments in various dynamic environments, demonstrating that the proposed algorithm is robust and efficient.

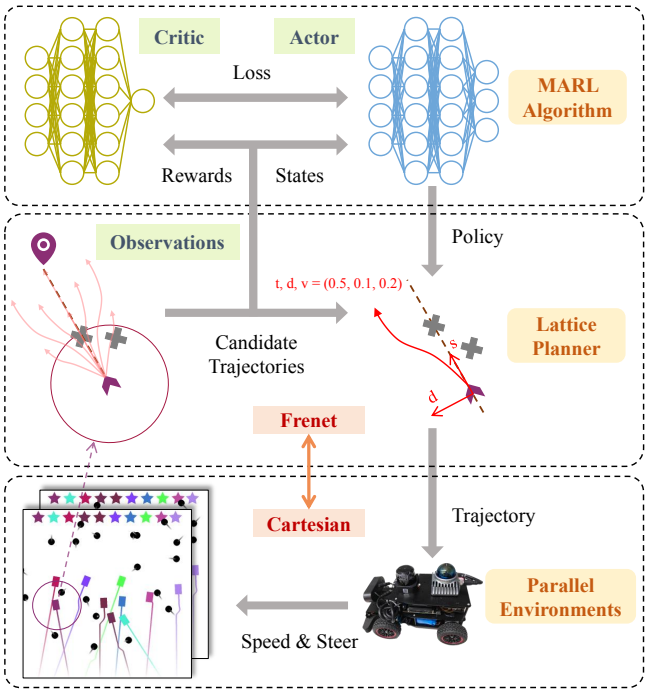


Fig. 2: The framework of MARF. We employed MAPPO [16] as the MARL algorithm. The policy network generates a trajectory in Frenet coordinate system that adheres to the kinematic constraints based on the limited observation space of each agent. The robots autonomously navigate the environment following this trajectory. Additionally, we utilized parallel environments to accelerate the training process.

II. PRELIMINARIES

A. Partially Observable Markov Decision Processes

In multi-agent systems, each agent has limited information about the environment. Therefore, we employ a Partially Observable Markov Decision Process (POMDP) to make decisions in scenarios with partial observability. Formally, a POMDP is defined by a tuple $(\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{R}, \mathcal{P}, \gamma)$. $\mathcal{N}$ indicates the number of agents. $\mathcal{S}$ is the set of possible states in the environment. $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is the set of possible actions that each agent can take. $\mathcal{O} = \mathcal{O}_1 \times \cdots \times \mathcal{O}_n$ is the set of all observations available to each agent in the current time step. $\mathcal{R} = r_1(s,a) \times \cdots \times r_n(s,a)$ is the set of reward functions for all agents. $\mathcal{P}(s'|s,a)$ is the state transition function, which represents the probability of transitioning from state s to state s' after taking action a. $\gamma \in [0,1)$ is the discount factor that determines the importance of future rewards. In practice, solving a POMDP involves finding a policy π that maps observations to actions, maximizing the expected cumulative reward over time.

B. Multi-Agent Proximal Policy Optimization (MAPPO)

The MAPPO algorithm [16] is a type of Actor-Critic algorithm, which combines policy gradient methods with value function approaches in RL. It simultaneously learns a policy network (Actor) and a value function network (Critic). The Actor network outputs the probability distribution of actions, while the Critic network provides an estimated value of the state.

The MAPPO algorithm builds on the Actor-Critic framework by using the importance sampling ratio and a clipping function to constrain the difference between the new and old policies. This approach maximizes the performance of incremental policy updates by an approximation. The importance sampling ratio represents the ratio of the probability distribution of the new policy to that of the old one sampled in the current state, which is defined as $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$. The clipping function is used to control the magnitude of policy updates, preventing excessive differences between the new and old policies, which is defined as $L^{clip}(\theta) = \min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)$. Here, $\hat{A}_t$ is the advantage function, the clip function restricts the range of values of the importance sampling ratio $r_t(\theta)$, and ϵ is a hyperparameter.

C. Frenet Lattice Planner

The lattice planner [17], [18], [19] is a forward search algorithm that discretizes the agent's configuration space into grids and searches for the optimal path on these grids, ensuring that the generated path fully satisfies the agent's kinematic and dynamic constraints. Typically, this algorithm is applied to the Frenet coordinate system, where the lane centerline serves as the reference line [20], [21].

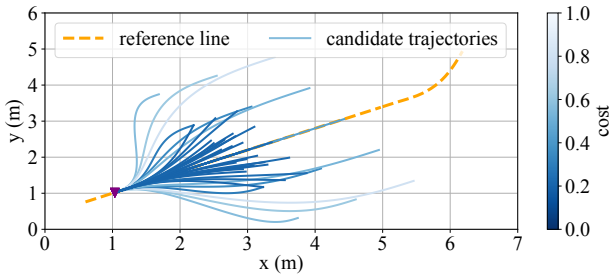


Fig. 3: The candidate trajectories are evaluated based on various metrics such as curvature, acceleration, and lateral offset. Each trajectory is assigned a different cost, with lower-cost trajectories indicating better adherence to kinematic and dynamic constraints.

As shown in Fig. 3, the longitudinal axis in the Frenet coordinate represents the distance traveled along a reference road, while the lateral axis measures the deviation from the reference line, making it ideal for representing curved paths. By sampling the Frenet space, polynomial trajectories are generated for both the lateral and longitudinal coordinates over time. These trajectories are then combined to form all candidate paths in the Cartesian coordinate system.

We model the agent as a nonholonomic system based on the Ackermann steering model. This model is particularly well-suited for real-world vehicles, where the steering mechanism limits lateral slip and prevents in-place turning. Taking the center of the vehicle's rear axle as the reference point, the position and velocity in the world coordinate system are defined as $\mathbf{p} = [p_x, p_y]^T$ and $\mathbf{v} = \dot{\mathbf{p}} = [v_x, v_y]^T$ respectively. The vehicle's kinematics are then defined as follows:

$$\begin{aligned} \dot{p_x} &= v\cos\phi, \\ \dot{p_y} &= v\sin\phi, \\ \dot{\phi} &= \frac{v\tan\delta_f}{L}, \end{aligned} \tag{1}$$

where (p_x, p_y) represents the vehicle position, ϕ is the vehicle's orientation, $v = \sqrt{v_x^2 + x_y^2}$ is the linear velocity, L is the wheelbase (the distance between the front and rear axes), and δ_f is the steering angle.

III. METHODOLOGY

This section introduces the proposed MAPF algorithm. We first detail the observation and action spaces, demonstrating how the Frenet lattice planner is integrated into the MARL framework. Following this, we introduce the design of the reward function and the entire training process.

A. Observation space

The observation of each agent is divided into two parts, as shown in Fig. 4. The first part o_{ego} refers to the ego states, including its velocity, as well as the position and orientation of the target relative to the agent. The second part o_{sur} consists of information about the surrounding environment. Mobile robots are typically equipped with various sensors, such as LiDAR, to collect environmental data, which is then transformed into a cost-map-like representation. Since real-world sensors often experience diminishing accuracy with increasing distance, we limit the agent's observation range to 5 meters. Specifically, the agent observes the positions and velocities of the nearest five other agents or obstacles within this range. If fewer than five are detected, the missing data is padded with zeros. This approach ensures that the dimension of observation space are not affected by the number of agents or obstacles. In addition, all position and velocity data are two-dimensional, so the total observation dimension amounts to 25.

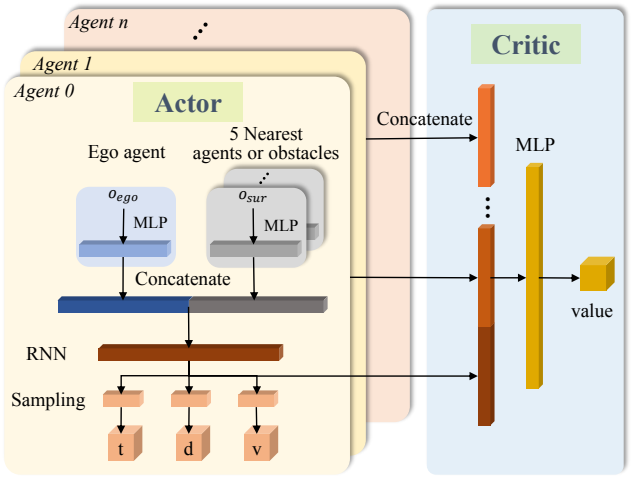


Fig. 4: Actor and Critic network. The input to the agent's policy network includes both its ego information and the states of the five closest neighboring agents or obstacles. This information is concatenated and processed through three distinct distribution networks to produce action spaces: t, d, v. The input to the value network consists of the concatenated Recurrent Neural Network (RNN) outputs from all agents' policy networks, producing a shared value estimate.

B. Action space

As shown in Fig. 2, the action space for each agent consists of the parameters t, d, v, where t is the trajectory time, d is the lateral offset, and $v = \dot{s}$ is the longitudinal velocity in Frenet. Specifically, we use the straight line connecting the agent's current position and the target position as the reference line. The longitudinal and lateral trajectories in the Frenet coordinate system are modeled using a quintic polynomial and a quartic polynomial, respectively. Boundary conditions are applied to set the lateral velocity, lateral acceleration, and longitudinal acceleration to zero at the end of the trajectory. Given that the agent's current state is known, we can establish $(T, d(T), \dot{d}(T), \ddot{d}(T) = (T_i, d_i, 0.0, 0.0))$ and $(T, s(T), \dot{s}(T) = (T_i, s_i, 0.0))$. By solving for the polynomial coefficients and performing coordinate transformation, we can obtain a trajectory in the Cartesian coordinate system. The trajectory parameters are shown in Table I.

TABLE I: The action space corresponds to the parameters of the lattice planner.

Parameter	Value
t (s)	0.5, 1.0, 1.5
d (m)	-0.2, -0.1, 0, 0.1, 0.2
v (m/s)	0, 0.1, 0.2, 0.3

C. Reward function

The reward function has several components, with the goal of guiding the agent to safely and efficiently move toward its target while satisfying the given constraints. To avoid the inefficiency caused by sparse rewards, the agent calculates the following rewards at each time step.

$$r_c = \begin{cases} -5 & \text{if } collide\ with\ agents \\ -10 & \text{if } collide\ with\ obstacles \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

where r_c is the penalty for collisions with other agents or obstacles, which discourages unsafe behaviors. The penalty for collisions with obstacles is set to be twice as high as that for collisions with other agents. This ensures that the total reward for both types of collisions is balanced so that they are treated equally from the perspective of the ego vehicle.

$$r_k = \begin{cases} -0.2 & \text{if } \kappa_{max} < \mathcal{K} < 2\,\kappa_{max} \\ & \text{or } |\mathcal{A}| - |a_{max}| > 0.5 \\ -0.5 & \text{if } 2\,\kappa_{max} < \mathcal{K} < 4\,\kappa_{max} \\ -1.0 & \text{if } 4\,\kappa_{max} < \mathcal{K} \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where r_k is the penalty for violations of kinematic and dynamic constraints, κ_{max}, a_{max} are curvature and acceleration limitations for the agent, and $\mathcal{K}, \mathcal{A}$ are maximum corresponding values in the current trajectory. Therefore, dynamic constraints are enforced by limiting $\mathcal{A}$, while kinematic constraints are limited by $\mathcal{K} = 1/R_{min}$, where R_{min} represents the minimum turning radius of agents.

$$r_l = -|d_{ego}|, \tag{4}$$

where d_{ego} is the ego agent's lateral derivation from the current reference line after executing an action, thus r_l discourages unnecessary lane changes.

$$r_g = 0.2 \cdot (||\mathbf{p}_{ego}^{t-1} - \mathbf{p}_{goal}|| - ||\mathbf{p}_{ego}^{t} - \mathbf{p}_{goal}||), \tag{5}$$

where r_g is the reward for approaching the target, and $\mathbf{p}_{ego}, \mathbf{p}_{goal}$ represent the Cartesian coordinate of ego agent and target. Additionally, a success reward R_s is given if the agent completes the task without any collisions and all trajectories satisfy the kinematic and dynamic constraints.

$$R_s = \begin{cases} 5 & \text{if } \|\mathbf{p}_{ego} - \mathbf{p}_{goal}\| < 0.1 \\ 10 & \text{if } \|\mathbf{p}_{ego} - \mathbf{p}_{goal}\| < 0.1 \\ & \text{and } |\theta_{ego} - \theta_{goal}| < 0.5 \\ 0 & \text{otherwise} \end{cases}, \tag{6}$$

where $\theta_{ego}, \theta_{goal}$ are the yaw angle in the Cartesian coordinate system of the ego and the target, in radians. Therefore, the final reward function is defined as:

$$r = r_c + r_k + r_l + r_g + R_s. \tag{7}$$

D. Training Process

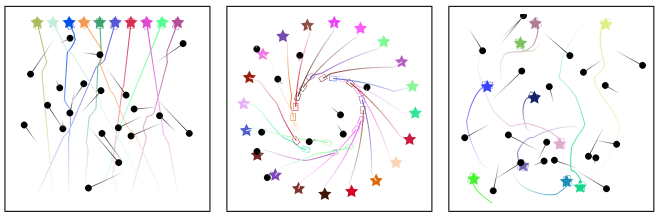
Many RL algorithms have been successfully applied to the field of robotic navigation. In this work, we utilize MAPPO as our MARL algorithm, which operates within a centralized training and decentralized execution framework, as shown in Fig 4. Specifically, each agent shares the same policy network, with different observations producing different trajectories. Additionally, all agents share a global value network, meaning that each agent's decisions will influence the others, promoting consensus and cooperation among the agents. Once the environment is initialized, the expected step size for each trajectory is determined based on the agents' distances or required speeds. Episodes are terminated if an agent either collides, exceeds the expected step size, or successfully completes the task.

IV. EXPERIMENTS

In this section, we introduce the scenario configurations and training parameters. And we compare our MARF with CL-MAPF [3] and PCA [7] across various metrics in different environments. Finally, real-world experiments demonstrate the robustness of our algorithm.

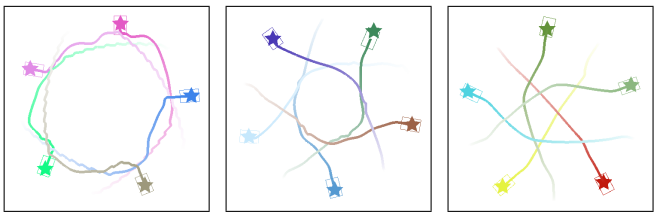
Algorithm 1 MAPPO with lattice planner

Initialize value network $V_\omega(s_t)$ and policy network π_θ;
repeat
  // Collect data in parallel environments
  Randomly reset environments;
  for $step = 1, 2, \ldots, T$ **do**
    for each agent $i = 1, 2, \ldots, N$ **do**
      Sample lattice parameters from policy π_θ
      Collecting observation, action, reward $\{o_i^t, a_i^t, r_i^t\}$ where $t \in [0, T]$
    end for
    break if all agents terminate;
    Estimate advantages by GAE [22] $\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}$;
  end for
  // Train policy network
  Calculate policy loss: $L(\theta) = \mathbb{E}[L^{clip}(\theta) + cS[\pi_\theta](s_t)]$, where $S[\pi_\theta](s_t)]$ is the entropy of the policy;
  Update policy parameters: $\theta = \theta + \alpha_\theta \nabla L(\theta)$;
  // Train value network
  Calculate value loss: $L(\omega) = (V_\omega(s_t) - \hat{A}_t)^2$;
  Update value parameters: $\omega = \omega + \alpha_\omega \nabla L(\omega)$;
until algorithm converged



(a) Crossing (b) Circle (c) Random

Fig. 5: Three training scenarios. At each environment reset, one of these scenarios is randomly selected. The agents' target points are represented by stars of the same color as the agent, while black circles indicate dynamic obstacles with a random direction and speed.



(a) Stage1 (b) Stage2 (c) Stage3

Fig. 6: Three stages of the training process. In (a), trajectories fail to meet kinematic and dynamic constraints. In (b), agents take a longer trajectory and move at a slower speed. In contrast, (c) demonstrates high-quality trajectories.

A. Environments and training details

As shown in Fig. 5, we use a 10m $\times$ 10m map with five to twenty agents and obstacles in the environment. The agents are modeled as rectangular objects with a length of 0.3m and a width of 0.2m. The radius of the obstacles is set to 0.1m.

We design several scenarios to train the agents' policies. In the crossing scenario (Fig. 5a), the start and target positions of the agents are randomly distributed on opposite sides of the map. Agents need to avoid dynamic obstacles by accelerating, decelerating, or changing lanes. In the circle scenario, each agent's target is directly opposite its starting point. This aims to emphasize collaboration among agents, like detouring in the same direction. Random scenario is designed to enhance the policy's generalization to unknown environments and prevent overfitting to the above two scenarios.

We adopt a curriculum learning approach [23] by dividing the training process into three phases to accelerate the convergence of the policy, as shown in Fig. 6. In the first stage, we only use the rewards r_c and r_s to ensure that agents sufficiently explore the action space, learn to avoid dynamic obstacles, and establish a connection to the target point. In the second stage, we introduce the reward r_k, which imposes kinematic and dynamic constraints on the agents, building on their basic ability to follow a global path. At this point, agents are capable of completing the task successfully. In the third stage, we incorporate the rewards r_d and r_g, guiding the agents to move quickly toward the target while minimizing unnecessary lane changes. This helps avoid inefficient detours and further improves the quality of the trajectories.

We conduct offline training on a platform equipped with an NVIDIA 4080 GPU and an Intel i7-13700K CPU, which takes approximately 5 hours. Both the Recurrent Neural Network (RNN) and Multi-Layer Perceptron (MLP) models consist of two layers, each with 64 units. Other hyperparameters are listed in Table II.

TABLE II: Hyperparameters in MARL training

Parameter	Value	Parameter	Value
learning rate	$7e-4$	entropy c	0.01
discount rate	0.99	GAE λ	0.95
PPO epoch	2	clipping ϵ	0.2
target KL	0.05		

B. Simulation Results

We select CL-MAPF [3] and PCA [7] as baselines for comparison. The former is an offline search-based algorithm, while the latter is an online method. We conduct 100 trials with varying numbers of agents and dynamic obstacles on different map sizes, as shown in Table III.

A critical metric in MAPF tasks is success rate. As the map size and numbers of agents increase, the success rates of CL-MAPF and PCA decrease significantly, while MARF maintains a degree of robustness. Despite being trained on a 10m $\times$ 10m map, the reinforcement learning framework allows MARF to generalize in different maps effectively. Furthermore, the lattice planner enhances agent navigation in complex environments by providing spatiotemporal trajectories.

In terms of other metrics, CL-MAPF, as a global path planner, does not account for time and velocity considerations. As a result, average speed is not a relevant performance metric for evaluating this algorithm. In contrast, MARF consistently outperforms PCA in this regard.

Additional distance traveled is defined as $1 - d_{min} \,/\, d_{act}$, where d_{min} is the shortest distance between the start and end points, and d_{act} represents the length of the actual executed trajectory. CL-MAPF provides an optimal theoretical solution when no dynamic obstacles are present. However, when dynamic obstacles are introduced, the pre-planned path may may result in collisions. In contrast, PCA often results in detours or deadlocks, especially in symmetric scenarios. By incorporating reference line information through the lattice planner, MARF directs agents along paths closer to their goals.

Moreover, MARF requires significantly less computation time. As the map size increases, CL-MAPF often exceeds the maximum search time limit (90s), and PCA's computation time also increases linearly with environmental complexity. In contrast, MARF consistently maintains an average computation time below 3 ms, showcasing strong real-time performance.

TABLE III: Comparison of several metrics over different experiment settings.

Environment Settings			Success Rate			Average Speed (m/s)			Extra Distance			Computing Time (ms)		
Map size	Agents	Obstacles	CL-MAPF	PCA	**Ours**	CL-MAPF	PCA	**Ours**	CL-MAPF	PCA	**Ours**	CL-MAPF	PCA	**Ours**
5m×5m	5	0	1.00	1.00	**1.00**	-	**0.27**	0.24	**0.01**	0.11	0.04	110	11.43	**1.18**
5m×5m	5	5	0.58	1.00	**1.00**	-	0.17	**0.21**	**0.16**	0.24	0.18	582	13.26	**1.25**
10m×10m	10	10	0.23	0.92	**1.00**	-	0.19	**0.25**	0.18	0.34	**0.13**	1127	30.25	**1.86**
10m×10m	10	20	0.17	0.80	**0.93**	-	0.23	**0.23**	0.26	0.32	**0.14**	2136	36.09	**1.82**
20m×20m	20	20	0.00	0.77	**0.98**	-	0.25	**0.29**	-	0.49	**0.36**	-	71.32	**2.83**
20m×20m	20	40	0.00	0.67	**0.86**	-	0.23	**0.26**	-	0.50	**0.39**	-	137.21	**2.81**

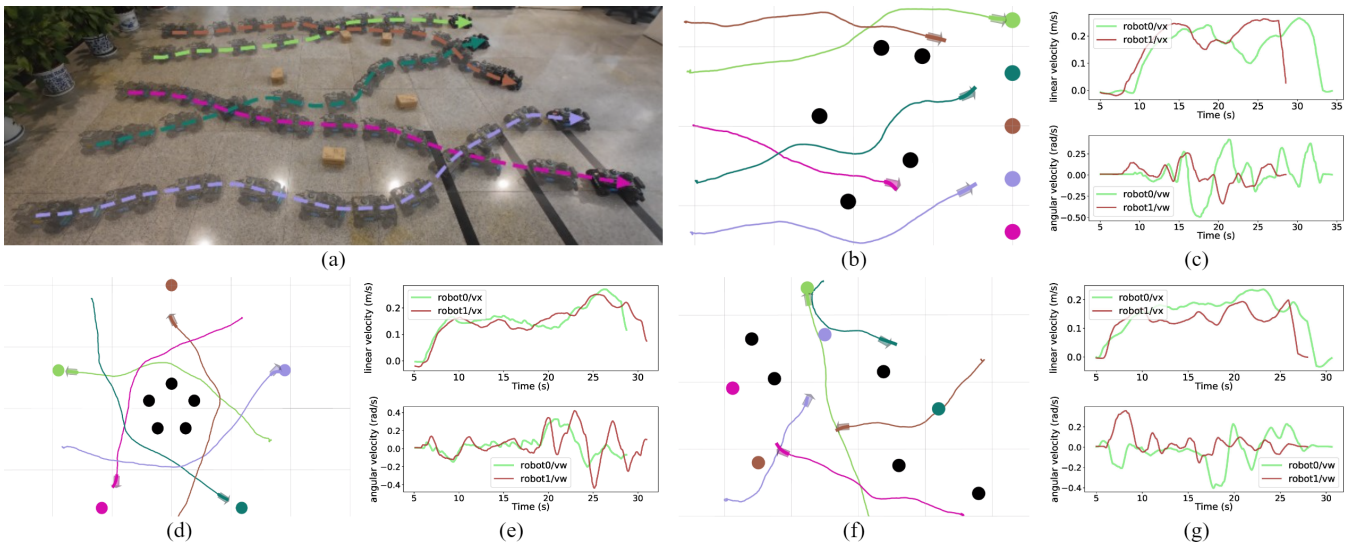


Fig. 7: Navigation tasks in the real world. (a) shows the trajectories of the robots moving from left to right in the crossing scenarios. (b) displays the corresponding visualization panel, where the black circles represent the obstacles depicted in (a). The arrows and colored circles indicate the positions and orientations of the robots, with their corresponding target points. The bold lines above the arrows represent the lattice trajectory output by the policy network at current step. (c) presents the linear and angular velocities of the two robots at the top of the scene. (d), (e) and (f), (g) are the visualization panels and velocities in circle and random scenarios.

C. *Real-world Experiments*

In addition to simulation experiments, we also test MARF in the real world. The size of the robots is consistent with the settings in the simulation, with a maximum speed of 0.6 m/s, a maximum acceleration of 0.5 m/s², and a minimum turning radius of 0.35 meters. Each robot uses a Jetson Nano for chassis control, operating on Ubuntu 20.04 and ROS Noetic. A mid360 LiDAR is mounted on the robots, with the PointLIO algorithm [24] used for localization, providing the robots' coordinates on the world map at a frequency of 10 Hz. Our algorithm is deployed on a laptop equipped with an Intel Core i7-1165G7 processor, 16GB of RAM, and an NVIDIA MX450 GPU. The laptop serves as the ROS master node, and communicates with the robots via a local area network (LAN) through a router.

The real-world experiment is conducted in a 4m×4m map environment. We deploy five robots to validate the algorithm across the three scenarios used in the simulation, as shown in 7. Fig. 7a illustrates the robots' motion trajectories in the crossing scenario, one of the three evaluated scenarios. The figures in the upper right (Fig. 7b, 7c), lower left (Fig. 7d, 7e), and lower right (Fig. 7f, 7g) represent the visualization panels and speed metrics in crossing, circle and random scenarios, respectively. In Fig. 7b, 7d and 7f, we can observe that the robot trajectories are smooth, with no sharp turns. Fig. 7c, 7e and 7g present the linear and angular velocities for two randomly selected agents. Although fluctuations in odometry feedback during the experiment caused some deviations in the feedback data, the overall curves remained smooth throughout the process.

V. CONCLUSIONS

In this paper, we propose a multi-agent path finding algorithm based on reinforcement learning and lattice planner. By parameterizing lattice planner in Frenet coordinate system as the action space for MARL, we generate trajectories that adhere to the Ackermann steering model. Training results show that our policy allows agents to effectively avoid dynamic obstacles and collaboratively complete navigation tasks. Both simulation and real-world experiments demonstrate the robustness and real-time performance of MARF. In the future, we plan to explore more complex environments with different shapes of obstacles.

REFERENCES

[1] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370214001386

[2] M. Phillips and M. Likhachev, "Sipp: Safe interval path planning for dynamic environments," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 5628–5635.

[3] L. Wen, Y. Liu, and H. Li, "Cl-mapf: Multi-agent path finding for car-like robots with kinematic and spatiotemporal constraints," *Robotics and Autonomous Systems*, vol. 150, p. 103997, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889021002530

[4] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The international journal of robotics research*, vol. 17, no. 7, pp. 760–772, 1998.

[5] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 1928–1935.

[6] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research: The 14th International Symposium ISRR*. Springer, 2011, pp. 3–19.

[7] G. Xu, Y. Chen, J. Cao, D. Zhu, W. Liu, and Y. Liu, "Multivehicle motion planning with posture constraints in real world," *IEEE/ASME Transactions on Mechatronics*, vol. 27, no. 4, pp. 2125–2133, 2022.

[8] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019.

[9] M. Damani, Z. Luo, E. Wenzel, and G. Sartoretti, "Primal$_2$: Pathfinding via reinforcement and imitation multi-agent learning - lifelong," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2666–2673, 2021.

[10] Z. Liu, B. Chen, H. Zhou, G. Koushik, M. Hebert, and D. Zhao, "Mapper: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 11 748–11 754.

[11] Z. Ma, Y. Luo, and H. Ma, "Distributed heuristic multi-agent path finding with communication," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 8699–8705.

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

[13] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1343–1350.

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

[15] L. Gao, Z. Gu, C. Qiu, L. Lei, S. E. Li, S. Zheng, W. Jing, and J. Chen, "Cola-hrl: Continuous-lattice hierarchical reinforcement learning for autonomous driving," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 13 143–13 150.

[16] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. M. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative multi-agent games," in *Neural Information Processing Systems*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:232092445

[17] M. Pivtoraiko and A. Kelly, "Efficient constrained path planning via search in state lattices," in *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*. Munich Germany, 2005, pp. 1–7.

[18] T. M. Howard, C. J. Green, A. Kelly, and D. Ferguson, "State space sampling of feasible motions for high-performance mobile robot navigation in complex environments," *Journal of Field Robotics*, vol. 25, no. 6-7, pp. 325–345, 2008.

[19] M. Pivtoraiko and A. Kelly, "Differentially constrained motion replanning using state lattices with graduated fidelity," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 2611–2616.

[20] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 1879–1884.

[21] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenét frame," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 987–993.

[22] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[23] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 41–48. [Online]. Available: https://doi.org/10.1145/1553374.1553380

[24] D. He, W. Xu, N. Chen, F. Kong, C. Yuan, and F. Zhang, "Point-lio: Robust high-bandwidth light detection and ranging inertial odometry," *Advanced Intelligent Systems*, vol. 5, 07 2023.