Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Low-cost Multi-Agent Navigation via Reinforcement Learning with Multi-Fidelity Simulator

JIANTAO QIU,(Student Member, IEEE), CHAO YU, WEILING LIU, TIANXIANG YANG, JINCHENG YU,(Student Member, IEEE), YU WANG,(Senior Member, IEEE), AND HUAZHONG YANG(Fellow, IEEE)

Department of Electronic Engineering, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing 100084, China

Corresponding author: Yu Wang (e-mail: yu-wang@tsinghua.edu.cn).

ABSTRACT In recent years, reinforcement learning (RL) has been widely used to solve multi-agent navigation tasks, and a high-fidelity level for the simulator is critical to narrow the gap between simulation and real-world tasks. However, high-fidelity simulators have high sampling costs and bottleneck the training model-free RL algorithms. Hence, we propose a Multi-Fidelity Simulator framework to train Multi-Agent Reinforcement Learning (MFS-MARL), reducing the total data cost with samples generated by a low-fidelity simulator. We apply the depth-first search to obtain local feasible policies on the low-fidelity simulator as expert policies to help the original reinforcement learning algorithm explore. We built a multi-vehicle simulator with variable fidelity levels to test the proposed method and compared it with the vanilla Soft Actor-Critic (SAC) and expert actor methods. The results show that our method can effectively obtain local feasible policies and can achieve a 23% cost reduction in multi-agent navigation tasks.

INDEX TERMS Deep reinforcement learning, intelligent robots, multi-robot systems, multi-fidelity simulators

I. INTRODUCTION

Multi-agent navigation is a fundamental problem in multi-robot tasks, which attempts to solve the problem of controlling multiple robots to move to their respective targets and avoid collisions. Multi-agent navigation is a critical technology in practical tasks such as unmanned warehouse logistics, unmanned search, rescue and autonomous driving, and has attracted researchers' extensive attention.

In most multi-agent navigation problems, robots take action based on their current coordinates, target coordinates, and information obtained by the sensor (laser, camera, etc.). There are already some velocity obstacle-based methods that can solve this problem, such as optimal reciprocal collision avoidance (ORCA) [1], reciprocal velocity obstacles (RVO) [2]. This approach requires a central node to collect all the information and control all the robots in a unified manner. The system performance is limited by the central node performance and communication bandwidth. To overcome the shortcomings of the centralized method, some researchers begin to study learning-based decentralized methods.

An increasing number of reinforcement learning (RL) methods are used to solve robotic control problems [3] [4]. The multi-agent navigation task can be abstracted into a multi-agent multi-step decision task. Each robot makes decisions according to the respective information input to complete the task together. Some model-free RL algorithms combined with deep learning can directly generate the end-to-end decentralized policy and achieve similar or even better performance than centralized methods.

Model-free RL algorithms are generally considered to have problems with low sample efficiency [5]. Several works focus on improving RL algorithms' sample efficiency, such as improving RL algorithms' exploration ability, model-based methods, and instance learning methods [6]. These methods aim to increase the utilization of existing data (using existing data to build models) and make full use of prior knowledge (learned from expert data) by increasing the diversity of collected data (better exploration capabilities).

In robotics, training in a simulated environment and testing in a physical environment is a general development process. To capture the dynamics of the actual environment more accurately and generate data closer to the physical world, people have made more effort in developing high-fidelity simulators (HF-Sim), which will make algorithms perform

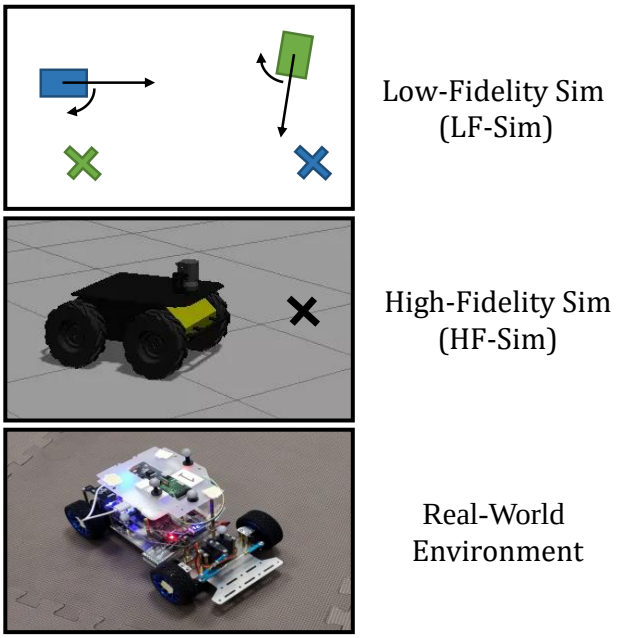

FIGURE 1. Pictures of low/high-fidelity simulators and photograph of real-world robot environment. The data cost of the three environments from top to bottom sequentially increased.

well in real applications. However, HF-Sims consume more computing resources, resulting in a high cost in training model-free RL algorithms.

Mark Cutler et al. [7] [8] proposed a multi-fidelity simulators RL algorithm that allows agents to learn approximate Q functions on low-fidelity simulators (LF-Sim) and use them to limit the state space that needs to be explored on HF-Sim. However, this method is hard to use in multi-agent tasks due to the discrete state setting and the inability to handle partial state representation. To reduce the total data cost using a multi-fidelity simulator approach in a multi-agent task, utilizing LF-Sim data that does not rely on the state-space setting is needed.

This paper proposes a framework of Multi-Fidelity Simulators for training Multi-Agent Reinforcement Learning, called the MFS-MARL framework. In this framework, the RL algorithm rollouts data from HF-Sim for training and detecting local difficult problems. The local feasible policies are generated by the depth-first search algorithm in the LF-Sim and combined with the original RL policy to help explore HF-Sim. The proposed method can improve the exploration ability of the RL algorithm and reduce the total data cost. Figure 2 summarizes the proposed method in an intuitive form.

The main contributions of this work include:

- A multi-fidelity simulators framework for training multi-agent reinforcement learning algorithms (MFS-MARL) is proposed. The reinforcement learning algorithm collects data on the high-fidelity simulators and generates expert policies by low-fidelity simulators to help reinforcement learning explore the state space.
- A backtrack-based local markov decision process (MDP) identifying method is proposed for decomposing the multi-agent task. A priority action queue-based depth-first search method is proposed for searching the local feasible policy on the local MDP as an expert policy.
- The experimental results show that MFS-MARL can save 23.0% of the data cost of vanilla SAC and achieve the convergence speed of a well-trained expert policy method.

The rest of the paper is organized as follows. The second section introduces the problem setting of the multi-robot system. The third section explains the relationship between this work and related work. The fourth section introduces the MFS-MARL framework and the RL algorithm training process. The fifth section introduces the experimental results and analyzes them. The last section summarizes the paper.

II. PROBLEM FORMULATION

A multi-robot task can be abstracted as a multi-agent Markov decision processes (MDP): $M =< \mathcal{S}^n, \mathcal{A}^n, P, R, O >$ where S and A are the state space and action space for one single agent. The probability of state transition is $P(S^n_{t+1}|S^n_t, a^n_t)$. For the partial observation task the observation function is $O^n_t = O(S^n_t)$. For tasks with assignable rewards, the reward function can be defined as $R^n_t = R(S^n_t, a^n_t, S^n_{t+1})$.

The optimization object for the RL algorithm is the average cumulative reward:

$$\mathbb{E}_{S_t\sim\rho(\pi),a_t\sim\pi,S_{t+1}\sim P(S'|S,a)}[\sum_{t=0}^{T} R(S_t, a_t, S_{t+1})] \quad (1)$$

where $\rho(\pi)$ is the state frequency function under policy π.

The simulator usually consists of two parts: state simulation + observation data rendering. The function of state simulation is to calculate the next state according to the current state and action by $S^n_{t+1} \sim P(S^n|S^n_t, a^n_t)$. The observation data rendering generates observation data according to the current state by $O^n_t = O(S^n_t)$.

In the field of RL, data are organized in the form of transition : $< S^n_t, a^n_t, S^n_{t+1}, R^n, O^n_t, O^n_{t+1} >$. The simulator needs to generate transition data through calculation, mainly for state simulation and observation data rendering. We use C to represent the average cost for generating one transition by the simulator. The average cost C is related to the number of agents, the fidelity level of state simulation and observation data rendering. A higher fidelity level usually means a higher average cost.

In this paper, the high-fidelity simulator (HF-Sim) and low-fidelity simulator (LF-Sim) have the same state space, action space, and reward function: $\mathcal{S}^n, \mathcal{A}^n, R$. HF-Sim has transition probability P_H, observation function O_H and average cost C_H. Conversely the LF-Sim has P_L,O_L and C_L.

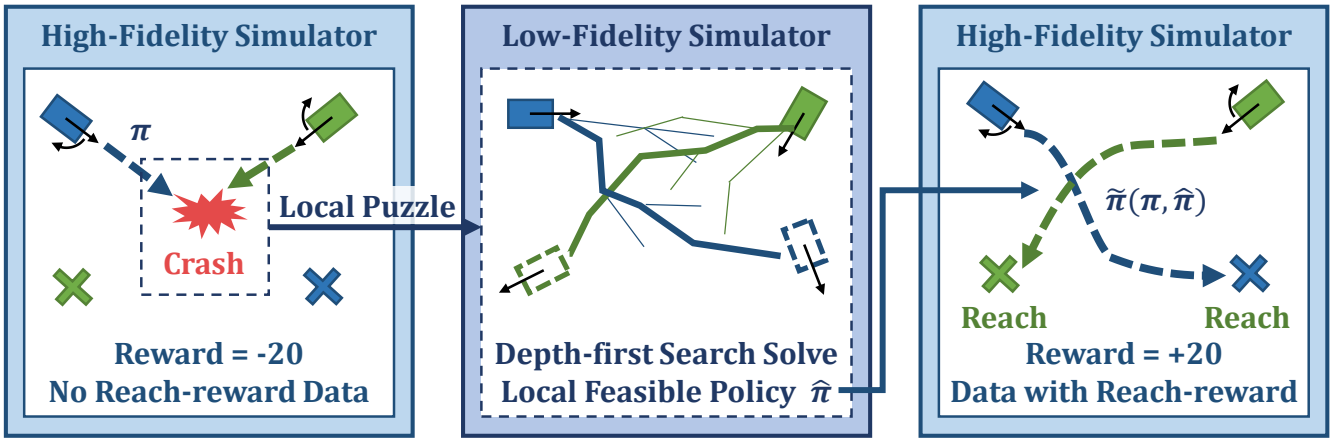

FIGURE 2. Diagram of the principle of the MFS-MARL algorithm, the reading order is from left to right. In the multi-agent navigation task, there will be a local puzzle where vehicles collide with each other and cannot move on. As the vehicle does not reach the target point, reach-reward cannot be obtained, which slows down the reinforcement learning algorithm training process. The proposed method uses the depth-first search method to solve the local puzzle on the low-fidelity simulator and obtain the local feasible policy $\hat{\pi}$. The local feasible policy $\hat{\pi}$ and the trained policy π are combined as mixed policy $\tilde{\pi}$ to rollout data on the high-fidelity simulator and obtain data with reach-reward to speed up the reinforcement learning algorithm training process.

III. RELATED WORK

Multi-Agent Navigation: There are massive works to solve multi-robot obstacle avoidance with the RL algorithm and simulator. For example, Long et al. [4] used the PPO algorithm end-to-end training obstacle avoidance algorithm on the Stage simulator. Ding et al. [9] trained the DDPG and HMM joint algorithm on the Gazebo simulator [10] to solve the multi-vehicle collaborative path planning. Tai et al. [11] trained ADDPG on V-REP [12] to solve the Mapless Navigation problem and Kahn et al. [13] trained a model-based RL algorithm on open raves to solve obstacle avoidance. These methods evaluate their performance on the simulator or the real world but only use single fidelity level data.

Imitation learning: Currently, there is a small number of works on imitation learning to solve multi-agent problems [14] [15]. They prove that using imitation learning on multi-agent tasks can effectively improve learning efficiency. However, obtaining expert data on multi-agent tasks is usually not easy and can be costly. This work uses a low-fidelity simulator to perform the depth-first search, aiming to find an approximate solution to a local problem to an inexpensive source of expert data.

RL sample efficiency: There are many studies on improving data efficiency, [6]. including exploration [16], model-based RL [17], transfer learning [18] [19], Hierarchical RL [20]. These methods all reduce training overhead by reducing the total quantity of data. However, this paper focuses on the cost difference between different data sources.

Multi-fidelity simulators: The work most relevant to this article include Cutler et al. [8] [7], which used different fidelity level simulators as data sources to train reinforcement learning algorithms utilizing the value learned in low-fidelity simulators. The difference in function and fidelity between different simulators induces a search policy in the high fidelity simulator. However, their work is hard to extend to large state-space scenes. Our work does not depend on state-space settings.

IV. MULTI-FIDELITY SIMULATORS MULTI-AGENT REINFORCEMENT LEARNING FRAMEWORK

The MFS-MARL framework includes the following parts: **(a) a High-Fidelity Simulator**, **(b) a MARL Algorithm**, **(c) a Local Puzzle Detector**, **(d) a Low-Fidelity Simulator**, **(e) a Local Puzzle Solver**, **(f) a Policy Mixer**. The high-fidelity simulator produces high-quality data for the MARL algorithm. The MARL algorithm stores the data in the replay buffer and updates the neural-network-based policy in an off-policy way. The local puzzle detector collects the state information of crashed vehicles and delivers it to the low-fidelity simulator. The local puzzle solver applies the depth-first search method to solve the local puzzle in the low-fidelity simulator and returns the local feasible policy to the policy mixer. The policy mixer replaces the actions of the crashed vehicles in the origin exploration policy. The framework is shown in Figure 3.

Compared to the vanilla multi-agent RL training framework, the MFS-MARL framework has a local puzzle detector to find the local puzzle, a low-fidelity simulator to simulate the local puzzle, a local puzzle solver to solve the local feasible policy, and a policy mixer to apply the local feasible policy in the high-fidelity simulator to rollout data with reach-reward. In the following subsections, the details of these additional components are explained.

A. LOCAL PUZZLE DETECTOR

We use the temporal locality and the spatial locality of agent interaction in multi-agent tasks to define the local puzzle. In multi-agent scenarios, the interaction between agents is local in space and time. The state transition of some agents over some time is independent of other agents' states and actions. The whole process can be divided into a series of particular periods so that for any period $[t_{start}, t_{end}]$ the agents can

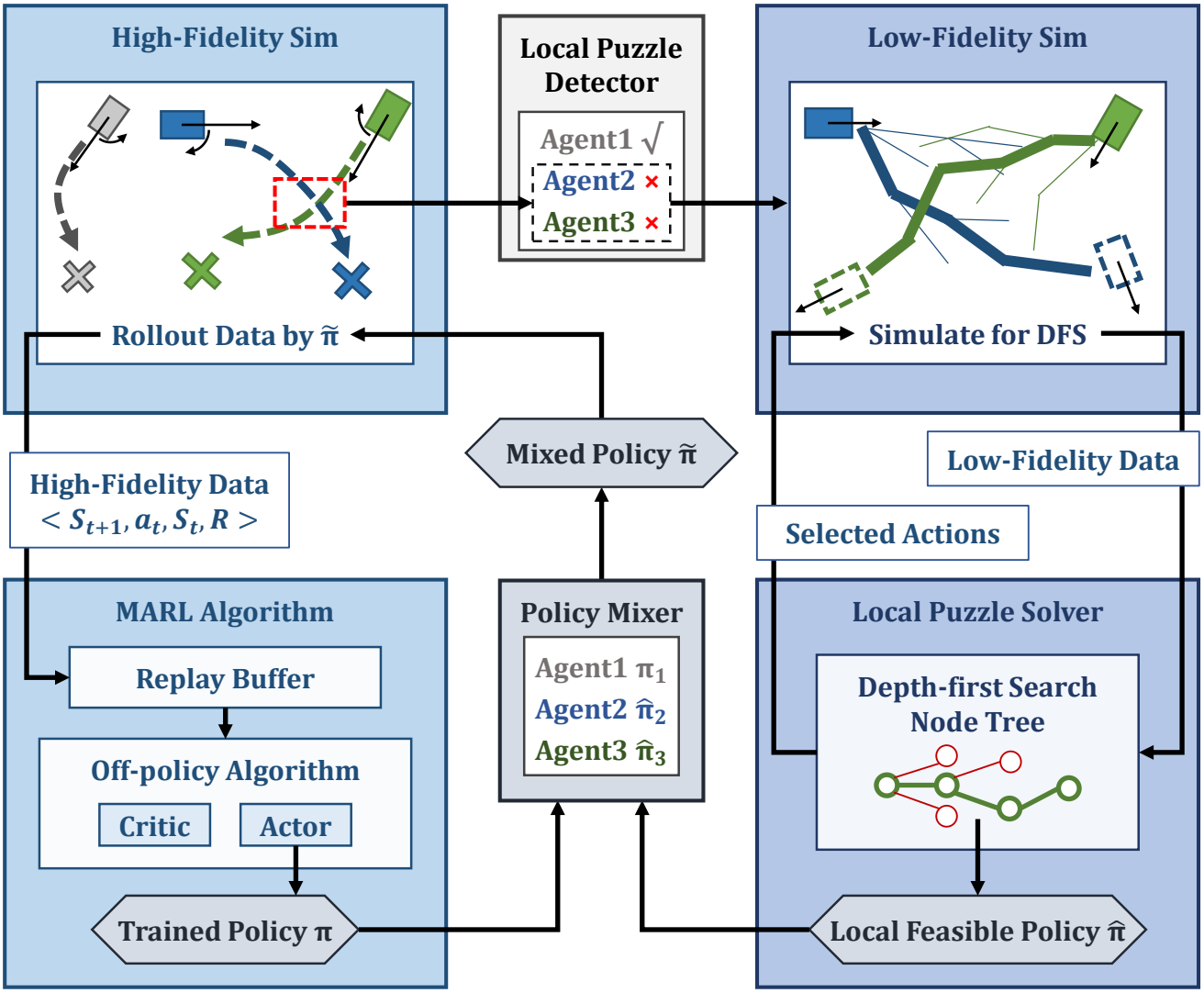

FIGURE 3. Diagram of multi-fidelity simulators multi-agent reinforcement learning framework. The algorithm framework includes the following parts: **High-Fidelity Simulator** produces high-quality data required by the MARL algorithm; **MARL Algorithm** stores the data in the replay buffer and updates the "trained policy" by off-policy method, the trained policy is the final output of the entire algorithm framework; **Local Puzzle Detector** extracts the Local puzzle in HF-Sim; **Low-Fidelity Simulator** simulates the local puzzle and provides the low-quality data required by the local puzzle solver; **Local Puzzle Solver** uses depth-first search method to build a search tree and solve local-feasible policy; **Policy Mixer** replaces the collision-causing part of the trained policy with the local-feasible policy and generates a mixed policy for rollout data in the high-fidelity simulator.

be split into two parts: $\{dep\}$ and $\{indep\}$. Any agent in $\{indep\}$ is independent of any other agent, and any agent in $\{indep\}$ depends on one or several agents in $\{dep\}$. For the i-th agent $i \in \{dep\}$, the transition probability can be written as

$$\begin{aligned} &P(S_{t+1}^{i}|S_t^{\{dep\}}, a_t^{\{dep\}}, S_t^{\{indep\}}, a_t^{\{indep\}}) \\ &= P(S_{t+1}^{i}|S_t^{\{dep\}}, a_t^{\{dep\}}) \end{aligned} \tag{2}$$

in which $t \in [t_{start}, t_{end}]$. A local solution can be searched in the local scope on $\{dep\}|_{t_{end}}^{t_{start}}$ without considering the influence of other parts.

In real-world applications, the state transition may be stochastic and related to the current policy. It is not always easy to predict and divide $\{dep\}|_{t_{end}}^{t_{start}}$. Here, we propose a local MDP decomposition method based on backtracking. The exploration policy is used for rollout in the data collection phase. When some agents run into the collision state or other states that make it difficult to continue the exploration at time t, the local MDP $\{dep\}|_{t+T}^{t-T}$ can be defined with $\{dep\}$ and T. $\{dep\}$ is the set of agent index indices involved in the collision, and T is the interactive time window.

The local MDP $\{dep\}|_{t_{end}}^{t_{start}}$ has a smaller state action space and time horizon related to the origin problem $\{all\}|_{t_{finish}}^{t_0}$. The local MDP can be reproduced on a low-fidelity simulator and a feasible solution can be obtained using a search-based local policy search algorithm.

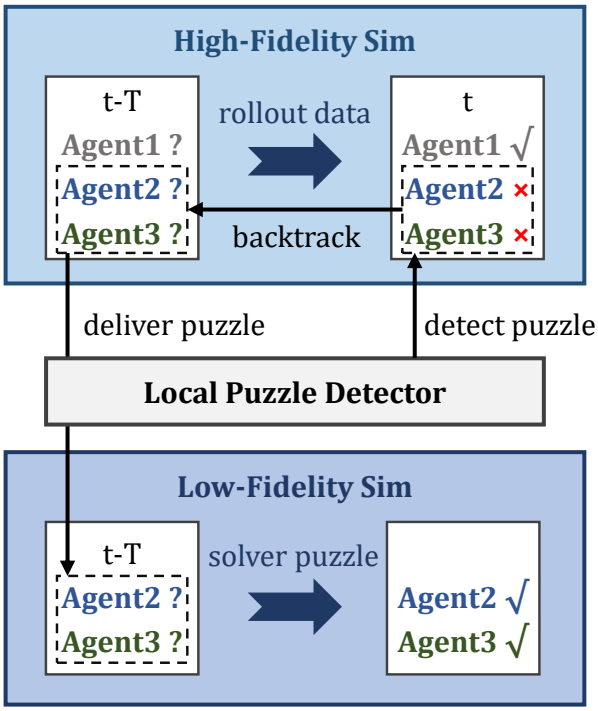

FIGURE 4. Diagram of local puzzle detector. The local puzzle detector extracts state information of the vehicles which crashed in time step $t - T$ from the high-fidelity simulator when a collision happened in time step t and delivers the state information to the low-fidelity simulator.

B. LOCAL PUZZLE SOLVER

The local puzzle solver searches the local feasible policy by the depth-first search (DFS) method with the initial state $S^{\{dep\}}$ on $\{dep\}|_{t_{end}}^{t_{start}}$. This subsection describes the DFS method for searching for local feasible policies on local MDPs.

First, a preferred action sequence $que^i = a_0^i, \cdots a_{|A|-1}^i$ is constructed for the state of the i-th agent S^i, and for all $|\{dep\}|$ agents, $que^{\{dep\}}$ generates $|A|^{|\{dep\}|}$ actions. To choose higher priority combined action for all agents, $que^{\{dep\}}$ is sorted in ascending order by the sum of subscripts of all actions. The DFS method preferentially selects the combined action with more 0 subscripts and always takes a_0^i as the first combined action.

The prior action a_0^i for each agent is determined by the relative position of the vehicle and its target point. If the target point is in the narrow area in front of the vehicle or back of the vehicle, the prior action is straight forward or straight backward. If the target point is in the turning limit of the car (out of the turning radius circle), the prior action is going forward or backward with turning to the target side. If the target point is too close to the vehicle and in the turning radius circle (beyond the turning limit of the car), the agent needs to adjust the position by the opposite action. The prior action map is shown in Figure 5.

Then set the initial state as the root node and apply DFS on the local MDP $S_{t_{start}}^{\{dep\}}$. The action $a_t^{\{dep\}}$ generated by

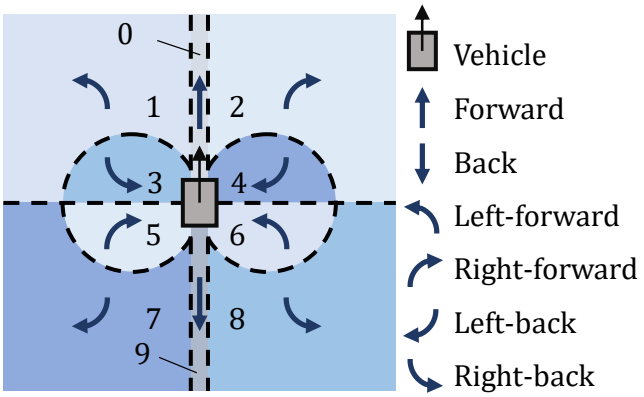

FIGURE 5. The prior action map for agents. In this picture, we use the frame of reference in which the vehicle is fixed in the middle of the map. The plant represents the possible area of the target point. The narrow area in the middle means that the vehicle can move to the target point without turning direction. The radius of the pair of circles drawn with dashed lines is the turning radius of the vehicle. The arrow in each area represents the best current action of the vehicle to reach the target point in the area.

$que^{\{dep\}}$ is taken in the simulator at the state $S_t^{\{dep\}}$, and the simulator will return the next state $S_{t+1}^{\{dep\}}$. We change the search step scale to t_{multi} times the original MDP, and the same action is repeated t_{multi} times on the original MDP. This "multiple-step" approach can further reduce the scale of the search problem.

The DFS algorithm will finish and return a local feasible policy after solving the current local puzzle, or finish and return "failure" at $t = t_{end}$. The local feasible policy will replace the exploration policy in the RL algorithm as an expert policy for $\{dep\}|_{t_{end}}^{t_{start}}$. We use $t_{max} = t_{end} - t_{start}$ to limit the DFS process to not be too long. Figure 6 is an intuitive example to explain the DFS algorithm we use.

Combining the pruning search space, limiting the max search steps, and rearranging the search order can limit the total search cost. In Section 5, we proved through experiments that these methods can avoid the exponential explosion of search costs.

C. POLICY MIXER

The policy mixer replaces actions of the crashed vehicles in the origin trained policy by the local feasible policy to help the crashed vehicles avoid collision and reach the goals. This subsection describes the details of the policy mixer.

As described above, the local policy $\hat{\pi}$ is searched on the local MDP $\{dep\}|_{t_{end}}^{t_{start}}$, so it needs to mix with the original policy π to form a mixed policy $\tilde{\pi}(S_t, \pi, \hat{\pi})$ as:

$$\tilde{\pi}^i(a_t^i|S_t^{all}) = \begin{cases} \hat{\pi}_t^i & t \in [t_{start}, t_{end}], i \in \{dep\} \\ \pi^i(a^i|S_t^{all}) & \text{otherwise} \end{cases} \tag{3}$$

Using mixed policy can help the i-th agent ($i \in \{dep\}$) solve the local puzzle and keep other agents from continuing to collect data by the original trained policy. It is worth noting that our mixed policy is only used for exploration during the

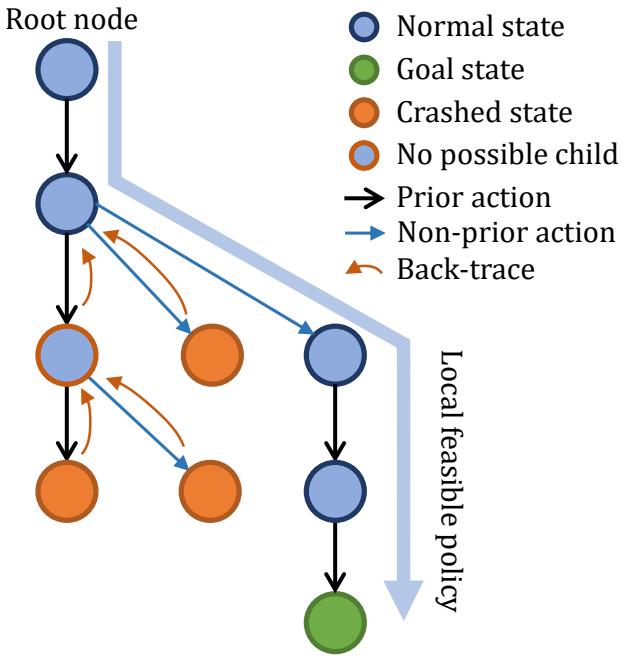

FIGURE 6. Diagram of depth-first search method. The depth-first search algorithm takes the initial state of the local puzzle as the root node. When the search reaches a certain node, the algorithm will remove the combined action from the preferred action sequence of this node, and use the combined action to simulate the low-fidelity simulator and generate a new state as a child node. If there is a collision in the new state, the search tree is backtracked; otherwise, the search is continued. Once all agents in the new state reach the target point, the search ends, and the local feasible policy is output as the result.

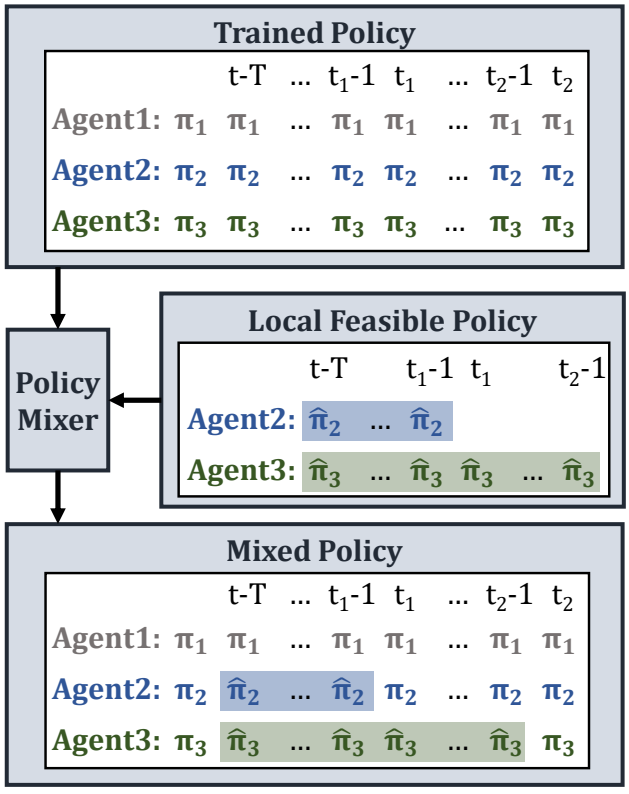

FIGURE 7. Diagram of policy mixer. The mixed policy uses actions from local feasible policy instead of the original trained policy. If any agent in the local puzzle reaches the goal, then it will use the actions from trained policy.

Algorithm 1 Multi-Fidelity Multi-Agent RL Algorithm

Require: High-Fidelity Sim : Σ^H,Low-Fidelity Sim : Σ^L,
Critic : $Q(S,a)$, Actor : π, Data Buffer : D
Depth-First Search Algorithm : $DFS(\cdot,\cdot)$
Ensure: Trained Policy: π
$D \leftarrow \varnothing, \pi \leftarrow \pi^{init}, Q(S,a) \leftarrow Q^{init}$,
$\hat{\pi} \leftarrow [], history \leftarrow []$
repeat
  $t \leftarrow 0, S_t \leftarrow$ init state of Σ^H
  repeat
    $history[t] \leftarrow S_t$
    $a_t = \tilde{\pi}(S_t, \pi, \hat{\pi})$
    $S_{t+1} = \Sigma^H.step(a_t)$
    if Not in a local puzzle **then**
      $D \leftarrow D \cup < S_t, a_t, S_{t+1}, R(S_t, S_{t+1}) >$
      $t \leftarrow t+1$
    else
      $t \leftarrow t - T_H$
      $\hat{\pi} \leftarrow DFS(history[t], \Sigma^L)$
    end if
  until $t > T_{finish}$
  Update π and $Q(S,a)$ via D
until π converges

training phase. In the test phase, only the trained policy is used without the local feasible policy.

In conclusion, we introduced the MFS-MARL framework to reduce RL training data costs. Using the backtrack-based local MDP decomposition method, the search space is much smaller than the original space, and the prior-action-based DFS method can find the local feasible policy as an expert policy. The overall algorithm is shown in Algorithm 1.

V. EXPERIMENTS AND RESULTS

In this section, we introduce the configuration of experiments and report the results. First, we describe the simulation setting and obtain the relationship between the simulator cost and performance loss through experiments. Then we measured the success rate and complexity of the proposed depth-first search method running on the simulator and determined the best configuration based on the results. Finally, we evaluated our algorithm on the multi-vehicle navigation task and compared it with the baseline and the well-trained expert policy method. The experimental results show that our method can effectively reduce the data cost.

A. SIMULATORS WITH MULTI-FIDELITY LEVEL

We use a simulator that simulates the four-wheeled vehicle movement on a two-dimensional plane to train the reinforcement learning algorithm. $N_{vehicle}$ vehicles are randomly placed on the plane in the training scenarios, and each vehicle is assigned a target point.

Action Space: The action that each vehicle can take is $a = [a_{vel}, a_{phi}]$, where $a_{vel} \in [-1, 1]$ controls the velocity of the vehicle and a_{phi} controls the front wheel angle for steering.

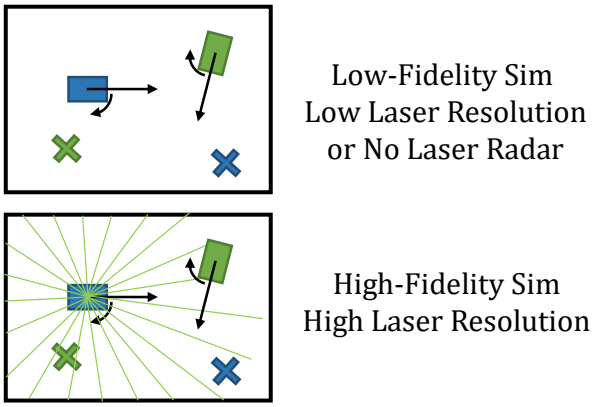

FIGURE 8. Low/high-fidelity multi-vehicles simulator with different laser resolutions. In the upper subpicture, the low-fidelity simulator does not simulate the laser radar, which reduces the computational cost. In the subpicture below, the high-fidelity simulator simulates high-resolution laser radar data and passes the data to the reinforcement learning algorithm for training.

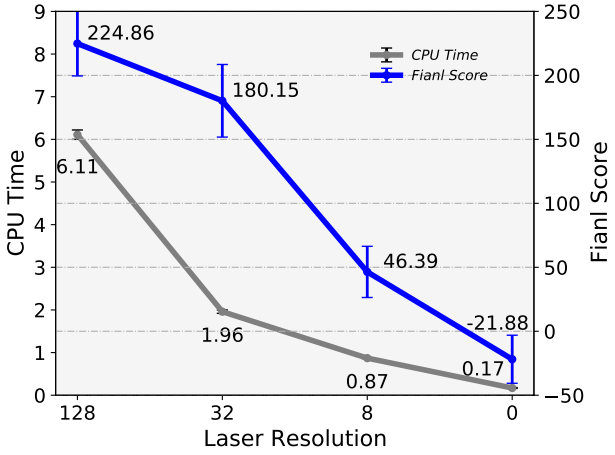

FIGURE 9. Graph of the execution time of the simulator with different laser resolution settings and the evaluation performance on the high-fidelity simulator. We record the execution time of the simulator with laser resolutions of 0,8,32 and 128 and plot the results in the gray curve. We train the policy by a baseline SAC algorithm then evaluate it on the high-fidelity simulator and plot the results in the blue curve.

State Space: The global state of the simulator can be represented by the combination of $3N_{vehicle}$-dim vehicle coordinates and $2N_{vehicle}$-dim target coordinates:

$$S_{global} = [x^i, y^i, \theta^i, x_t^i, y_t^i], i \in \{1 \cdots N_{vehicle}\}$$

Observation: We use a partial observation setting. Each vehicle can observe the relative coordinates $O_{pos} = [\widetilde{x}, \widetilde{y}]$ of the assigned target point relative to itself, and the laser sensor information $O_{laser} = [l_1, l_2, \cdots, l_{N_{laser}}]$, where the maximum measuring distance is 3 and N_{laser} is the resolution of the laser.

Reward: Each vehicle receives $r_{reach} = 10$ points for reward when reaching the target point and obtains $r_{crash} = -10$ when crashing into other vehicles.

B. PERFORMANCE GAP AND DATA COST

We conduct experiments to measure and compare the performance gap and data cost of simulators with different fidelity levels. The experiment uses different laser mine resolutions N_{laser} as different fidelity levels. We set $N_{laser} = 128$ as HF-Sim and $N_{laser} = 32, 8, 0$ as LF-Sim[1]. The simulators are shown as in Figure 8.

We use a desktop computer equipped with an Intel i9-7920X CPU and two GTX 1080Ti GPUs for all experiments. The operating system is Ubuntu 16.04 LTS and uses the configuration of Python 3.7.9 + PyTorch 1.7.0.

We record the execution time of the simulator under the 'naive policy' (that is, each vehicle moves to the target point directly) 100 times and plot the mean CPU time in Figure 9. The CPU time of a simulator with $N_{laser} = 128$ is 6.11s, which is approximately 7 times that of $N_{laser} = 8$ and 36 times that of a simulator without a laser sensor. We use $C_H : C_L = 36$ as the cost ratio between low fidelity and high fidelity.

[1]The O_{laser} spans from low resolution to 128 when $N_{laser} = 32, 8$. The O_{laser} is a constant vector when $N_{laser} = 0$.

To measure the performance gap, we train the policy by the baseline SAC algorithm on simulators with different N_{laser} levels and evaluate the policy on HF-Sim $N_{laser} = 128$. We use the final score gap between HF-Sim and other levels to measure the performance loss caused by the data of low-fidelity simulators.

The results show that the policies trained on the lower N_{laser} level simulators cause more performance loss. This phenomenon means that directly using the data of the low-fidelity simulators for training will harm the final performance on the high-fidelity simulator.

C. DFS SUCCESS RATE AND SAMPLE COMPLEXITY

The DFS process in our proposed algorithm includes two parameters: the maximum search step t_{max} and multiple search step t_{multi}. The first parameter can prevent the search process from being too long, and the second parameter can control the complexity of the search. The following are the scanning experiments on these two parameters.

We applied DFS in random scenarios with different numbers of vehicles to search for the feasible policy in which all vehicles can reach the target point without collision. Table 1 records the success rate and average step number under the condition of $t_{max} = 10^4$ with $N_{vehicle} = [2, 6] \in \mathbb{N}$ and $t_{multi} = 1, 2, 3$.

The result shows that more vehicles cause a lower success rate and larger sample number. In general, the vehicle collisions are mainly two cars, three cars rarely occur, and four cars and above are almost absent. According to the results of two cars and three cars, $t_{multi} = 2$ can obtain a lower average number of samples, and the number of samples is less than 100 in 99% of the cases. We use $t_{multi} = 2$ and $t_{max} = 100$ as the default settings in the following experiments.

$N_{vehicle}$	$t_{multi}=1$		$t_{multi}=2$		$t_{multi}=3$	
	rate	step	rate	step	rate	step
2	**0.98**	25.0	0.96	**21.9**	0.81	34.4
3	**0.95**	51.1	0.92	**37.0**	0.59	53.0
4	**0.91**	126.5	0.83	**66.3**	0.42	84.4
5	**0.86**	295.8	0.74	**118.2**	0.29	139.3
6	**0.78**	489.3	0.63	**241.4**	0.20	255.2

TABLE 1. Success rate and average search step of DFS on random scenarios under different settings. We applied DFS in random scenarios with different numbers of vehicles to search for the feasible policy and record the success rate and the average search step. The $t_{multi}=1$ has the highest success rate for all cases and $t_{multi}=2$ has a minimum number of search steps.

D. DATA COST REDUCTION VIA MULTI-SIMULATOR

We compare our proposed method with the baseline SAC algorithm on the simulator. Each experiment is repeated and conducted with 8 seeds. The algorithm trains for 1000 cycles and evaluates the policy on the HF-Sim every 10 cycles (10 cycles for 1 epoch). In each cycle, the algorithm rollouts 100 transition samples and stores them in a data buffer and then updates the SAC network parameters for 20 times.

Figure 10 is the score curve of the training process, the y-axis is the episodic score, and the x-axis is the epoch number. The red curve is for the proposed method, and the gray curve is for baseline SAC. By taking the baseline algorithm final score of 224.3 as the standard, our method can reduce the training process by 23.0%. By taking 90% of the baseline algorithm final score 201.9 as the standard, our method can reduce the training process by 22.1%.

The mean total number of samples of the LF-Sim used in the training process is 1936.0 in the experiments. 10^5 HF-Sim samples are used to train the SAC in the training process (100 epochs $\times$ 10 cycles $\times$ 100 samples). The data cost of LF-Sim is $1936.0/(36\times10^5)\approx5.4\times10^{-4}$ times of the HF-sim data with $C_H:C_L=36$ and is negligible relative to the overall cost. The experimental results prove that the proposed method can effectively reduce the data cost.

To compare our local policy and well-trained expert policy, we selected the policy network with the highest final performance from the policies trained by our method as the well-trained expert policy. This well-trained expert policy is used to replace the local feasible policies and trained by the same configuration as our method as the well-trained expert policy method. The results show that our method can achieve almost the same performance as the well-trained expert policy method and use fewer HF-Sim data.

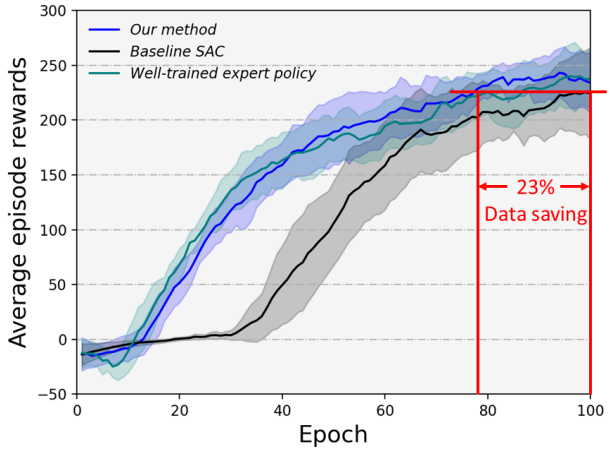

FIGURE 10. The graph of the episodic reward for baseline SAC (black), the proposed method (blue) and the well-trained expert policy method (green). The dark solid line is the mean score over all seeds, and the light translucent area represents the standard deviation. By taking the baseline SAC final mean score 224.3 as the standard, our method can reduce the training process by 23.0%

VI. CONCLUSION

In this paper, we propose a multi-fidelity simulator framework, which is named MFS-MARL, to improve the data efficiency of multi-agent reinforcement learning. The key challenge is how to identify interactive scenarios and utilize data from different fidelity simulators. To achieve joint training and accelerate convergence, we introduce a backtrack-based local MDP identifying method for decomposing task space and a depth-first search with a priority action queue for searching local feasible policy as an expert policy. We trained the SAC algorithm by the MFS-MARL framework and compared it with the baseline version SAC and the well-trained expert policy method. The experimental results show that our method can save 23.0% more high-fidelity data than vanilla SAC and achieve the same data efficiency as the well-trained expert policy method but use fewer HF-Sim data.

REFERENCES

[1] D. Bareiss and J. van den Berg, "Generalized reciprocal collision avoidance," *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1501–1514, 2015.
[2] 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*. IEEE, 2008, pp. 1928–1935.
[3] 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)*, May 2017, pp. 285–292.
[4] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 6252–6259.
[5] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," in *Advances in Neural Information Processing Systems*, 2018, pp. 8224–8234.
[6] Y. Yu, "Towards sample efficient reinforcement learning." in *IJCAI*, 2018, pp. 5739–5743.
[7] M. Cutler, T. J. Walsh, and J. P. How, "Real-world reinforcement learning via multifidelity simulators," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 655–671, 2015.
[8] M. Cutler and J. P. How, "Autonomous drifting using simulation-aided reinforcement learning," in *IEEE International Conference on Robotics & Automation*, 2016.
[9] W. Ding, S. Li, H. Qian, and Y. Chen, "Hierarchical reinforcement learning framework towards multi-agent navigation," in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2018, pp. 237–242.
[10] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots & Systems*, 2004.

[11] 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.

[12] E. Rohmer, S. P. N. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *IEEE/RSJ International Conference on Intelligent Robots & Systems*, 2013.

[13] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv preprint arXiv:1702.01182*, 2017.

[14] H. M. Le, Y. Yue, P. Carr, and P. Lucey, "Coordinated multi-agent imitation learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1995–2003.

[15] J. Song, H. Ren, D. Sadigh, and S. Ermon, "Multi-agent generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 7461–7472.

[16] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," in *Advances in neural information processing systems*, 2016, pp. 4026–4034.

[17] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7559–7566.

[18] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[19] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

[20] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3540–3549.