**ME-MADDPG: An efficient** **learning-based motion planning method for multiple agents in complex environments**

**Abstract:**

Developing efficient motion policies for multi-agents is a challenge in a decentralized dynamic situation, where each agent plans its own paths without knowing the policies of the other agents involved. This paper presents an efficient learning-based motion planning method for multi-agent systems. It adopts the framework of multi-agent deep deterministic policy gradient (MADDPG) to directly map partially observed information to motion commands for multiple agents. To improve the efficiency of MADDPG in sample utilization, so as to train more brilliant agents that can adapt to more complex environments, a strategy named mixing experience (ME) is introduced to MADDPG, and this has led to our proposed ME-MADDPG algorithm. The novel ME strategy can be embodied into three specific mechanisms: 1) An artificial potential field (APF) based sample generator to produce high-quality samples in the early training stage; 2) A dynamic mixed sampling strategy to mix the training data from different sources with a variable proportion; 3) A delayed learning skill to stabilize the training of the multiple agents. A series of experiments have been conducted to verify the performance of the proposed ME-MADDPG algorithm, and it has been demonstrated that, compared with MADDPG, the proposed algorithm can significantly improve the convergence speed and convergence effect in the training process, and it has also shown better efficiency and better adaptability in complex dynamic environments while it is used for multi-agent motion planning applications.

**Keywords:** multi-agent; deep reinforcement learning; motion planning; MADDPG

# 1 INTRODUCTION

Multi-agent motion planning (MAMP) for autonomous systems remains an open research challenge, and it has recently attracted much more interest along with some applications, such as drone formation shows, self-driving cars, intelligent warehouse storage, unmanned goods delivery, and UAV swarm operation. These applications are mainly concerned with finding safe and efficient trajectories for agents to move from an arbitrary departure location to the desired destination while avoiding obstacles or inter-agent collision1,2.

Despite various approaches that have been developed to solve motion planning problems in different situations, to navigate multiple agents move safely across a complex environment is still a challenging problem even with some static obstacles2-4. Typically, each agent makes its own decisions only based on the partially observed information from an onboard sensor and probably knows nothing about the policies and intents of other agents. The partial observability makes the sample-based approaches inapplicable 5-9, because it is difficult to sample satisfied policies for all the agents simultaneously within a large search space. When faced with a dynamic circumstance, the decision-making process becomes even more difficult. Uncertain surroundings and unpredicted moving obstacles make any pre-planned motion strategy unavailable. Some direct re-planned policies must be generated as soon as possible so that all the agents can adapt to the changing environment 10. It becomes intractable for some search-based approaches 11-16 because continuous online re-planning requires high timeliness that many search-based algorithms could not reach.

Recently, researchers have resorted to learning-based approaches for the MAMP problem. Reinforcement learning (RL)17, for instance, can transfer the computational cost of the online planning process to an offline training procedure, where policy functions are trained to encode cooperative behavior of the agents by trial-and-error interacting with different environments repeatedly 18. After that, the trained policy functions will be exploited in practical scenes to generate optimal moving policies for each agent dynamically. Since they try to optimize current policy by maximizing predicted long-term rewards, RL approaches turn out to be very adaptable to dynamic environment19,20. Among the RL-based motion planning methods, multi-agent deep deterministic policy gradient (MADDPG) is the most popular one that was proposed by Lowe et al. 21 to solve the MAMP problem in cooperative and competitive missions. MADDPG is an improvement of DDPG to adapt to a multi-agent environment. The core part is that the critic network of each agent is designed to obtain action information of all other agents, and it carries out centralized training and decentralized executing (CTDE). In MADDPG, each agent is trained with a global information-based critic network and a local information-based actor network, and each agent is allowed to have its own reward function so that it can be used in cooperative tasks or competitive tasks22,23. MADDPG has made great progress in MAMP applications compared with traditional RL approaches; However, because MADDPG comes from DDPG, it inevitably inherits some disadvantages of DDPG. Firstly, it uses a deterministic policy that is unfavorable for the exploring of action, and inefficient sample utilization makes it more difficult to train a convergent model. Secondly, as training an actor highly depends on the critic, it can easily lead to a bad actor because random sampling at the beginning stage is unlikely to support learning an accurate critic24-26. These shortcomings make MADDPG brittle with respect to environments and the number of agents, and small changes of the environment or small growth of the agent number may make MADDPG no longer applicable.

This paper is motivated by providing a stable and effective solution to the MAMP problems in complex environments, where agents have limited sensing and communication capabilities, and the surroundings are deployed with scatted moving obstacles. An improved deep reinforcement learning algorithm, Mixed Experience Multi Agent Deep Deterministic Policy Gradient (ME-MADDPG), is proposed. In this novel algorithm, a traditional Artificial Potential Field (APF) 14 method is introduced into the MADDPG to produce high-quality samples at the beginning of the training stage so that the Actor and Critic can quickly converge to a good state. And then, a mixed sampling mechanism with a dynamic probability is designed to balance random sampling and APF sampling in the following training stage. It ensures an increasing exploration of the unknown space, and the Actor and Critic will be constantly optimized in the exploration until they converge to a better state. At last, a delayed learning trick inspired by TD3 24 is introduced to the whole training stage. Since it avoids the repeatedly changing of strategic direction, this trick could stabilize the training to a certain extent. To verify the performance of the proposed ME-MADDPG algorithm, a multi-agent task environment has been constructed, and a series of experiments are conducted, and some valuable conclusions are drawn.

The remainder of this paper is organized as follows. The existing studies on MAMP problem is presented in Section 2. Section 3 introduces MAMP problem and formulate it as a Markov game model. Section 4 presents the core approach, ME-MADDPG, describes its structure and underlying principles, and elaborates its three tricks. Section 5 verifies the efficiency and adaptability of the ME-MADDPG algorithm for MAMP application through simulation experiments. Lastly, section 6 provides a comprehensive summary of this paper and presents directions for future work.

# 2 RELATED WORKS

In this Section, some existing work on MAMP problem will be reviewed, which roughly fall into two categories: planning-based methods and learning-based methods. As a class of traditional approaches, generally, planning-based methods can be further classified into sample-based and search-based.

1) *Sample-based planning methods*: This is an important type of method to solve motion planning problems with global constraints. By continuously sampling, this type of method firstly builds an undirected graph on a known map and then searches a relatively optimal path in graph 5. Among sample-based methods, rapidly-exploring random tree (RRT) 6 is the most representative one, and a series of improved algorithms based on RRT have been proposed to solve the motion planning problem. For example, Rahul et al.7 presented the RRT-Connect algorithm, which can provide better tree expansions and connectivity checks when it was used for multi-vehicle planning. In 8, Ragaglia et al. extended Poli-RRT\* to a multi-agent cooperative setting where multiple vehicles shared the same environment and needed to avoid each other besides some static obstacles. In order to improve the efficiency of cooperative pathfinding in dense environments, [Jiang](https://xueshu.baidu.com/s?wd=author%3A%28J%20Jiang%29%20&tn=SE_baiduxueshu_c1gjeupa&ie=utf-8&sc_f_para=sc_hilight%3Dperson) et al. 9 improved the MA-RRT\* by introducing the Potential Field method into the sampling process and have produced MA-RRT\*PF algorithms. The advantage of this type of method is that it can search high-dimensional space quickly without modeling the environment. It should be noted that, as a pure random algorithm, sample-based planning methods are sensitive to complex environments, where massive obstacles may decrease its efficiency, and some existing narrow channels may even directly turn the method invalid.

2) *Search-based planning method:* This type of method performs better practicability than sample-based planning methods when faced with complex environments. If the environment is known in advance, some heuristic search algorithms, such as A\*10, D\*12, LPA\*13, or artificial potential field (APF) method 14 can be directly used to generate optimal paths for multiple agents. If the agents know nothing about the environment, a simultaneous localization and mapping (SLAM) method can be utilized to construct a map of the environment first, and heuristic search algorithms or APF method is then adopted to produce proper paths. In 15, a SLAM-based feature map using an ultrasonic sensor was proposed to solve the environment adaptation problem in autonomous driving, and a modified genetic algorithm was employed to optimize path-planning. [Alzugaray](https://xueshu.baidu.com/s?wd=author%3A%28I%20Alzugaray%29%20&tn=SE_baiduxueshu_c1gjeupa&ie=utf-8&sc_f_para=sc_hilight%3Dperson) et al.16 proposed a path-planning algorithm designed to work in the loop of the SLAM estimation of a monocular-inertial system, and the algorithm was demonstrated to work well while navigating a small UAV in an unknown environment using the incrementally generated SLAM map. Obviously, the search-based planning method relies on an environment model, and an inaccurate model may result in poor performance. In addition, this type of method adopts an open-loop mechanism that makes decisions without any prediction and reasoning of the future, which blocks its adaptation to dynamic environments.

3) *Learning-based method*. With the rapid development of machine learning techniques, learning-based methods emerge and gradually become a new favorite. Among this type of method, RLs have attracted the greatest attraction as they do not require large training data sets; instead, they learn by trial-and-error and decide by maximizing predicted long-term rewards 30. Chen et al. 27,28 used a deep network to approximate the value function of RL and learned a cooperative critic for agents to select expected times to reach their goals from initial states. However, the setting is oversimplified that the motion parameters of the agent were not optimized. Long et al.29 presented a decentralized, end-to-end approach for multi-robot systems based on DRL, which directly mapped raw sensor measurements to an agent’s steering commands. However, without any agent-level information, it cannot learn the cooperation and competition relationship between the multiple agents. MADDPG 21 is a representative method proposed by Lowe et al. that uses a critic-actor framework to encode the cooperative behavior of the agents. Its centralized training and decentralized executing scheme make large-scale agent applications possible. Theoretically, once the offline training is done, the deployed agent is allowed to quickly select its own action without knowing any other agents’ actions. However, just like other RL algorithms, MADDPG suffers from sample inefficiency, especially at the beginning of the training stage25. This also leads to MADDPG being too sensitive to a dynamic environment and the number of agents. To improve the sample efficiency, Samaneh et al.30 attempted to introduce pre-supervised learning (SL) step before the reinforcement learning, and an FMP method is introduced to develop a hybrid approach. Inspired by this, the APF method and the MADDPG method are combined in this research, and the ME-MADDPG approach is proposed. In the following Sections, more explanation will be given in detail on this novel method.

# 3 PROBLEM FORMULATION

## 3.1 Multi-agent motion planning problem

In our MAMP scenario, a series of agents have to move across an unknown area that is scattered with some moving obstacles until assigned targets are finally reached. As shown in Figure 1, agent *i* (represented by the circle with a blue border) is deployed at the location and it has to move across a limited space to target (represented by the circle filled with blue color) without a collision on any of the other two agents (agent *j* and agent *k*) and four moving obstacles (represented by circles with a black border). For agent *j* and agent *k,* they have the same mission. The obstacles are moving irregularly, and each agent can perceive the possible obstacle threats ahead in the detection range with a LiDAR or other sensor onboard. The agents are required to reach the assigned targets as soon as possible. They can try cooperative strategies to minimize the total time spent.



**FIGURE 1**. The multi-agent motion planning scenario, where the circle with colored border represents a different agent, and the arrow on it indicates the motion direction, and the circle filled with pure color is the destination of the corresponded agent, and the solid colorful line connecting two circles indicates the agent’s trajectory, and the circles with black border indicate moving obstacles, and the dashed line begins from the obstacle is regarded as its random trajectory.

In a MAMP mission, all the agents are required to reach their respective targets as soon as possible, but no collision is allowed, i.e., the objective of the mission is to minimize the total arrival time with target-reaching constraints and collision-free constraints as

|  |  |  |
| --- | --- | --- |
|  |  | (1) |
|  |  | (2) |
|  |  | (3) |

where denote the positions of agent , target and obstacle at time , respectively. represents the time of agent takes to reach target . is the number of the agents, is the number of the obstacles, and is the time spent for completing all tasks. is the safe distance between two agents while denotes the safe distance between an agent and an obstacle. Eq. (2) specifies the requirements that all agents must reach the target points. Eq. (3) defines the collision-free constraints, where the first one means any two agents cannot collide at all times, while the second defines an agent and any obstacle cannot collide at any time. in Eq. (1) is the decision variable that represents the policy set of all agents at all times.

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

where  is the motion policy of agent at time , and . In the following Sections, the specific definition of  will be given, and then a solution for calculating  will be introduced.

## 3.2 Multi-agent motion model

To simplify the MAMP problem, a two-dimensional setting is allowed in the paper. Taking one agent as an example, the motion model can be established based on the work by Lawrence et al. 31.



**FIGURE 2.** Motion model of agents and obstacles, where ,, are the radii of agent *i*, target *i*, and obstacle *k*. These three radii are used for collision detection. are the linear velocity, the heading angle, and the target velocity angle of agent *i*, respectively. are the linear velocity and the heading angle of obstacle *k*. denote the positions of them. represents the radius of the agent’s sensing area.

As illustrated in Figure 2, is set as the radius of agent , as the radius of its target, and as the radius of obstacle *k*, respectively, these three radii are used for collision detection. A collision happens whenever the distance between agent *i* and obstacle *k* is less than , or the distance between agent *i* and agent *j* is less than . When agent *i* arrives at target *i*, becomes less than . Suppose the agent is able to sense a circular area with a radius of , whoever enters this area, its position information will be perceived by the agent. To model the agent’s motion, let denote the position of agent , and indicate its linear velocity, denote the heading angle, and represent the target velocity angle, i.e., the angle between the agent’s velocity vector and the target line vector, respectively. and are limited within is defined as the angular velocity of agent . Consider a unicycle kinematics model, the agent motion can be expressed as:

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

where and construct the control variable of agent *i,* i.e., *.*

## 3.3 Multi-agent planning model

MAMP is a typical sequential decision problem, and it can therefore be modeled by stochastic games (also known as Markov games) 32, which is usually described by the five-tuple . In this tuple, represents a collection of *N* agents; , where denotes the state of agent . ***S*** is joint state, which can be described by the Cartesian product of the states of all the agents’; , where is the action of agent . is joint action, which can be described by the Cartesian product of the actions of all the agents’; is the state transition model, which represents the probability that joint state transfers from to after executing joint action ; , where represents the reward of agent obtained by interacting with the environment. is joint reward, which can be described by the Cartesian product of the rewards of all the agents. Before modeling the MAMP problem as a Markov game, let’s define the joint state, the joint action, and the joint reward first.

### 3.3.1 Joint state and action

The joint state is defined as . For agent , its state is defined as , where is a collection of the position , the speed , and the radius of agent ; is the environmental information observed by agent , where the position of target and the state information of the detected agent or obstacle are included. is the state space of agent in the absolute coordinate system. Considering that the absolute positions cannot be obtained directly in many practical applications, the polar coordinate system can be used as an alternative, and as such, the state of agent is defined as , where is the distance between agent and its target,,, and are the target velocity angle, the linear velocity, the heading angle, and the radius of agent, respectively. are the relative distance, the linear velocity, the heading angle, and the radius of the nearest agent or obstacle within the sensing range of agent . If agent does not detect any agent or obstacle, then is set to .

The joint action is defined as , where . is the linear velocity and is the angular velocity of agent . In this paper, the agent is set to move at a constant speed, and then the action of agent is reduced to .

### 3.3.2 Reward function

Rewards are the only feedback signals that can be used for the agent’s learning. A well-shaped reward function usually contains as much human experience information as possible that will provide the agent with better learning performance33,34. The reward function is usually set for an agent to achieve its goal as easily as possible. In this paper, the goal for an agent to achieve is to get to its assigned target as soon as possible without any collision. Therefore, a positive reward can be given if the agent successfully arrives at its target. However, in cases of arriving at a target, different policies can lead to different time consumptions. Hence, a time-dependent punishment can be introduced into the reward function, in which a longer time consumed means a greater punishment. Consequently, the reward term in arriving case for agent *i* can be defined as:

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

where 5 is the reward for arriving, and the  represents the penalty with regard to the consumed time . scales the degree of the penalty and is set to 0.1 in this paper.  is defined as the time spent when agent *i* moves directly to the target at a uniform speed. It is the minimum time agent *i* may spend moving to the target. When a collision happens, a big punishment would need to be applied to the reward function, and in this paper, it is set to:

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

To avoid sparse reward, when an agent moves in an intermediate state, i.e., neither reaches the target location nor collides, a non-sparse reward is introduced based on the moving trend of the agent. To do so, four principles need to be obeyed: 1) The closer to a target, the greater the benefit; 2) The greater the deviation from a target, the greater the penalty; 3) When the agent is approaching the target, a reward given, when the agent is moving away from the target, a punishment fall; and 4) The closer to another agent or an obstacle, the greater the punishment. These four principles can be formulated as:

|  |  |  |
| --- | --- | --- |
|  |  | (8) |
|  |  | (9) |
|  |  | (10) |
|  |  | (11) |

In Eq. (8), is the distance between agent to its target at start time, and represents their distance at the current time. The larger the , the smaller the . This principle will attract agent to move as close to its target as possible. In Eq. (9), target velocity angle is used to represent the deviation from the target. Obviously, a larger leads to a smaller , which will drive agent to adjust its direction towards its target as much as possible. In Eq. (10), and are the distance between agent and its target at the current time and the next time, respectively. A larger  results in punishment to and a smaller results in a reward to . Principle 3 will make the agent try to move approaching to the target. In Eq. (11), is the sense distance of agent , and is the distance between agent and the nearest agent or obstacle within the sensing range. When agent detects an object, a punishment occurs, and the closer the detected object, the larger the punishment. If agent doesn’t detect an object, there is no punishment. This principle will teach the agent to stay away from collisions as far as possible. In summary, the reward function that when agent *i* moves normally can be weighted as:

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

where is a normalization function for the reward, and is necessary to avoid failures caused by the magnitude difference among the four reward terms. here is used to map the reward of Eqs. (8), (9), (10), and (11) to the range of -10 to 10. are set to represent the contributions of the four terms to the final reward, and they satisfy the condition . Ultimately, the reward function of agent can be summarized as:

|  |  |  |
| --- | --- | --- |
|  |  | (13) |

### 3.3.3 Markov games model



**FIGURE 3.** The Markov games model of MAMP, where , andare the joint state, the joint reward, the joint action of all *N* agents at time *t*

After the joint state, the joint action, and the reward function have been defined, the Markov-based MAMP model can be described as figure 3: at each epoch , agent collects the joint state , and selects its optimal action  by maximizing the joint cumulative return of all the agents. When all the agents’ actions are selected, and the actions are used to control the agent’s movement, the states are changed to  , and agent receives a reward . For agent , the cumulative return is defined as a discount return function：

|  |  |  |
| --- | --- | --- |
|  |  | (14) |

where denotes discount factor and . is the reward of agent at time . The policy 𝜋 is defined to map the joint state to a probability distribution over the joint action . At arbitrarily time *,* 𝑄-function is defined to describe thefuture cumulative return when performing an action  at state as:

|  |  |  |
| --- | --- | --- |
|  |  | (15) |

For agent , it can select its optimal action by solving the following 17:

|  |  |  |
| --- | --- | --- |
|  |  | (16) |

Usually, it is difficult to solve Eq. (16) directly for the curse of dimensionality of the Q-value calculation in Eq. (15). In the learning-based planning methods, some neural networks are adopted to approximate the function of and, and by these approximations, the intractable planning problem is translated into a neural network training problem. In Section 4, we will discuss the proposed learning-based algorithm for solving the problem in Eq. (15) and Eq. (16).

# 4 MIXED EXPERIENCE MADDPG FOR MAMP

In this Section, the proposed novel algorithm, ME-MADDPG, is presented in detail.

## 4.1 MADDPG approach

As a basis of our algorithm, the MADDPG algorithm needs to be discussed first. The core idea of MADDPG is a centralized training and a decentralized execution, i.e., the agent uses its own state to produce action and uses the joint state to estimate the *Q*-function. The structure of MADDPG is shown in Figure 4.

From its structure, it can be seen that MADDPG uses an Actor-Critic framework. When the number of agents is , there are corresponding Actor-Critics. The Actor realizes policy estimation, while the Critic makes Q-value Estimation, and the replay buffer contains the joint state, joint action, and joint reward by interacting with the environment. At time , Actor firstly produces an action based on its own state, and then the action is executed, and a reward will be received by agent *i* all the other agents carry out the same procedure. After that, the states, the actions, and the rewards of all the agents are collected to be used to evaluate the *Q-*value through Critic of each agent; In this process, with a continuous accumulation of samples, the *N* Critic-networks and *N* Actor-networkswill be trained periodically through gradient descent method.



**FIGURE 4.** The structure of MADDPG, where *N* coupled Critics, and *N* independent Actors are included.

There are at least two drawbacks when MADDPG is used to solve a MAMP problem: 1) Since the agent explores the environment in a random manner, the probability of the agent reaching the target is extremely small in the early stage, especially when the environment is scatted with many moving obstacles and other moving agents. When the replay buffer is full of experiences of collisions, it is really hard to train multiple smart agents by these bad experiences. Some high-quality positive feedback experiences are crucial for the initial training, especially in a multi-agent system. 2) MADDPG comes from DDPG, which adopts a direct updating scheme in the learning process that the parameters of both the Critic and the Actor are updated at each time step. This direct updating scheme changes the strategic direction of the policy too frequently, which in return confuses the agent in policy learning. As a result, it often leads to unstable agents 36.

## 4.2 ME-MADDPG approach

### 4.2.1 Framework of ME-MADDPG



**FIGURE 5.** Thestructure of ME-MADDPG, where an additional sampling module (the blue rectangle entitled Artificial Potential Field) and an additional experience pool (the blue rounded rectangle entitled replay buffer 1) are integrated into original MADDPG

ME-MADDPG still uses an Actor-Critic framework, as illustrated in Figure 5. Compared with MADDPG, ME-MADDPG introduces an APF Module into the framework and a new replay buffer is added to store the samples from APF. In planning, an agent uses the APF to produce actions with a probability , and uses its Actor to produce actions with a probability , where is defined as the mixed sampling probability that will gradually decrease from a large initial value. A larger initial ensures APF to be utilized to generate actions as much as possible at the beginning stage so that more high-quality samples are produced before Actor is trained to be smart enough. The actions generated by APF will be stored in Replay buffer 1, i.e., , while actions generated by Actor be stored in Replay buffer 2, i.e., . Both the and will be used for training. Before training begins, a batch size of samples has to be prepared first. Unlike MADDPG that can only sample from a single replay buffer, ME-MADDPG can conduct sampling from both and at the same time, and then the samples will be mixed according to the probability . This mechanism ensures that some high-quality samples can be used for training at the initial stage, and as the training goes deeper, a smaller probability will bring more explorations to the agents. It is a good mechanism that trades off exploration against exploitation.

### 4.2.2 APF for ME-MADDPG

Consider an *N* agents application. Let be the set of all agent policies parameterized by . Each agent aims to maximize its own total expected return:

|  |  |  |
| --- | --- | --- |
|  |  | (17) |

and the gradient of the expected return for agent can be written as:

|  |  |  |
| --- | --- | --- |
|  |  | (18) |

where  is the joint state at the current time;  represents a policy network with parameter ; and  is a centralized action-value function that takes the joint state and joint action as input, and outputs the Q-value for agent . is the replay buffer, which includes and , and each sample can be organized as , where  represents the joint state of the agents at next time. In , the action of agent is produced through the policy network, that is, ; while, in , the action of agent is generated by APF 2. All the states and rewards are obtained by interacting with the environment.

In APF, the normalized attractive force of agent at to target at is defined as:

 (19)

The normalized repulsive force of agent at to another agent or obstacle at is defined as:

 (20)

The resultant force  on agent can be calculated by combining the attraction and repulsion as:

|  |  |  |
| --- | --- | --- |
|  |  | (21) |

where represents the collection of neighboring objects of agent , and is the collision function, which represents the degree of influence of each neighboring object on the repulsion of agent . is defined by:

|  |  |  |
| --- | --- | --- |
|  |  | (22) |

where is the Euclidean distance between agent and neighboring object , and is the minimum collision distance between agent and the neighboring object. and are constants within , and usually . The calculations of parameters are as follows:



By the resultant force of agent , we obtain its velocity control policy. Since the line velocity of agent is fixed in this paper, we only need to find its angular velocity .

|  |  |  |
| --- | --- | --- |
|  |  | (23) |

where is the angular velocity control constant, represents the heading angle,  represents the angular direction of the resultant force , and represents the derivative of with respect to time at the position. can be calculated by.

|  |  |  |
| --- | --- | --- |
|  |  | (24) |

Eqs. (23) and (24) can be used to calculate the angular velocity of agent at time . The angular is used to control the agent’s movement.

In DRL,  is defined as the action-value function of agent , and needs to be estimated and calculated based on the joint action and joint state of the replay buffer .  is updated as:

|  |  |  |
| --- | --- | --- |
|  |  | (25) |

where , and  represents the target policy network with delayed update parameters .  denotes the parameter of the *Q*-network of agent , and  represents the parameters of the target *Q*-network. In Eq. (25), is the sum of the immediate reward of agent , and the target *Q*-value where the target *Q*-value is given by the target *Q*-network, and the action selection in the next state is determined by the target policy network according to the joint state. The cost function  is the mean squared error of the output of the *Q*-network and . Finally,  can be updated by minimizing , and then  can be updated by using gradient descent algorithm. The parameters of the target policy network and the target *Q*-network are updated as follows:

|  |  |  |
| --- | --- | --- |
|  |  | (26) |
|  |  | (27) |

### 4.2.3 Learning of ME-MADDPG

In this Section, the learning techniques for ME-MADDPG is presented based on the model described in Section 4.2.2., and a delayed learning skill is also introduced to stabilize the training.

(1) Experience generation method

Compared with MADDPG, ME-MADDPG can generate actions based on not only the Actor, but also the APF model. Therefore, there are two replay buffers and . Both and contain experiences with the joint state, joint action, joint reward, and the next joint state. The difference is that the experiences in is generated depending on the Actors of all agents and the experiences in come from the APF model. It has been proved that the APF model can output current optimal solutions at any discrete-time, but the long-term optimality of these policies cannot be guaranteed. That is, APF provides myopic policies. On the contrary, RL provides non-myopic policies. Since the learning process is guided by maximizing a long-term cumulative return, if it is properly trained, the Actor in RL will produce high-quality policies that are adaptable to a complex environment. However, the Actor is faced with a dilemma that it is difficult to converge quickly. In the early stage of training, the Actor can only produce some low-quality samples, which can make the training more difficult. In this paper, an experience generation mechanism is designed that makes full use of the advantages of APF and Actor. The mechanism is shown in Algorithm 1.

|  |  |
| --- | --- |
| Algorithm 1: Experience generation method | |
| 1: | Initialize probability, replay buffer and |
| 2: | **for**  **do** |
| 3: | Initialize noiseand joint state, generate a random value |
| 4: | **while** not all agents arrive **and** not all agents collide **and**  **do** |
| 5: | **if**  **do** |
| 6: | for all the agents, select actions by the artificial potential field method |
| 7: | Execute actions , and calculate reward , obtain new state |
| 8: | Store  in the replay buffer |
| 9: | **else** **do** |
| 10: | for all the agents, select actions by exploration or their own Actors |
| 11: | Execute actions , calculate reward , obtain new state |
| 12: | Store  in the replay buffer |
| 13: | **end if** |
| 14: | **end while** |
| 15: | **end for** |

(2) Mixed sample collection strategy

In MADDPG, a batch of experiences are sampled randomly from the replay buffer for training. As mentioned previously, owns little experience about how to successfully reach the target in the early stage of training. This makes the preliminary training inefficient. To address this problem, a sample collection strategy is proposed based on dynamically mixing the experiences sampled from and . The sample mixing ratio is determined by probability , and the mixed sample collection strategy is shown in Algorithm 2. In order to continuously provide good mixed samples, the probability is designed to change dynamically. In the early episodes, probability is set to a fixed value. When the number of experiences collected by the agents reaches the upper limit of the capacity of the experience replay buffer, the training of the Actors and Critics begins. The probability will gradually decrease until it reaches 0 as the number of training increases. The decrease of reduces the experiences sampled from , and increase the experiences sampled from . In this way, it can ensure that when the agents become smarter, they will gradually increase their explorations of the environment, and through exploration, they will learn new skills to help adapt to complex environments. The probability adjustment strategy is shown in Algorithm 3.

|  |  |
| --- | --- |
| Algorithm 2: Mixed sample collection strategy | |
| 1: | **ProbabilitySample** |
| 2: |  |
| 3: | Sample a random minibatch of  samples  from |
| 4: | Sample a random minibatch of  samples  from |
| 5: | Randomly combine two sets of samples intosamples |
| 6: | **Return** |

|  |  |
| --- | --- |
| Algorithm 3: Probability adjustment strategy | |
| 1: | **ProbabilityChange** |
| 2: |  |
| 3: | **if**  **and**  **do** |
| 4: |  |
| 5: | **else do** |
| 6: |  |
| 7: | **end if** |
| 8: | **Return** |

(3) Delayed learning skill

MADDPG uses a direct updating scheme in training; that is, the network parameters of the Actor and Critic are updated at each time step. This scheme will frequently change the strategic direction of the policy, causing the agent to become lost in policy learning. This may lead to unstable agents, and policy jittering occasionally occurs while exploiting it in practical applications 36. To handle this, a delayed learning skill is introduced, in which the Actor and critic network learning operations are delayed to the end of each episode. This way ensures the Actor and the Critic obey the same principle in an ongoing episode and avoids the repeatedly changing of strategic direction. Besides, a fixed interval is set for the soft updates of the target networks. The delayed learning is shown in Algorithm 4.

|  |  |
| --- | --- |
| Algorithm 4: Delayed learning | |
| 1: | **for**  **do** |
| 2: | Set action probability**ProbabilityChange** |
| 3: | **for**  **do** |
| 4: | **for** agent  **do** |
| 5: | Sample **ProbabilitySample** |
| 6: | Update critic by minimizing the loss in Formula (25) |
| 7: | Update actor using the sampled policy gradient in Formula (18) |
| 8: | **end** **for** |
| 9: | **if** target update interval reaches, **do** |
| 10: | Soft update target network parameters for the agent |
| 11: | **end if** |
| 12: | **end** **for** |
| 13: | **end for** |

Combining the three learning techniques above, we get the pseudocode of ME-MADDPG algorithm in Algorithm 5.

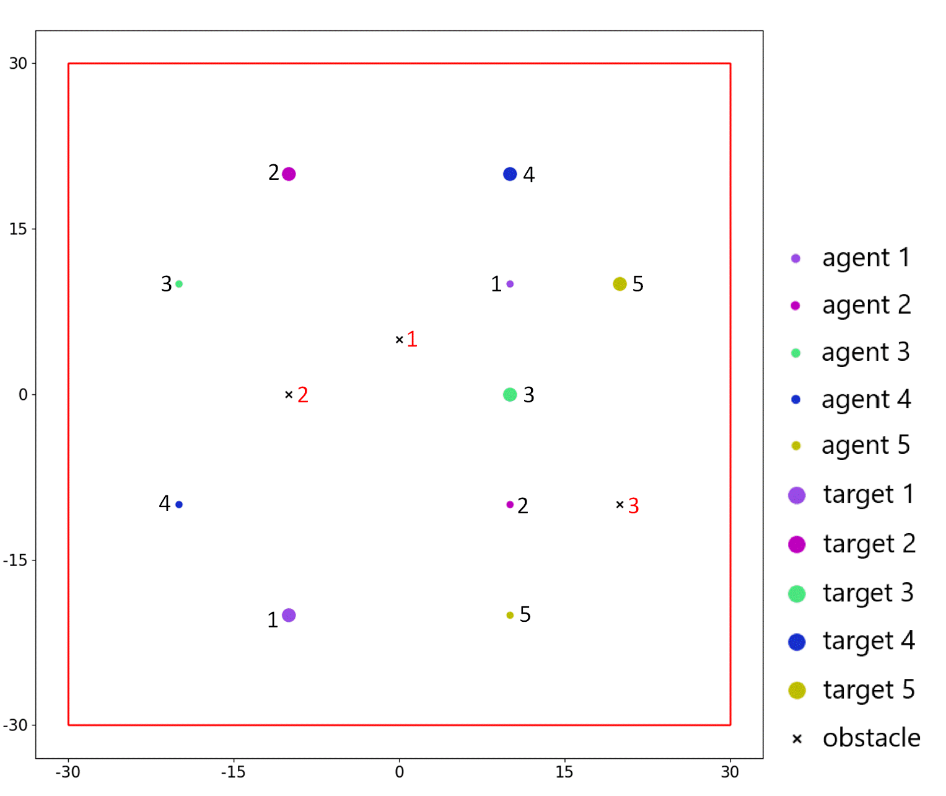
|  |  |
| --- | --- |
| Algorithm 5: ME-MADDPG for n agents | |
| 1: | Mixing probability; double replay buffer; batch size; target update interval |
| 2: | **for**  **do** |
| 3: | Initialize noiseand union state , generate a random value |
| 4: | Set action probability**ProbabilityChange** |
| 5: | **while** not all agents arrive **and** not all agents collide **and**  **do** |
| 6: | **if**  **do** |
| 7: | Calculate artificial potential field |
| 8: | for agent , select action  by |
| 9: | Execute actions , reward  , and new state |
| 10: | Store  in the replay buffer |
| 11: | **else** **do** |
| 12: | for agent , select action |
| 13: | Execute actions , reward  , and new state |
| 14: | Store  in the replay buffer |
| 15: | **end if** |
| 16: | , |
| 17: | **end while** |
| 18: | **for**  **do** |
| 19: | **if**  **do** |
| 20: | **for** agent  **do** |
| 21: | Sample **ProbabilitySample** |
| 22: | Set |
| 23: | Update critic by minimizing the loss: |
| 24: | Update actor using the sampled policy gradient: |
| 25: | **end for** |
| 26: | **if**  **do** |
| 27: | Update target network parameters for the agent : |
| 28 | **end if** |
| 29: | **end if** |
| 30: | **end for** |
| 31: | **end for** |

# 5 EXPERIMENTS AND DISCUSSIONS

This Section presents a series of experiments to verify the efficient performance of the ME-MADDPG by applying it to solving the MAMP problem. These experiments are divided into three categories: training experiments, exploiting experiments, and statistical experiments.

## 5.1 Experimental settings

For training and testing of the ME-MADDPG, an experiment environment was built and it can be used to simulate different multi-agent missions as depicted in Figure 6. Some agents (indicated by the small numbered, colored circles), some targets (indicated by the big numbered, colored circles), and some obstacles (represented by the numbered black crosses) are deployed in a mission area (the red square) with a side length of 60m. The agents represent some intelligent vehicles with a safe radius of , and each agent has an onboard sensor that can detect a circular area with a radius . The linear velocity of the agents was set as , and their angular velocity was bounded within . Each agent was assigned to a fixed target indicated with the same color as the agent. The target’s radius was set to . The obstacles were randomly or regularly deployed in the mission area with a radius . They can be set as static or mobile entities. In a mobile mode, their linear velocity has been randomly selected within the range of , and their angular velocity was randomly selected within the range of . They moved randomly without any navigating law. The number of obstacles can be adjusted according to mission requirements. It should be noted that the motion parameters of the agents were all set corresponding to the size of the mission area, and just represented an abstraction of real-world application.



**FIGURE 6.** The experiment environment with five agents, five targets, and three obstacles, where the smaller numbered colorful circles are agents, the bigger numbered colorful circles are targets, and numbered black crosses are obstacles

As for parameters of the algorithm, ME-MADDPG used 2*N* neural networks to construct an *N*-Agent-Actor-Critic structure. In this paper, we considered the case of 5 agents, i.e., *N* = 5. Then, each Actor owned a network structure of , where the 9 input nodes were the agent state, , and the 1 output node represented the action for agent . The Critic is constructed by a fully connected neural network, where the 50 input nodes were the state and action combination of the 5 agents, i.e.,, and the 1 output node represented the evaluated *Q*-value. Both the Actor and Critic use as the activation function. Adam optimizer is employed to learn network parameters. All the hyperparameters were selected as:

**Table 1.** The hyper-parameters of ME-MADDPG

|  |  |  |
| --- | --- | --- |
| No. | Hyper-parameters | Values |
| 1 | actor network structure |  |
| 2 | critic network structure |  |
| 3 | discount factor, | 0.98 |
| 4 | batch size, | 128 |
| 5 | double replay buffer, | 50000,50000 |
| 6 | the actor learning rate | 0.001 |
| 7 | the critic learning rate | 0.0001 |
| 8 | the maximum episode length, | 250s |
| 9 | the maximum episode number, | 20000 |
| 10 | moving interval, | 0.5s |
| 11 | reward function weights, | [0.15,0.3,0.3,0.25] |
| 12 | sampling probability, | 0.5 |
| 13 | reduction rate of sampling probability | 1/20000 |
| 14 | target network update interval, | 200 |

Most hyperparameters of ME-MADDPG are the same as MADDPG, and their values were selected by performing repeated preliminary parameter-tuning experiments based on the training experience of MADDPG. Besides, three additional parameters were introduced into ME-MADDPG compared with MADDPG, the sampling probability , the reduction rate of sampling probability and the target network update interval (corresponding to rows 12, 13, 14 in Table 1.). To eliminate the interference of these three parameters on the conclusion of the performance of ME-MADDPG, we determine their values by performing a series of grid search experiments. Please refer to Table 2 for more details, where only partial results of grid search are shown. After weighing, the values of No.1 group were finally selected.

Table 2. Grid search results of the additional hyperparameters in ME-MADDPG.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. |  |  |  | Arrival rate | | |
| No obstacles | 5 static obstacles | 5 motion obstacles |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

## 5.2 Training experiments

Before exploiting in MAMP application, ME-MADDPG should be trained first. To verify the advantages of ME-MADDPG in terms of convergence speed and convergence effect, some training experiments have been conducted, in which MADDPG was introduced for comparison purposes. MADDPG was implemented with the same hyper-parameter settings as ME-MADDPG. The two algorithms have been trained with the same dynamic mission environment, as depicted in Figure 6. Each experiment had 20000 episodes and, in each episode, the agents, targets, and obstacles were randomly re-deployed throughout the mission area.

To measure the performance, two indicators have been used and recorded: arrival rate and average reward. The arrival rate is defined as the ratio of agents who successfully reaches its targets to all the agents. In each episode, an arrival rate was calculated, and in every 100 episodes, the average arrival rate was counted. The average reward is defined as the average value of the rewards obtained by all the agents, and in every 100 episodes, the average reward has been counted and updated. The final training results are illustrated in Figure 7, where the statistical trends of the average arrival rate and the average reward with respect to the training episodes are counted and displayed in Figure 7(a) and Figure 7(b), respectively. Since OU noise was added to help the agent to explore the actions during training, the curves appear fluctuations. Note that in the figures, the solid lines represent the statistical means, while the shaded areas along the solid lines indicate statistical upper and lower limits of the means. The statistical data about arrival rate and average reward for ME-MADDPG and MADDPG is shown in Table 3.

|  |  |
| --- | --- |
| (a) | (b) |

**FIGURE 7.** Convergence curves of ME-MADDPG and MADDPG. **(a)** Statistical trends and of the average arrival rate with respect to the training episodes; **(b)** Statistical trends of the average reward with respect to the training episodes. The solid lines are the statistical means, while the shaded areas that wrap the solid lines are statistical upper and lower limits of the means.

**Table 3.** The statistical of arrival rate and average reward for ME-MADDPG and MADDPG.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | In episode 5000 | | Last 15000 episodes | |
| Arrival rate | Average reward | Mean arrival rate | Mean average reward |
| MADDPG | 71.63% | 1.8016 | 82.71% | 2.3390 |
| ME-MADDPG | **81.64%** | **2.2344** | **88.46%** | **2.5959** |

From Figure 7, it can be seen that: MADDPG has converged after about 7000 episodes while ME-MADDPG used about 5000 episodes to converge, i.e., ME-MADDPG has contributed 2000 episodes’ acceleration for the training. Additionally, ME-MADDPG converged to an arrival rate of 81.64% and an average reward 2.2344 in episode 5000, while the counterpart of MADDPG was 71.63% and 1.8016, respectively. Compared to MADDPG, ME-MADDPG has led to an increase of 14.1% () in the arrival rate and an increase of 24.02% () in the average reward. After both the two algorithms converged, they continued training until 20000 episodes. In the last 15000 episodes, the mean arrival rate and the mean average reward were further counted of the two algorithms. The relevant statistics can be found in Table 3, where (88.46%,2.5959) for ME-MADDPG and (82.71%,2.3390) for MADDPG. An increase of 6.95% for the mean arrival rate and an increase of 10.98% for the mean average reward has been gained by ME-MADDPG comparing with MADDPG. From the training experiments, it is evident that ME-MADDPG can outperform MADDPG with a faster convergence speed and a better convergence effect, and that is mainly due to the efficient data utilization and the stable policy update of the proposed ME strategy. In the following experiments, verifying the performance of ME-MADDPG is continued from other different aspects.

## 5.3 Exploiting experiments

After training, two motion planners based on ME-MADDPG and MADDPG were constructed. To test the application efficiency of ME-MADDPG, a series of exploiting experiments were conducted, in which the trained planners were used for controlling multiple agents' move in different environments.

(1) Experiment scenarios

Four specific MAMP missions have been considered as shown in Figure 8, where Figure 8(a) presents a mission with 5 agents, 5 targets and 20 regularly deployed obstacles, Figure 8(b) is a mission with 5 agents, 5 targets and 20 randomly deployed obstacles, Figure 8(c) shows a mission with 5 agents, 5 targets and 36 regularly deployed obstacles, Figure 8(d) shows a mission with 5 agents, 5 targets and 36 randomly deployed obstacles. Each mission includes two scenarios, one is with static obstacles, and the other is with mobile obstacles. The initial velocity directions of the agents and obstacles were randomly generated. Table 4 lists all the experiment scenarios.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| (a) | (b) |  |
|  |  |  |
| (c) | (d) |  |

**FIGURE 8.** Different missions with 5 agents, 5 targets and 20 or 36 obstacles. The agents are required to move from to , respectively. (a) Mission with 20 regularly deployed obstacles; (b) Mission with 20 randomly deployed obstacles; (c) Mission with 36 regularly deployed obstacles; (d) Mission with 36 randomly deployed obstacles.

**Table 4.** Exploiting experiment scenarios

|  |  |  |  |
| --- | --- | --- | --- |
| Exp No. | Obstacle settings | | |
| Number | Static vs. Mobile | Random vs. Regular |
| 1 | 20 | Static | Regular |
| 2 | 20 | Mobile | Regular |
| 3 | 20 | Static | Random |
| 4 | 20 | Mobile | Random |
| 5 | 36 | Static | Regular |
| 6 | 36 | Static | Random |

(2) Experiment results

According to the above experimental scenario settings, the relevant exploiting experiments were undertaken one by one, and results are presented in Figures 9-14 and Tables 5-10.

A. Exp1: 20 static regularly deployed obstacles

|  |  |  |
| --- | --- | --- |
| (a) MADDPG | (b) MADDPG | (c) MADDPG |
| (d) ME-MADDPG | (e) ME-MADDPG | (f) ME-MADDPG |

**FIGURE 9.** Trajectories of 5 agents driven by MADDPG and ME-MADDPG in the scenario with 20 static regularly deployed obstacles. (a), (b) and (c) are the trajectories of MADDPG at different times, while (d), (e), and (f) are the trajectories of ME-MADDPG at different times.

**Table 5.** Movement time and total path lengths of the 5 agents when driven by MADDPG and ME-MADDPG in the scenario with 20 static regularly deployed obstacles.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | MADDPG | | | | | ME-MADDPG | | | | |
| Agent number | 2 | 4 | 1 | 5 | 3 | 4 | 1 | 2 | 3 | 5 |
| Movement time (s) | 50.5 | 52 | 53 | 54.5 | **55** | 43 | 52.5 | 52.5 | 57 | **57** |
| Total path length (m) | **265** | | | | | **262** | | | | |

B. Exp2: 20 mobile regularly deployed obstacles

|  |  |  |
| --- | --- | --- |
| (a) MADDPG | (b) MADDPG | (c) MADDPG |
| (d) ME-MADDPG | (e) ME-MADDPG | (f) ME-MADDPG |

**FIGURE 10.** Trajectories of 5 agents driven by MADDPG and ME-MADDPG in the scenario with 20 mobile regularly deployed obstacles. (a), (b) and (c) are the trajectories of MADDPG at different times, while (d), (e), and (f) are the trajectories of ME-MADDPG at different times.

**Table 6.** Movement time and total path lengths of the 5 agents when driven by MADDPG and ME-MADDPG in the scenario with 20 mobile regularly deployed obstacles.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | MADDPG | | | | | ME-MADDPG | | | | |
| Agent number | 2 | 5 | 1 | 4 | 3 | 4 | 5 | 2 | 1 | 3 |
| Movement time (s) | 71.5 | 78.5 | 83.5 | 83.5 | **89** | 43 | 47.5 | 50.5 | 54.5 | **58.5** |
| Total path length (m) | **406** | | | | | **254** | | | | |

C. Exp3: 20 static randomly deployed obstacles

|  |  |  |
| --- | --- | --- |
| (a) MADDPG | (b) MADDPG | (c) MADDPG |
| (d) ME-MADDPG | (e) ME-MADDPG | (f)ME-MADDPG |

**FIGURE 11.** Trajectories of 5 agents driven by MADDPG and ME-MADDPG in the scenario with 20 static randomly deployed obstacles. (a), (b) and (c) are the trajectories of MADDPG at different times, while (d), (e), and (f) are the trajectories of ME-MADDPG at different times.

**Table 7.** Movement time and total path lengths of the 5 agents when driven by MADDPG and ME-MADDPG in the scenario with 20 static randomly deployed obstacles.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | MADDPG | | | | | ME-MADDPG | | | | |
| Agent number | 3 | 2 | 5 | 1 | 4 | 3 | 5 | 1 | 2 | 4 |
| Movement time (s) | 43 | 45.5 | 48 | 55 | **63** | 33.5 | 34.5 | 37.5 | 41.5 | **49.5** |
| Total path length (m) | **224.5** | | | | | **196.5** | | | | |

D. Exp4: 20 mobile randomly deployed obstacles

|  |  |  |
| --- | --- | --- |
| (a) MADDPG | (b) MADDPG | (c) MADDPG |
| (d) ME-MADDPG | (e) ME-MADDPG | (f)ME-MADDPG |

**FIGURE 12.** Trajectories of 5 agents driven by MADDPG and ME-MADDPG in the scenario with 20 mobile randomly deployed obstacles. (a), (b) and (c) are the trajectories of MADDPG at different times, while (d), (e), and (f) are the trajectories of ME-MADDPG at different times.

**Table 8.** Movement time and total path lengths of the 5 agents when driven by MADDPG and ME-MADDPG in the scenario with 20 mobile randomly deployed obstacles.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | MADDPG | | | | | ME-MADDPG | | | | |
| Agent number | 5 | 2 | 3 | 4 | 1 | 5 | 3 | 1 | 2 | 4 |
| Movement time (s) | 39 | 42.5 | 45 | 55.5 | **66** | 32.5 | 37 | 37.5 | 46 | **51.5** |
| Total path length (m) | **278** | | | | | **204.5** | | | | |

E. Exp5: 36 static regularly deployed obstacles

|  |  |  |
| --- | --- | --- |
| (a) MADDPG | (b) MADDPG | (c) MADDPG |
| (d) ME-MADDPG | (e) ME-MADDPG | (f) ME-MADDPG |

**FIGURE 13.** Trajectories of 5 agents driven by MADDPG and ME-MADDPG in the scenario with static regularly deployed obstacles. (a), (b) and (c) are the trajectories of MADDPG at different times, while (d), (e), and (f) are the trajectories of ME-MADDPG at different times.

**Table 9.** Movement time and total path lengths of the 5 agents when driven by MADDPG and ME-MADDPG in the scenario with 36 static regularly deployed obstacles.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | MADDPG | | | | | ME-MADDPG | | | | |
| Agent number | 5 | 1 | 2 | 4 | 3 | 4 | 2 | 1 | 5 | 3 |
| Movement time (s) | 52.5 | 57.5 | 59.5 | 63.5 | **66** | 41 | 44 | 49 | 51 | **51** |
| Total path length (m) | **299** | | | | | **236** | | | | |

F. Exp6: 36 static randomly deployed obstacles

|  |  |  |
| --- | --- | --- |
| (a) MADDPG | (b) MADDPG | (c) MADDPG |
| (d) ME-MADDPG | (e) ME-MADDPG | (f)ME-MADDPG |

**FIGURE 14.** Trajectories of 5 agents driven by MADDPG and ME-MADDPG in the scenario with static randomly deployed obstacles. (a), (b) and (c) are the trajectories of MADDPG at different times, while (d), (e), and (f) are the trajectories of ME-MADDPG at different times.

**Table 10.** Movement time and total path lengths of the 5 agents when driven by MADDPG and ME-MADDPG in the scenario with 36 static randomly deployed obstacles.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | MADDPG | | | | | ME-MADDPG | | | | |
| Agent number | 5 | 2 | 3 | 1 | 4 | 3 | 5 | 1 | 2 | 4 |
| Movement time (s) | 38.5 | 46.5 | 48.5 | 53 | **67.5** | 34 | 34.5 | 44.5 | 47.5 | **49.5** |
| Total path length (m) | **254** | | | | | **210** | | | | |

(3) Experiment analysis

**Table 11.** The overall results about the movement time and total path lengths of all the exploiting experiments

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Exp No. | Obstacle settings | | | MADDPG | | ME-MADDPG | |
| Number | Static  vs.  Mobile | Random vs. Regular | Maximum Time (s) | Total path length (m) | Maximum Time (s) | Total path length (m) |
| 1 | 20 | Static | Regular | 55 | 265 | 57 | 262 |
| 2 | 20 | Mobile | Regular | 89 | 406 | 58.5 | 254 |
| 3 | 20 | Static | Random | 63 | 224.5 | 49.5 | 196.5 |
| 4 | 20 | Mobile | Random | 66 | 278 | 51.5 | 204.5 |
| 5 | 36 | Static | Regular | 66 | 299 | 51 | 236 |
| 6 | 36 | Static | Random | 67.5 | 254 | 49.5 | 210 |

Figures 9-14 illustrate the trajectories of all the agents driven by MADDPG and ME-MADDPG in all the 6 exploiting experiments. To facilitate comparison, the results of the path length and the movement time are given in Table 11.

From the 6th and the 8th columns of Table 11, it can be seen that, in all the 6 experiments, ME-MADDPG has provided shorter paths than MADDPG. Further, from the 7th and the 9th columns of Table 11, it is shown that in most the cases (except Exp. 1, a slightly different, may be caused by a different initial direction), the agents driven by ME-MADDPG can move to the same targets faster than MADDPG. Overall, ME-MADDPG appears to perform better than MADDPG.

Compare the path lengths in Exp. 1 and Exp. 2. The path length provided by MADDPG has shown a significant increase, while the ME-MADDPG did not and even had a slight decrease. Comparing Figure 9 and 10, it can be seen that the agents driven by MADDPG can make big changes to their path so as to avoid the moving obstacles (Figure 10(c)), and that is why their path lengths had a significant increase compared with the static cases (Figure 9(c)). Let’s go back to Figure 9 and 10, the agents’ paths hardly changed (Figure 10 (f)) except agent 5 (the Cyan one) compared with the static case (Figure 9(f)), the agent driven by ME-MADDPG appears to adapt well to the dynamic environment. At least, compared with MADDPG, it has demonstrated a significant improvement in the adaptability to a dynamic environment. Furthermore, when comparing the results of Exp. 3 and Exp. 4, the same conclusion can be drawn upon.

Let’s make Exp. 1 and Exp. 3, Exp. 2 and Exp. 4 as two control groups and use them to explore the impact of random deployment and regular deployment on the algorithms. By comparing their path lengths, respectively, it can be observed that both ME-MADDPG and MADDPG have generated a shorter path in the environment with randomly deployed obstacles than with regularly deployed obstacles. When comparing Figures 9 and 11, and Figures 10 and 12, respectively, a phenomenon can be found that despite regular deployment makes obstacles scattered, each agent has produced a very similar path, so when they move simultaneously, they have to adjust their path in order to avoid collision with each other, and that could eventually lead to a longer path. In this case, there is no significant difference between ME-MADDPG and MADDPG.

Continue our discussion to explore the impact of the number of obstacles on the algorithms. Here, Exp. 1 and Exp. 5, and Exp. 3 and Exp. 6 are divided into two control groups. Let’s explore the regular deployment case first. By comparing the path lengths driven by ME-MADDPG and MADDPG in Exp. 1 and Exp. 5, it can be found that, for MADDPG, it turns to be harder when there were more obstacles in the environment because it has produced a longer path. On the other hand, however, for ME-MADDPG, it produced an even shorter path in the environment with 36 obstacles than 20 obstacles. By comparing Figure 9(f) and Figure 13(f), it can be observed surprisingly find that ME-MADDPG has produced more compact paths in the environment with 36 obstacles (Figure 13(f)) than 20 obstacles (Figure 9(f)). This is mainly because when the agents are in a denser Environment, they become conservative, and they can always detect obstacles that have made them have to constantly fine-tune the paths, which makes the path gradually compact. On the contrary, for an environment with fewer obstacles, many decisions could be made without detecting obstacles; the path chosen could become bolder and finally has led a dispersed path. The comparison indicates the fact that ME-MADDPG is not as sensitive to the number of obstacles as MADDPG. From the results of Exp. 3 and Exp. 6, it can be seen that the path length of MADDPG increased 29.5m when the number of obstacles increased from 20 to 36, while the path length of ME-MADDPG increased by 13.5, and this supports the conclusion that ME-MADDPG is not as sensitive to the number of obstacles as MADDPG.

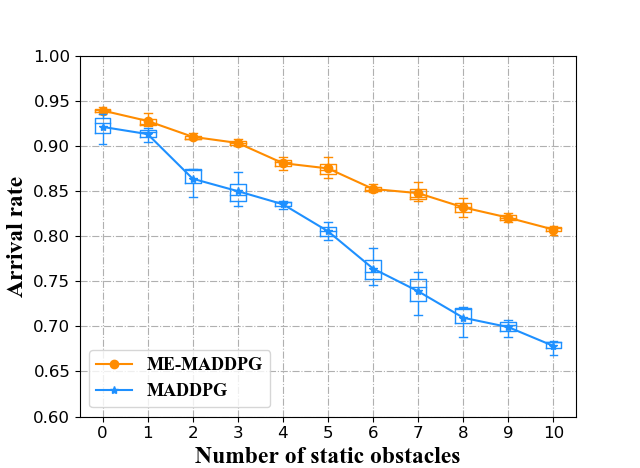
In summary, ME-MADDPG shows better toughness and adaptability to dynamic complex environments than MADDPG.

## 5.4 Statistical experiments

Based on many exploiting experiments, the efficiency of ME-MADDPG has been verified. In this Section, a series of statistical experiments were presented to further guarantee the advantage is statistically valid, not a random result.

(1) Adaptability to different numbers of static obstacles

In this experiment, 5 agents were driven by MADDPG and ME-MADDPG to move through a mission area scatted with different numbers of static obstacles. The number of obstacles increased from 0 to 10 one by one. Each experiment with a specific number of obstacles was repeatedly executed with 1000 episodes, and in each episode, the obstacles were randomly generated with random positions and speeds. The statistical results are shown in Figure 15.

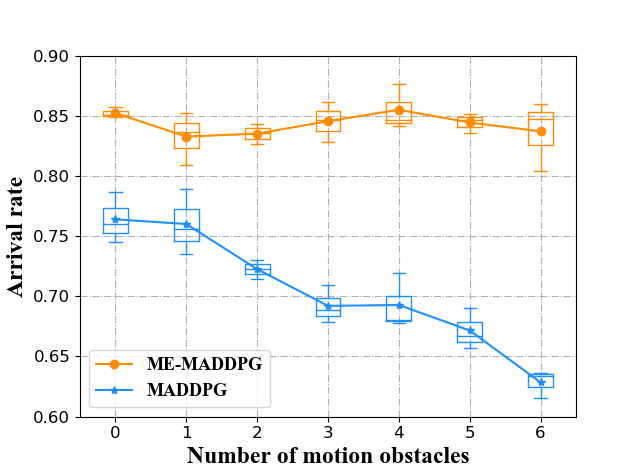


**FIGURE 15.** The statistical trendsof the arrival rate under different numbers of static obstacles of MADDPG and ME-MADDPG

From the statistical box-plots in Figure 15, it can be seen that both the MADDPG and ME-MADDPG provided good stability since the arrival rate did not fluctuate greatly when a determined number of obstacles were set. Horizontally, as the number of obstacles grew, the arrival rates decreased for both algorithms. But the decline rate of MADDPG was obviously greater than ME-MADDPG’s. When there were 5 obstacles, the average arrival rate of MADDPG has already been less than 80%, and when the number increased to 10, its average arrival rate rapidly dropped to below 66.88%, which indicates poor adaptability of MADDPG to the number of static obstacles. In contrast, ME-MADDPG performed a slower decline when the number of obstacles increased. With 10 obstacles, the arrival rate remained up to 80%.

(2) Adaptability to different number of mobile obstacles

In this experiment, we set 6 obstacles totally. In these 6 obstacles, the proportion of mobile obstacles increased gradually, including cases of 0/6, 1/6, 2/6, 3/6, 4/6, 5/6, 6/6. Similarly, each experiment was repeated 1000 episodes, and in each episode, all the obstacles, including static and mobile ones, were randomly generated with random positions and speeds. The statistical results are illustrated in Figure 16.

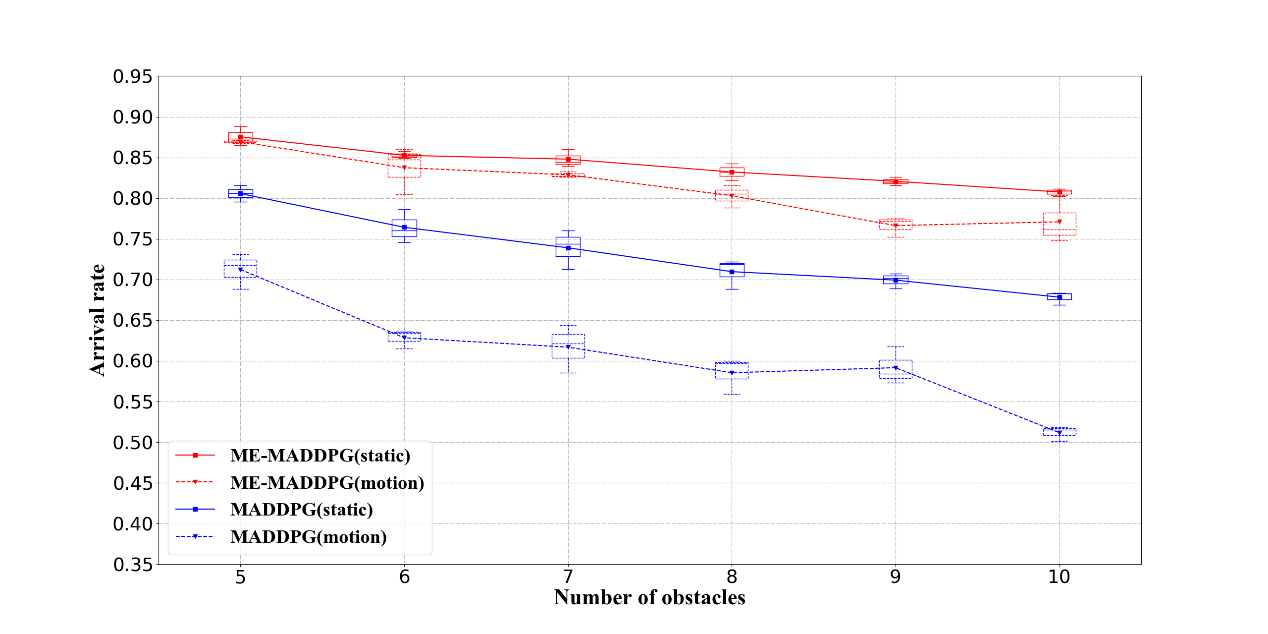


**FIGURE 16.** The statistical trendsof the arrival rate under different numbers of moving obstacles of MADDPG and ME-MADDPG

Figure 16 shows the fact that the average arrival rate of ME-MADDPG was almost stable at 85% when the number of mobile obstacles changing between 0-6. This implies ME-MADDPG owns good adaptability to the environment with obstacles less than 6, no matter whether they are all mobile or static. However, the results also indicate MADDPG was sensitive to the number of mobile obstacles, and its average arrival rate dropped rapidly from 76.02% to 63.60%, with the number of mobile obstacles being increased from 0 to 6. This advantage of ME-MADDPG is also statistically valid.

(3) Adaptability to different numbers of both static and mobile obstacles

In this experiment, all the cases with 5 to 10 static obstacles and 5 to 10 mobile obstacles were tested. Similarly, each experiment ran with 1000 episodes, and in each episode, all the obstacles, including static and mobile ones, were randomly generated with random positions and speeds. The statistical results were included in Figure 17 for easier comparison.



**FIGURE 17.** The statistical trendsof the arrival rate under different numbers of static and mobile obstacles of MADDPG and ME-MADDPG

From the curves of the arrival rate, it can be seen that ME-MADDPG has performed better than MADDPG in all the cases, both in dynamic and static environments. When the deployed obstacles changed from static to mobile, the performance of MADDPG was greatly affected, that the average arrival rate was reduced by almost 12%. However, ME-MADDPG has maintained a good performance. In particular, when the number of static obstacles increased from 5 to 10, the average arrival rate of ME-MADDPG only decreased from 87.3% to 80.2%. When the number of mobile obstacles increased from 5 to 10, the average arrival rate of ME-MADDPG also declined slowly, from 86.9% to 76.1%. In the case of 10 obstacles, the difference of the average arrival rate between static and mobile obstacles stayed at 4.06%, which indicates good robustness of ME-MADDPG to a dynamic environment. The statistical arrival rates stably support the performance advantages of the ME-MADDPG.

Considering all the experiments, the training experiments in **5.2**, the exploiting experiments in **5.3,** and the statistical experiments in **5.4**, we conclude that the proposed ME-MADDPG gains great improvement in performance comparing with MADDPG.

# 6 CONCLUSIONS AND FUTURE WORK

This paper proposes a new multi-agent DRL algorithm, ME-MADDPG, in which the artificial potential field method has been introduced to construct a dynamic mixing sampling strategy, and a delaying learning policy is adopted to stabilize the training process. A series of experiments have been designed to verify the performance of the proposed ME-MADDPG algorithm, including the training experiments, exploiting experiments, and statistical experiments. The results have indicated that compared with MADDPG, ME-MADDPG has a faster convergence speed and a better convergence effect efficient multi-agent motion planner and has good adaptability to different dynamic complex environments.

Despite well performed in solving MAMP problem, ME-MADDPG still has some shortcomings that may limit its application. For instance, with a centralized training decentralized execution framework, it will be difficult for ME-MADDPG to be expanded to large-scale applications. This is because as the number of agents increases, train a convergent model will be more and more difficult. Some distributed architectures, such as A3C, maybe a good choice for improving the training mechanism. Alternatively, introducing supervised learning methods into MARLs could be a promising research direction, and some excellent traditional evolutionary methods are also valuable that will be considered in improving the MARLs in our future work. In fact, this direction is attracting some attention in combinatorial optimization communists.

# ACKNOWLEDGMENTS

The authors would like to express their sincere gratitude to the National Natural Science Foundation of China (Grant No. 62003267), the Natural Science Foundation of Shaanxi Province (Grant No. 2020JQ-220), and the Open Project of Science and Technology on Electronic Information Control Laboratory (Grant No. JS20201100339) for providing funds to support this research. Also, we would thank Prof. Chen for his linguistic assistance during the preparation of this manuscript.

# REFERENCES

1. Luis C E, Schoellig A P. Trajectory Generation for Multiagent Point-To-Point Transitions via Distributed Model Predictive Control. *IEEE Robot. Autom. Lett.* 2018;1-1.
2. D Panagou. A Distributed Feedback Motion Planning Protocol for Multiple Unicycle Agents of Different Classes. *IEEE Trans. Autom. Control*. 2017;62(3):1178-1193.
3. Israelsen J, Beall M. Automatic collision avoidance for manually tele-operated unmanned aerial vehicles. In: *IEEE International Conference on Robotics and Automation (ICRA 2014)*. Hong Kong, China, 2014 May 31-June 7.
4. Parker L E. Path planning and motion coordination in multiple mobile robot teams. *Encyclopedia of complexity and system science*. 2009; 5783-5800.
5. Karaman S, Frazzoli E. Sampling-based Algorithms for Optimal Motion Planning. *Int J Rob Res.* 2011;30(7):846-894.
6. Rodriguez S, Tang X, Lien J M. An obstacle-based rapidly-exploring random tree. In: *The 2021 International Conference on Robotics and Automation*. IEEE, 2006.
7. Kala R, Warwick K. Multi-vehicle planning using RRT-connect. *Paladyn, Journal of Behavioral Robotics*. 2011;2(3): 134-144.
8. Ragaglia M, Prandini M, Ba Scetta L. Multi-agent Poli-RRT\* Optimal constrained RRT-based planning for multiple vehicles with feedback linearisable dynamics. In: *Modelling & Simulation for Autonomous Systems Workshop*. 2016.
9. Jiang J, Wu K. Cooperative Pathfinding based on high-scalability Multi-agent RRT\*. https://arxiv.org/abs/1911.07840
10. Hildebrandt A, Klischat M, Wahrmann D. Real-Time Path Planning in Unknown Environments for Bipedal Robots. In: *IEEE International conference on robotics and automation*. 2017;2(4): 1856-1863.
11. Grenouilleau F, van Hoeve W J, Hooker J N. A multi-label A\* algorithm for multi-agent pathfinding. In: *International Conference on Automated Planning and Scheduling*. 2019;29: 181-185.
12. Al-Mutib K, AlSulaiman M, Emaduddin M. D\* lite based real-time multi-agent path planning in dynamic environments. In: *2011 Third International Conference on Computational Intelligence, Modelling & Simulation. IEEE*. 2011;170-174.
13. Ogata K. A Generic Approach on How to Formally Specify and Model Check Path Finding Algorithms: Dijkstra, A\* and LPA. *Int J Softw Eng Know*. 2020;30(10): 1481-1523.
14. Zhao T, Li H, Dian S. Multi-robot path planning based on improved artificial potential field and fuzzy inference system1. *J. Intell. Fuzzy Syst.* 2020;39(5):7621-7637.
15. Kim J M, Heo J M, Jung S Y. Path-planning using Modified Genetic Algorithm and SLAM based on Feature Map for Autonomous Vehicle. *Journal of Korean Institute of Intelligent Systems*, 2009, 19(3):381-387.
16. Alzugaray I, Teixeira L, Chli M. Short-term UAV path-planning with monocular-inertial SLAM in the loop. In: 2*017 IEEE International Conference on Robotics and Automation (ICRA)*. *IEEE.* 2017;2739-2746.
17. Sutton R, Barto A. Reinforcement learning: an introduction. Second Edition, MIT Press, 2017.
18. Foerster J N, Assael Y M, De Freitas N, et al. Learning to communicate with deep multi-agent reinforcement learning. https://arxiv.org/abs/1605.06676
19. Leibo J Z, Zambaldi V, Lanctot M. Multi-agent Reinforcement Learning in Sequential Social Dilemmas. https://arxiv.org/abs/1702.03037.
20. Shalevshwartz S, Shammah S, Shashua A. Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. https://arxiv.org/abs/1610.03295.
21. Lowe R, Wu Y, Tamar A. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *Neural Information Processing Systems*, 2017;6379-6390.
22. Wang Z, Wan R, Gui X. Deep Reinforcement Learning of Cooperative Control with Four Robotic Agents by MADDPG. In: *2020 International Conference on Computer Engineering and Intelligent Control (ICCEIC). IEEE*. 2020;287-290.
23. Han Q, Shi D, Shen T. Joint Optimization of Multi-UAV Target Assignment and Path Planning Based on Multi-Agent Reinforcement Learning. I*EEE Access*. 2019; 7:146264-146272.
24. Fujimoto S, Hoof H, Meger D. Addressing function approximation error in actor-critic methods. In: I*nternational Conference on Machine Learning. PMLR*. 2018;1587-1596.
25. Wang L, Wang K, Pan C. Multi-agent deep reinforcement learning-based trajectory planning for multi-UAV assisted mobile edge computing. *IEEE Trans. Cogn. Commun. Netw.* 2020;7(1): 73-84.
26. Zhang G, Li Y, Xu X, et al. Multiagent Reinforcement Learning for Swarm Confrontation Environments. In: *International Conference on Intelligent Robotics and Applications.* 2019;533-543.
27. Chen Y F, Liu M, Everett M, et al. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In: *2017 IEEE international conference on robotics and automation (ICRA). IEEE*. 2017;285-292.
28. Chen Y F, Everett M, Liu M. Socially aware motion planning with deep reinforcement learning. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE*. 2017;1343-1350.
29. Long P, Fan T, Liao X. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In: *2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE*. 2018;6252-6259.
30. Semnani S H, Liu H, Everett M, et al. Multi-agent motion planning for dense and dynamic environments via deep reinforcement learning. *IEEE Robot. Autom. Lett.* 2020;5(2): 3221-3226.
31. Lawrence D, Frew E, Pisano W. Lyapunov vector fields for autonomous UAV flight control. In: *AIAA Guidance, Navigation and Control Conference and Exhibit.* 2007;6317.
32. Littman M L. Value-function reinforcement learning in Markov games. *Cogn Syst Res.* 2001;2(1): 55-66.
33. Wan K , Gao X , Hu Z, et al. A RDA-Based Deep Reinforcement Learning Approach for Autonomous Motion Planning of UAV in Dynamic Unknown Environments. In: *4th International Conference on Control Engineering and Artificial Intelligence(CCEAI 2020)*, 2020, 1487:012006.
34. Wang C, Wang J, Shen Y. Autonomous Navigation of UAVs in Large-Scale Complex Environments: A Deep Reinforcement Learning Approach. *IEEE Trans. Veh. Technol.* 2019;68(3): 2124-2136.
35. Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning. https://arxiv.org/abs/1509.02971
36. Wan K, Gao X, Hu Z, et al. Robust motion control for UAV in dynamic uncertain environments using deep reinforcement learning. *Remote Sens.* 2020;12(4): 640.