2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT) | 979-8-3503-6944-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/DCOSS-IoT61029.2024.00014

Multi-Behavior Multi-Agent Reinforcement Learning for Informed Search via Offline Training

Songjun Huang†, Chuanneng Sun†, Ruo-Qian Wang‡, and Dario Pompili†
† Department of Electrical and Computer Engineering, Rutgers University–New Brunswick, NJ, USA
‡ Department of Civil and Environmental Engineering, Rutgers University–New Brunswick, NJ, USA
{*songjun.huang, chuanneng.sun, rq.wang, pompili*}*@rutgers.edu*

***Abstract*—In modern informed search missions, Multi-Robot Systems (MRSs) are playing more and more important roles due to their flexibility in exploring environments. Reinforcement learning (RL) is now widely used as a decision-making method for MRS. However, existing RL-based and conventional model-based frameworks cannot deal with some challenges posed by the real-world environment. To address these challenges, a Multi-Behavior Multi-Agent Reinforcement Learning (MBMARL) framework via offline reinforcement learning method was developed. In this framework, each agent is deployed with multiple behavior policies to let the agent have choices on behaviors given a state. The proposed framework is compared with traditional reinforcement learning frameworks, including Multi-Agent Actor Critic (MAAC) and REINFORCE. The result shows that MBMARL outperforms others in both aspects of total reward and convergence time.**

***Index Terms*—Informed Search, Offline Reinforcement Learning, Multi-Agent Reinforcement Learning.**

I. Introduction

Overview: Informed search, also known as heuristic search, is a type of search algorithm to efficiently navigate through large search spaces utilizing heuristic information. The main objective of informed search is to find a solution or path to a goal state while minimizing the number of steps or nodes expanded during the search process. Informed search is widely used in various robotics applications such as search and rescue (SAR) [1], path planning [2], [3], and underwater space exploration [4]. In these applications, Multi-Robot Systems (MRSs) find extensive employment, prompting researchers to develop various algorithms and frameworks to enhance MRS functionalities. However, the real-world environments in which these frameworks are implemented pose various significant limitations and challenges that can make these algorithms and frameworks infeasible.

Motivation: The *first challenge* is the absence of comprehensive global environmental information. For example, in certain search tasks, robots and participants do not know the locations of the targets. Instead, only location-related information is available, such as field strength or substance density. This situation poses challenges for conventional informed search algorithms such as A* and its derivatives [5], [6]. Typically, these algorithms are based on heuristic functions defined using distance metrics [7], [8] to achieve optimal performance. However, it is difficult to establish a connection between distance metrics and available location-related information. As a result, such distance metrics are unattainable in these scenarios. Even if we can adopt location-related information as a heuristic function, it is difficult to design a cost function that can be coupled with the adopted heuristic function.

This work is supported by the NSF RTML Award No. CCF-1937403.

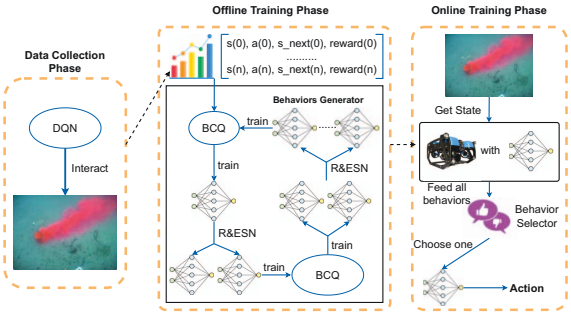


Fig. 1: The MBMARL framework can be divided into three phases: Data Collection, Offline Training, and Online Training. In the Offline Training phase, R&ESN means replicating and embedding structured noise.

The *second challenge* is the complexity of real-world environments, which often leads to problems of sparsity and local optimum. For example, consider an MRS tasked with finding the origin of a diffusive substance in a marine ecology mission [9]. The substance's distribution can become notably sparse because of factors such as wind and currents. At the same time, unpredictable eddy currents can create irregularities, leading to locally high concentrations away from the source. A similar complexity arises when MRS is required to autonomously explore equally significant Points of Interest (PoIs). These scenarios present several issues: (i) Traditional machine learning is not suitable, lacking appropriate datasets for training in each distinct environment; (ii) Model-based approaches, such as density gradient ascent tracking, fail because robots can become trapped in local high-concentration areas and cannot escape; (iii) Conventional reinforcement learning methods struggle due to sparse reward problems [10].

The *third challenge* is poor communication conditions. In some scenarios, like underwater environments or post-disaster situations, stable communication cannot be ensured. This makes conventional multi-agent reinforcement learning (MARL) frameworks impractical, where stable communication is required and difficult to achieve. Although some researchers propose centralized training with decentralized execution (CTDE) [11], achieving this is challenging for several reasons: (i) Discrepancies exist between real-world

2325-2944/24/$31.00 ©2024 IEEE
DOI 10.1109/DCOSS-IoT61029.2024.00014

and simulated environments; (ii) The real-world effectiveness of policies trained in simulations is heavily based on the accuracy with which the simulated environments model reality; (iii) Policies lack the capacity to optimize during execution without communication. As a result, a framework that addresses these challenges with good performance is needed.

Related Work: Researchers have proposed frameworks to perform individual informed search tasks using a single robot or collaboratively with MRS. In these studies [12]–[15], the researchers proposed several model-based frameworks improved from conventional informed search algorithms such as A*. Although they can achieve better performance in their applications, they still need global environmental information to acquire the distance needed to design the cost and heuristic function. The use of location-related information to search has led to research on gradient tracking methods. In these works [16]–[18], researchers proposed several gradient tracking algorithms for searching and planning. However, they do not take into account the local optimum introduced by the complexity of the real-world environment. To address the mentioned challenges, researchers are increasingly focusing on the development of learning-based algorithms, particularly RL. In these works [19]–[21], the proposed MARL frameworks achieve good performance. However, all frameworks need consistent state exchange among agents, which is impractical under poor communication. Also, the sparse reward problem is not fully considered in these works. The researchers then introduced the concept of offline RL [22]. The proposed offline RL frameworks [23] can deal with the sparse reward problem. However, they are all done in a single-agent manner. At the same time, since there is only one behavior policy, the local optimum problem cannot be solved.

Our Approach: To tackle these challenges, we propose a framework called Multi-Behavior Multi-Agent Reinforcement Learning (MBMARL). This stable RL-based framework circumvents the need for comprehensive global environmental data, effectively handles intricate real-world settings, and minimizes dependence on communication. Fig. 1 describes the proposed framework. The framework can be divided into three different phases. The workflow is the following—i) the real-world data collected in the data collection phase will be used in the offline training phase to develop multiple behavior policies via offline RL; ii) the behavior policies together with a behavior selector are equipped for each agent; iii) each agent is deployed to the environment while the behavior selector is trained in the online training phase. This framework addresses challenges as follows: (i) Global Information Not Needed: The learning-based method negates the requirement for global environmental data. Throughout training, optimal performance does not necessitate distance metrics; (ii) Sparse Reward and Local Optima: The sparse reward issue is tackled through offline RL training of behavior policies [22]. Agents possess multiple behavior policies, mitigating local optimum entrapment and improving action selection diversity within states; (iii) Reduced Communication Dependency: Training of behavior policies, which guides agents to act under different states, occurs offline, obviating the need for communication. Behavior selector training involves communication, but incorporates measures to accommodate poor communication conditions. Under different communication scenarios, our framework consistently outperforms the baseline approaches.

II. Proposed Solution

In this section, we will elaborate on the theoretical basis, design, and functionalities of our MBMARL framework.

A. Data Collection Phase

In our MBMARL framework, data collection is done by letting agents interact with the environment using any policy. We adopt an untrained Deep Q-Learning (DQN) model for data collection, and this model is trained iteratively during the process. The training process is as described in [24]. Actually, any interaction policy can be used here, even random movement, since we just need to let agents gather a series of transitions for offline training. The reason why we use this untrained DQN model during data collection is that it allows us to showcase the advantage of offline RL, where the performance of the policy obtained from offline RL can exceed the performance of the policy used in data collection. This makes offline RL a viable option in certain applications compared to imitation learning and supervised learning. More detailed results are presented in Sect. III.

During the data collection phase, an agent will first receive a state s. The agent will select an action a from the set of actions A given s with the ϵ-greedy policy to encourage exploration to gather more data. Q is a neural network with input values s and output values for all actions $a \in A$. Based on the selected action a, the agent will reach the next state s_n. For each step i, we collect the state s_i, action a_i, reward r_i, and the next state s_{ni} to form a transition $d =< s_t, a_i, r_i, s_{ni} >$ and store it in the replay buffer B, which will be used for training Q. To train this Q, we introduce a target network T, which is initialized to have the same weights as Q and updated at a predefined frequency by loading the weights of Q. We first sample a batch of B. The states are fed to Q, and the next states are fed to T to obtain the outputs q and y, respectively. Then, q and y will be used to calculate the loss as $L(\omega) = \frac{1}{2}\big(q - (r + \gamma \cdot y)\big)^2$, where ω is the weight of Q; r is the reward and γ is the discount factor. In this way, we collect massive transitions and use them as our data set D.

B. Offline Training Phase

The data set D is used for offline training, as described in Algo 1. In the offline training phase, various offline policy constraint RL algorithms described in [22] can be used to train a model. In our experiment, we take BCQ as an example due to its notable performance in mitigating the extrapolation error and incorrect estimation of Q value [25]. We first divide D into multiple batches and build 3 new networks Q_{bcq}, T_{bcq} and G. Q_{bcq} and T_{bcq} are defined as those used in data collection. G is the network with the input of the state and the output of action values, and we can compute the probability of every action with G since $G(a|s) \approx \pi(a|s)$. When selecting an action, rather than directly choosing the action that maximizes

Algorithm 1 Offline Training Phase

for $Epoch = 1, 2, ...$ **do**
  for ba in all batches **do**
    Choose batch β from D
    Constrain β_{new} from β with BCQ
    Train Q_{bcq} with β_{new}
  if $Epoch \mod e == 0$ **then**
    R-and-P Q_{bcq} to k new models
  if Number of models meets requirement **then**
    break
Get r of each model
Remove low reward models
Calculate KL-divergence among models
Choose minimum-dependency n behaviors

$Q_{bcq}(s)$, we first filter the batch by only considering the actions that satisfy $\frac{G(a|s)}{\max_{\hat{a}} G(\hat{a}|s)} \geq \tau$, where τ is a threshold. We call this condition $\mathcal{C}$. If $\tau = 0$, it is traditional Q-learning. If $\tau = 1$, it becomes the cloning of the behavior in the data collection. During the offline training phase, to prioritize training the model based on D while also exceeding the performance of the behavior policy used in data collection, we set τ at 0.3. Then, the policy becomes the following,

$$\pi(s) = \underset{a|\frac{G(a|s)}{\max_{\hat{a}} G(\hat{a}|s)} \geq \tau}{\arg\max} Q_{bcq}(s, a). \tag{1}$$

In this policy, rather than selecting an action from all actions, we select an action from the actions that satisfy $\mathcal{C}$ at each step. To train the policy, the loss function is as follows,

$$\mathcal{L}(\theta) = l_R\left(r + \gamma \underset{a|\frac{G(a|s)}{\max_{\hat{a}} G(\hat{a}|s)} > \tau}{\max} T_{bcq}(s', a') - Q_{bcq}(s, a)\right), \tag{2}$$

where l_R is the learning rate; s' and a' are the next state and action. For each epoch, we train the whole data set D once. After every training interval e, we perform R&ESN.

R&ESN: This process involves replicating the model, embedding structured noise in each new model, and getting models closely related to the original model. R&ESN is just one possible method to generate multiple distinct behaviors, and there may be other potential methods that can be explored in future research. Structured noise is not completely random, but instead is based on the average of the absolute weight values in each layer of the model $n_{rd} = U\big(-\sum_{\omega\in\Omega}\omega, \quad \sum_{\omega\in\Omega}\omega\big)$, where U represents the uniform distribution, two items in the parentheses are the lower bound and upper bound, ω is each weight in this layer, and Ω is the set of all weights in this layer. The structured noise in our design is intended to be distinct for different models while retaining some original features of the model. The number of R&ESN times required depends on the desired number of behavior policies, and the number of behaviors needed should be decided from experimental experience.

Behavior Policies Filtering Mechanism: The workflow or R&ESN and this mechanism are depicted in Fig. 2. We

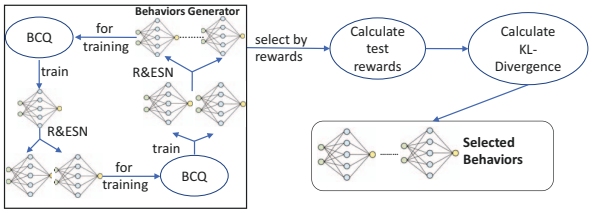


Fig. 2: The workflow of Replicate and Embed Structured Noise (R&ESN) and Behavior Policies Filting Mechanism.

test each behavior policy in the pool by deploying it to an agent to interact with the environment and record the total reward. The low-reward behavior policies are then removed. To ensure diversity between the selected behavior policies, we minimize interdependencies by calculating the Kullback-Leibler (KL) divergence [26] among the remaining behavior policies. Only when different behavior policies offer different actions for the same state can the agent choose the most appropriate action from them. We first sample a set of test states $S_{test} = (s_{test_1}, s_{test_2}, \cdots, s_{test_n})$ from the environment. Then we input S_{test} into each behavior policy model and record the output actions as $\mathcal{O}_i = (a_1^i, a_2^i, \cdots, a_n^i)$, where i indicates that this is the output set of the behavior policy i^{th}. The KL divergence between the i^{th} and the j^{th} behavior policy can be calculated as $\mathcal{KL}_{ij} = \mathcal{KL}(\mathcal{O}_i, \mathcal{O}_j) = \mathcal{O}_j \cdot (log\mathcal{O}_j - log\mathcal{O}_i)$. The $\mathcal{KL}_{ij}$ will then be further used to calculate the total KL value of each policy behavior. For the i^{th} behavior policy, it can be calculated as $\mathcal{KLV}_i = \sum_{j=1, j\neq i}^{n_{bp}} \mathcal{KL}_{ij}$ where n_{bp} means the number of behavior policies. After we have $\mathcal{KLV}$ for all behavior policies, we choose the n behavior policies with the largest $\mathcal{KLV}$.

C. Online Training Phase

During online training, we train behavior selectors for different agents to select behavior policies based on the state. The behavior selector uses ϵ-greedy exploration with ϵ initially set to 0.9 and the decay rate is 0.001 per step.

To train this neural network model C_θ parameterized by θ, we define a target neural network $C_{\theta'}^t$, where θ' is initialized to be the same as θ. Then $C_{\theta'}^t$ remains unchanged until C_θ is updated 100 times. Then, $C_{\theta'}^t$ is updated to be the same as C_θ. In other words, $C_{\theta'}^t$ is updated after each 100 updates of C_θ. The loss function is,

$$\mathcal{L}_\theta = \frac{1}{2}\left(C_\theta(s|a) - r + \gamma \cdot C_{\theta'}^t(s'|a')\right)^2, \tag{3}$$

where a and a' are the selected model id and the next selected model id, s and s' are the current input states and next input states. r is the reward obtained from the action guided by the selected behavior. γ is the discount factor and it is 0.95 here.

Fig. 3 illustrates the behavior selector training process. Each agent is equipped with a behavior selector, and these selectors can communicate to share agent states. The behavior selector then uses all states to train itself. However, if communication is poor between agents, the behavior selector uses the last received information. The selected behavior policy is used for the next steps i, where i is the switching interval.

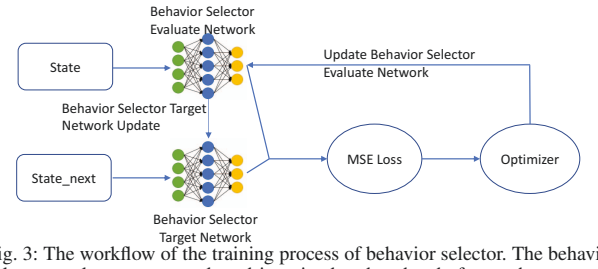


Fig. 3: The workflow of the training process of behavior selector. The behavior selector evaluates a network and is trained and updated after each step.

This framework relies less on communication, as the information exchange for training behavior selectors is less than the training of behavior policies, and behavior policies are trained in the offline phase, which requires no communication. At the same time, the framework does not need global environmental information for training. Additionally, agents deployed with multiple behavior policies trained with offline RL can deal with local optimum and sparsity.

III. Performance Evaluation

Comparison Plan: We compare the proposed algorithm, MBMARL, with two other algorithms: (i) Decentralized REINFORCE, in which each agent follows the REINFORCE algorithm to act without exchanging any information [27]; (ii) Multi-Agent Actor-Critic (MAAC) [28], where each agent observes/receives the states and actions of all other agents to update its critic and independently updates its actors based on its own experiences; when an agent is isolated (i.e., cannot communicate with others), it cannot receive the states and actions from others and therefore cannot perform the update for both its actor and critic; (iii) model-based gradient ascent tracking methods.

Environment Setup and Definition: We model two environments and conduct a field experiment and emulation. The first environment is an underwater gas leak scenario and is also where we conduct most of our experiments. We believe that this is a typical example of an informed search in a complex environment. The spatial distribution cannot be described with mathematical equations, and the dataset we use to describe this distribution is based on real-world situations, which encompass the complexities of sparsity and local optimum due to currents and eddies. To simulate it, we define an agent as a 3D rigid body capable of translating in space and moving in six directions (forward, backward, right, left, up, and down). The ***state*** of each agent includes its position and the concentration of the plume at that position. The environment is described by a cube with dimensions defined as the differences between the maximum and minimum coordinates on the x, y, and z axes. The ***action*** is a vector of size 6, corresponding to the six possible directions. After each step, the agent finds the nearest point in the data set and updates its state accordingly.

The reward function is defined as follows: (i) If the concentration from the current state is greater than that of the previous state, the reward is defined as +5. In contrast, the reward is defined to be -5; (ii) If the agent explores outside the boundary, the reward is defined as -100 and the agent will go back into the boundary; (iii) If the location with the highest concentration in the data set is in the detection range of the agent, the reward is defined to be +100 and this episode is performed.

To validate the generalizability of our framework in regular 2D navigation and search, we tested it in the second environment, which is the open benchmark environment MiniGrid-Empty [29]. We first enlarged the size of the grid environment to 30×30. Then we set the initialization positions of the agents randomly in the upper left part of the environment. Additionally, we define the reward function as $r = 1/(d_t^{a_i} + 1)$, where $d_t^{a_i}$ is the distance between the agent i^{th} and the target. If the agent moves outside the boundary, the current episode is done and the reward for this step is -10.

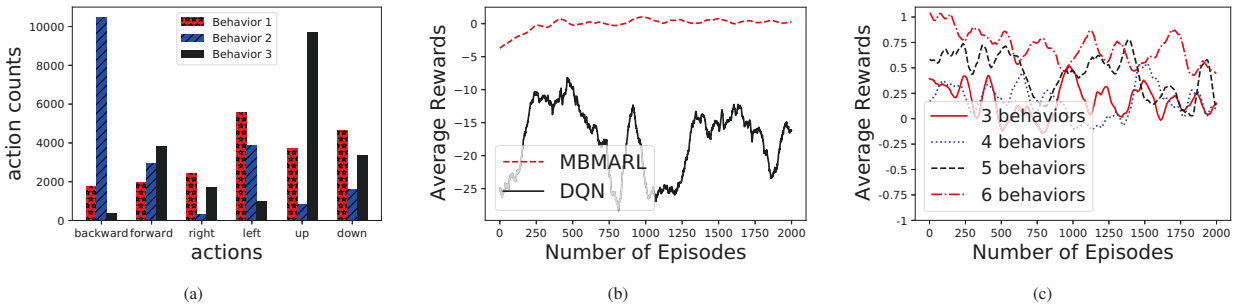
To further test the framework, we conduct a field experiment in a parking lot. This experiment simulates the application of searching for injured or lost personnel with uncertain information about their whereabouts. Possible locations can be regarded as equally significant POIs. We also performed an emulation with an underwater environment simulator and tested the framework on an embedded AI computing device, which can be used in future experiments.

Training and Testing Setup: Both the models of DQN, BCQ, and behavior selectors have a two-layer fully connected neural network structure with a hidden layer size of 256. For MAAC, the actor model has two layers with a hidden size of 256. The first layer is followed by a ReLU and the second is followed by a Tanh. The critic model has three layers with a hidden size of 256. The first two layers are followed by ReLU, respectively. The critical learning rate is 0.005. It uses the Ornstein-Uhlenbeck noise [30] to encourage exploration. For REINFORCE, the model has one shared input layer with a hidden size of 256 followed by ReLU and two output layers. For all unspecified models, the learning rate is 0.001, the Horizon is 1, and the discount factor is 0.99.

Another thing worth noticing is that, in the experiments, we uniformly use a discrete space in the action space for comparison. However, we claim that the proposed framework can also work for continuous state and action spaces by changing the action space to continuous space, the data collection method to the algorithm for continuous action space such as DDPG, and the offline training method to continuous offline RL algorithms such as continuous BCQ [25].

A. Simulation Results from Environment 1

Different Policies Act Differently: We must ensure different behavior policies can behave differently under the same states in order to make them work as expected. Fig. 4(a) shows the action distributions of different behavior policies. Each behavior policy is tested under the same 20,000 randomly sampled states. The result shows that they behave differently, with behavior 1 showing more evenly distributed actions compared to behaviors 2 and 3. Behavior 2 tends to move backward along the x-axis and left along the y-axis, while being conservative in the z-direction. Behavior 3 is more active along the z-axis but conservative in other directions. However,



(a) (b) (c)

Fig. 4: (a) Action distribution of different behavior policies; (b) Average episode rewards over agents at each step for the data collection policy and MBMARL in online training; (c) Average episode rewards vs. number of episodes for the number of behaviors = 3, 4, 5 and 6 under the communication failure probability = 0.5 and behavior switch interval = 50.

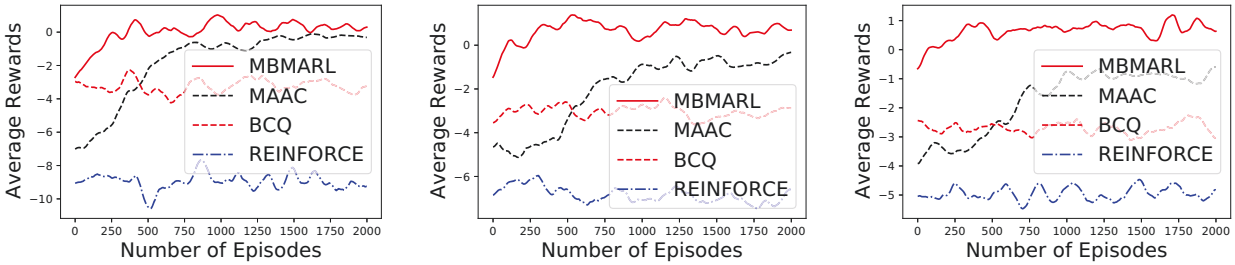


Fig. 5: Average episode rewards of the agents vs. Number of training episodes when Number of agents = 2, 4, 6, respectively. The communication failure probability of MBMARL and MAAC is 0.5; the behavior switch interval of MBMARL is 10.

agents with these behavior policies are able to navigate to areas with larger concentrations when necessary.

Trained Policy vs. Data Collection Policy: Offline RL offers the advantage of achieving better policies compared to the data collection algorithm, making it more practical than other similar approaches such as imitation learning and supervised learning [31]. Even with data sets collected from less optimized behavior policies, the trained offline RL model can perform better. In our experiment, the data collection policy is a DQN model trained during the process. Fig. 4(b) shows that the behavior policy does not converge to a specific reward value and fluctuates over 2000 episodes, while the well-trained 3-behavior framework performs better with an average reward of 0 to +5. This demonstrates that the offline RL model can achieve high rewards and fast convergence even when trained with a poorly behaved data collection policy.

Rewards vs. Number of Behaviors: The number of behaviors is a critical parameter in our MBMARL framework. With more behaviors, agents have more action choices for a given state. In Fig. 4(c), we plot the average rewards of episodes against the number of behaviors, assuming a probability of communication failure of 0.5 and a switching interval of 50. The average rewards for different numbers of behaviors fluctuate between -0.25 and +1, with slightly higher rewards observed with more behavior policies. The large switching interval is the reason for this, as agents may not change behaviors in time based on the acquired state, resulting in unsuitable behavior selection. As the switching interval decreases, the advantages of this become evident, as confirmed by the following results.

Rewards vs. Number of Agents: In Fig. 5, we compare the average rewards on agents at each step versus the number of agents for the MBMARL (with 3 behaviors), MAAC, and REINFORCE frameworks in online training. *During the online training, for MBMARL, the behavior policies are already well trained offline and only the behavior selectors are updated; for MAAC and REINFORCE, the behavior policy is updated.* The communication failure probability for MBMARL and MAAC is set to 0.5. We found that a switching interval of 50 steps is too large, so we set it to 20 in this experiment. The three figures show that MAAC performs better with more agents, as each agent acquires more information after each step, leading to better trained critics. However, MAAC still converges to a negative value after around 1000 episodes. REINFORCE performs poorly due to lack of communication and monotony in behavior policy. MBMARL quickly converges to the range of 0 to +2 after around 500 episodes. As the number of agents increases, the intensity of the fluctuations decreases. Compared to traditional RL algorithms with or without communication, our MBMARL framework achieves shorter convergence times and higher total rewards with varying numbers of agents.

Ablation Study on the Effect of Multi-Behavior: From the comparison of MBMARL and BCQ in Fig. 5, we want to prove that multibehavior is effective. BCQ has all the same offline training procedures with the same number of episodes but without R&ESN and behavior filtering mechanisms. From the results, we can find that MBMARL outperforms BCQ.

Reliance on Communication: Fig. 6 compares the performance of MBMARL and MAAC with 4 agents with different probabilities of communication failure with the switch interval of 20. This is also the online training phase, as described in

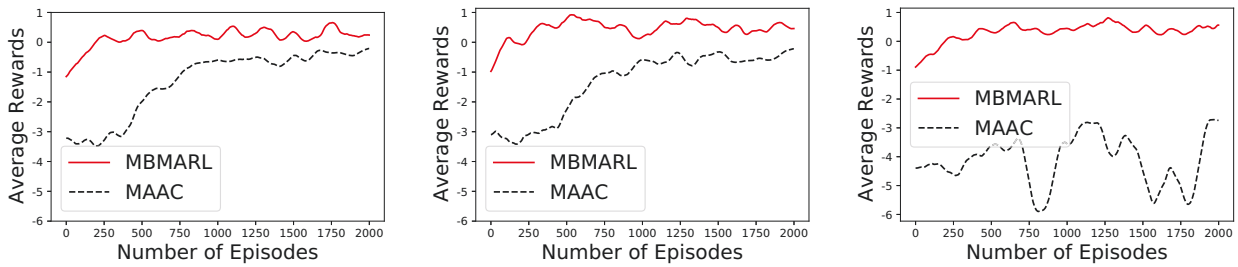
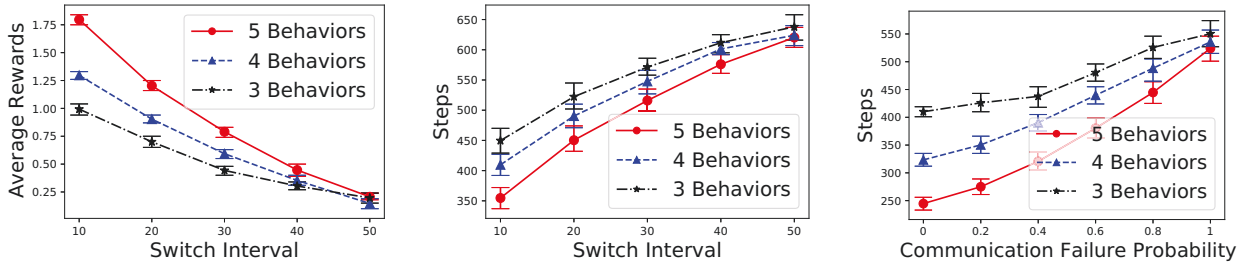


Fig. 6: The episode average reward over agents at each step in the online training of MBMARL and MAAC vs. number of episodes when communication failure probability = 0.8, 0.5, 0.2, respectively, when switching interval of MBMARL is 20, number of behavior policies of MBMARL is 3 and number of agents 4. REINFORCE is not compared in this result, since in REINFORCE there is no communication in the REINFORCE framework.



(a) (b) (c)

Fig. 7: (a) Episodes average rewards vs. switch interval when the number of agents is 4 and communication failure probability is 0.5; (b) Steps to find the source vs. switch interval when the number of agents is 4 and communication failure probability is 0.5; (c) Steps to find the source vs. communication failure probability when the switching interval is 10 and the number of agents is 4.

the previous paragraph. From the figures, it is evident that the performance of MAAC is highly dependent on communication. When the probability of communication failure is high (e.g. 0.8), MAAC's average episode reward does not converge and remains low. As communication improves, MAAC performs better with increasing episodes. We find that the improvement is not substantial as the probability of communication failure decreases (e.g., from 0.5 to 0.2). This is due to the lack of data in underwater environments, where the behavior policy lacks sufficient information for training even with improved communication. In contrast, our MBMARL framework consistently outperforms MAAC in terms of average episode reward and is less affected by different communication conditions. The complete impact of communication on behavior selector training cannot be adequately illustrated solely by using a single metric in Fig. 6. The reason is that in each episode, there is a limitation on the number of steps in which the agents are not able to find the source. However, it will be noted in the following results.

Switch Interval and Communication Condition: Fig. 7(a) and 7(b) demonstrate that increasing the switching interval in behavior selectors leads to decreased total rewards and increased steps to find the source for each agent. However, the rate of decrease in rewards and increase in steps is lower as the switching interval increases. Fig. 7(c) shows that better communication results in fewer steps to find the source, especially for agents with more behaviors. Despite similar average episode rewards under different communication conditions (as shown in Fig. 6), agents can find higher concentrations in various directions under different communications. As we described in the previous subsection, if the limitation on the number of steps is removed, improved communication allows behavior selectors to select behaviors more effectively, resulting in decreased steps to find the source.

We also test the traditional density ascent tracking method and evaluate it with the number of steps to find the source. Out of 100 independent trials, agents only found the source 27 times and the mean steps were 643. For our MBMARL framework, under 4 behavior policies, 0.5 communication failure probability, and a switching interval of 10, the agents find the source in 93 out of 100 trials, and the average steps are 389. This demonstrates the ability of our framework compared to model-based methods.

B. Simulation Results from Environment 2

Our framework's generalizability is validated in the MiniGrid-Empty benchmark environment with slight modifications. By doing this experiment, we want to claim that our framework is still effective in regular 2D navigation and search missions where global environmental information is available so that distance metrics can be used. Fig. 8 presents the results, showing that our MBMARL outperforms MAAC and REINFORCE. Although MAAC and REINFORCE require continuous action spaces, we successfully adapted our framework. Despite differences in action distributions from different behavior policies, MBMARL performs well when the number of agents is 4 or 6, similar to the results in environment 1. On the other hand, MAAC and REINFORCE show negative convergent average rewards, suggesting that agents are moving out of the environment boundary. We also record the steps

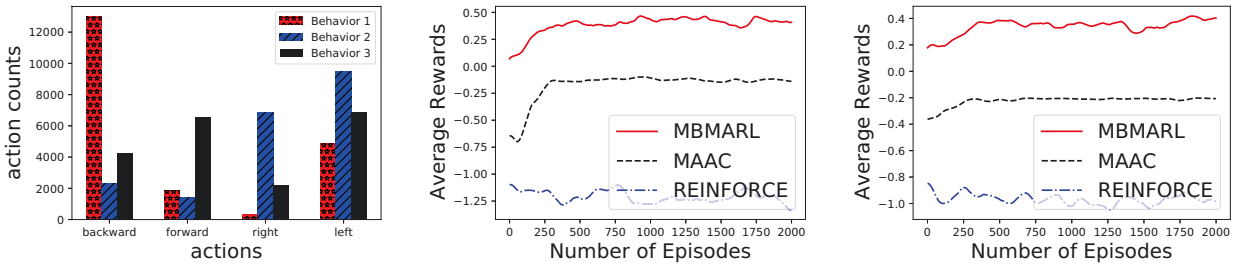
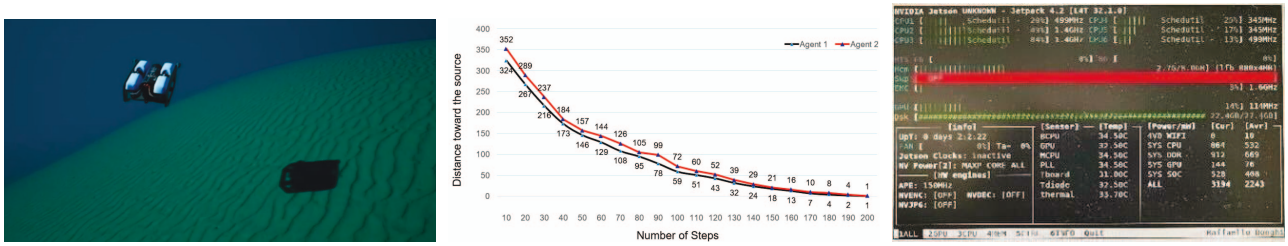


Fig. 8: Results in environment 2, where the first figure is the action distribution of different behaviors. Average rewards over the agents at each step vs. Number of training episodes when Number of agents = 4 and 6 are presented respectively in the second and the third picture. The probability of communication failure for MBMARL and MAAC is 0.5 and the behavior switch interval for MBMARL is 20.



(a) (b) (c)

Fig. 9: (a) The underwater environment in the simulator. (b) The figure describes the result of the distance between the target and the BlueROVs during the search process. (c) The result of the hardware test using NVIDIA Jetson TX2.

Fig. 10: The field experiment in the parking lot. The two solid line circles in both figures indicate that the only target could be within this region with a 40% probability. The dashed arcs represent the probability field of each region. The black dot is the starting position of the agent. The solid red line is the trajectory of our framework, and the solid black is the trajectory of the single-behavior framework.

to find the target with different numbers of behavior policies. Across 20 independent tests, the targets are found in 54.7, 39.8, and 32.4 steps with 3, 4, and 5 behavior policies, respectively. These results demonstrate the generalizability of MBMARL beyond underwater environments, as long as the information received by agents indicates a correlation with the target state in an MDP, making it applicable for informed search.

C. Field Experiments

We define the environment with two regions where the target could be of equal probability. Each region can be regarded as a source of a probability field. Influenced by a region, each location has a value v. The closer the location to this region, the larger v. As a result, the value at each location influenced by the two adapted regions is $v = v_b + v_g$, where v_b and v_g are the values caused by the blue and green regions, respectively. The target is in the blue circle region, which is not known by the agent. We expect the agent to travel to the blue region.

The performance of the single behavior depends on the starting position of the agent. If the starting position is closer to the blue region in the first scenario, a single behavior can find the blue region as well as the multi-behavior framework. The steps to find the region are almost the same: 134 (single-behavior) vs. 137 (multi-behavior). However, in the second scenario, where the starting point is closer to the green field, with only one behavior, the agent is trapped because once it leaves the green region, v decreases, since in the area near the green region, the region field is stronger. To increase the reward, the agent returns to the green region and cannot travel to the blue region. With multiple behaviors, the agent initially gets lost in the area near B. However, once the behavior switch interval is met and the agent can select a more appropriate behavior compared to the previous one, it moves out of the area near B towards A.

D. Emulation Experiments

To test the framework, we also performed an emulation. The selected application is to find the source of the plumes in an underwater environment. The underwater environment is built in Unreal Engine 4.26 with the ocean and landscape shown in Fig. 9(a). We chose a spot on the underwater land as the source of the plume. To allow the AUVs to work, we rendered the distribution model of the plume in this underwater environment so that the AUVs knew the concentrations. Each AUV can move along the x, y, and z axes at each step. The step size can change from 1 to 10. We placed a particle system at the source position and a light source to make it visible. AUVs are controlled by the open-source AirSim simulator [32]. By importing the BlueROV model into that, we can control it with Python. This video shows the process of two BlueROVs searching along the concentration to the plume source. These two BlueROVs are working together. Fig. 9(b) shows the change in distance between the two agents and the

target with the number of steps. We also tested the algorithm on an embedded AI computing device, Nvidia Jetson TX2, as shown in Fig. 9(c). We ran another 500 simulations of plume source search tasks to test the device more comprehensively. The working power consumption is approximately 3 watts. However, we must admit that in real experiments it will increase due to the Wi-Fi connection and the sensors working, but the increment will not be significant. As a result, the power consumption level is acceptable to work on a robot for real experiments. In 500 different search missions, we use 6 agents with a communication failure probability of 0.8 and a behavior change interval of 2. In total, those 500 tasks took around 39,000 seconds. Each task takes an average of 78.7 seconds. Each task takes an average of 21.3 steps, and each step takes an average of 3.7 seconds to finish.

IV. Conclusion and Future Work

We proposed an MBMARL framework, where agents possess multiple behaviors for adaptive actions in a single state. Our experimental results demonstrated that: (i) MBMARL outperforms compared frameworks in terms of rewards, convergence time, and steps to find the target in complicated environments with uncertain information; (ii) more behaviors provide agents with diverse strategies for handling sparsity and local optimum; (iii) the dependency on communication of MBMARL is lower than compared frameworks.

For future work, we plan to: (a) Utilize Offline RL to build models for explainable behaviors; (b) Conduct fair comparisons between MBMARL and MAAC by training the latter offline; (c) Explore methods for generating structured noise in R&ESN using techniques such as Variational Auto-Encoder (VAE); (d) Enhance and update the behavior model in real-time during online tests as needed.

References

[1] A. Quattrini Li, R. Cipolleschi, M. Giusto, and F. Amigoni, "A semantically-informed multirobot system for exploration of relevant areas in search and rescue settings," *Autonomous Robots*, vol. 40, pp. 581–597, 2016.

[2] S. Choudhury, J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. Scherer, "Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4207–4214, IEEE, 2016.

[3] Z. Yao, X. Li, B. Lang, and M. C. Chuah, "Goal-lbp: Goal-based local behavior guided trajectory prediction for autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[4] D. Uvaydov, D. Unal, K. Enhos, E. Demirors, and T. Melodia, "Sonair: Real-time deep learning for underwater acoustic spectrum sensing and classification," in *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pp. 9–16, 2023.

[5] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[6] O. O. Martins, A. A. Adekunle, O. M. Olaniyan, and B. O. Bolaji, "An improved multi-objective a-star algorithm for path planning in a large workspace: Design, implementation, and evaluation," *Scientific African*, vol. 15, p. e01068, 2022.

[7] E. F. Krause, "Taxicab geometry," *The Mathematics Teacher*, vol. 66, no. 8, pp. 695–706, 1973.

[8] J. C. Gower, "Properties of euclidean and non-euclidean distance matrices," *Linear algebra and its applications*, vol. 67, pp. 81–97, 1985.

[9] Z. Meng, A. Williams, P. Liau, T. G. Stephens, C. Drury, E. N. Chiles, X. Su, M. Javanmard, and D. Bhattacharya, "Development of a portable toolkit to diagnose coral thermal stress," *Scientific reports*, vol. 12, no. 1, p. 14398, 2022.

[10] S. Liu and M. Zhu, "Distributed inverse constrained reinforcement learning for multi-agent systems," *Advances in Neural Information Processing Systems*, vol. 35, pp. 33444–33456, 2022.

[11] G. Chen, "A new framework for multi-agent reinforcement learning–centralized training and exploration with decentralized execution via policy distillation," *arXiv preprint arXiv:1910.09152*, 2019.

[12] Z. Zhang, J. Jiang, J. Wu, and X. Zhu, "Efficient and optimal penetration path planning for stealth unmanned aerial vehicle using minimal radar cross-section tactics and modified a-star algorithm," *ISA transactions*, vol. 134, pp. 42–57, 2023.

[13] S. Alshammrei, S. Boubaker, and L. Kolsi, "Improved dijkstra algorithm for mobile robot path planning and obstacle avoidance," *Comput. Mater. Contin*, vol. 72, pp. 5939–5954, 2022.

[14] S. Erke, D. Bin, N. Yiming, Z. Qi, X. Liang, and Z. Dawei, "An improved a-star based path planning algorithm for autonomous land vehicles," *International Journal of Advanced Robotic Systems*, vol. 17, no. 5, p. 1729881420962263, 2020.

[15] Z. Yao, J. Xu, S. Hou, and M. C. Chuah, "Cracknex: a few-shot low-light crack segmentation model based on retinex theory for uav inspections," *arXiv preprint arXiv:2403.03063*, 2024.

[16] L. Wang, S. Pang, and G. Xu, "3-dimensional hydrothermal vent localization based on chemical plume tracing," in *Global Oceans 2020: Singapore–US Gulf Coast*, pp. 1–7, IEEE, 2020.

[17] E. Heiden, L. Palmieri, S. Koenig, K. O. Arras, and G. S. Sukhatme, "Gradient-informed path smoothing for wheeled mobile robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1710–1717, IEEE, 2018.

[18] Z.-l. Meng, "Design and implementation of sound tracking multi-robot system in wireless sensor networks," in *2017 First International Conference on Electronics Instrumentation & Information Systems (EIIS)*, pp. 1–6, IEEE, 2017.

[19] Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, K. Guo, and H. Xie, "Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning," *IEEE access*, vol. 7, pp. 39974–39982, 2019.

[20] A. Alagha, R. Mizouni, J. Bentahar, H. Otrok, and S. Singh, "Multi-agent deep reinforcement learning with demonstration cloning for target localization," *IEEE Internet of Things Journal*, 2023.

[21] S. Liu and M. Zhu, "Learning multi-agent behaviors from distributed and streaming demonstrations," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[22] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[23] Y. Yang, X. Ma, C. Li, Z. Zheng, Q. Zhang, G. Huang, J. Yang, and Q. Zhao, "Believe what you see: Implicit constraint approach for offline multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10299–10312, 2021.

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

[25] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International Conference on Machine Learning*, pp. 2052–2062, PMLR, 2019.

[26] J. M. Joyce, *Kullback-Leibler Divergence*, pp. 720–722. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[27] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[28] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, PMLR, 2016.

[29] M. Chevalier-Boisvert, L. Willems, and S. Pal, "Minimalistic gridworld environment for gymnasium," 2018.

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[31] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[32] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017.