C3F: Constant Collaboration and Communication Framework for Graph-Representation Dynamic Multi-Robotic Systems

Hongda Jia, Zijian Gao, Cheng Yang*, Bo Ding, Yuanzhao Zhai, Huaimin Wang

***Abstract*— Deep reinforcement learning (DRL) methods have been widely applied in distributed multi-robotic systems and successfully realized autonomous learning in many fields. In these fields, robots need to communicate and collaborate with other robots in real time, and reach agreed cognition for task assignment, which puts high requirements on efficiency and stability. However, robots may often get damaged even crash in complex environments, and have to be dynamically substituted. It seems not robust enough for most existing DRL works to make new robots fast adapt to current team policies, causing performance degradation. In this work, we get inspired by the genetic mechanism of social animals' instincts, and propose a robust multi-robotic collaboration and communication framework, *C3F*. It introduces graph-based representation to discover more features on the relevance among robots, and takes advantage of meta learning mechanism to conclude the general meta policy. When some robots crash and get replaced by new ones, this meta policy will be reused to guide new robots on how to quickly follow the existing collaboration and communication rules, and fast adapt to their roles in the team. The experiments on both the Webots simulator and the Starcraft II platform indicate that our methods have better performance compared with some SOTA methods, showing strong robustness and remarkable adaptability to the dynamic substitution in multi-robotic systems.**

I. INTRODUCTION

In distributed multi-robotic systems, robots can collaborate to solve problems in parallel. They do not rely on centralized management and have been verified for outstanding performance in resource utilization and learning efficiency [1]. Besides, distributed multi-robotic systems possess unique advantages in stability and fault tolerance [2][3].

In some real-world tasks, we notice robots may suffer from failure problems and get dynamically changed. For example, terrible collisions often occur when many robots are exploring and mapping a new environment in collaborative navigation tasks (as Fig. 1). The broken robots have to quit and new robots will dynamically join. However, the new robots may be different from the broken ones in type and size. Similar situations always happen in many robotic exploration tasks in unknown environments, such as planetary exploration, military reconnaissance, and disaster rescue.

Benefiting from the robustness of distributed systems, individual-level failures can hardly interfere with other robots. For example, if one robot in Fig. 1 suddenly crashes, the others can still reach their intended target areas.

However, from the collective-level perspective, the overall performance of multi-robotic systems may still suffer from negative impacts, even though the crashed robots are removed and replaced by new ones. One main reason is that, when the new robots start to work, it may be unsuitable for them to directly reuse the prior policy models. It is due to that the structure and ability of the new robots are usually not exactly the same as the crashed ones. So the prior collaboration and communication policy of the whole system may get changed when new robots join as substitution, and the well-trained policies of crashed robots may no longer suit the new ones. Besides, there is a more extreme case that crashed robots may be too broken to get their prior models. So the new robots usually have to learn from scratch about which robot and when to communicate with, as well as fast adaption to the existing collaboration policy on task assignment. This challenge may pause the overall learning progress and cause a reduction in performance.

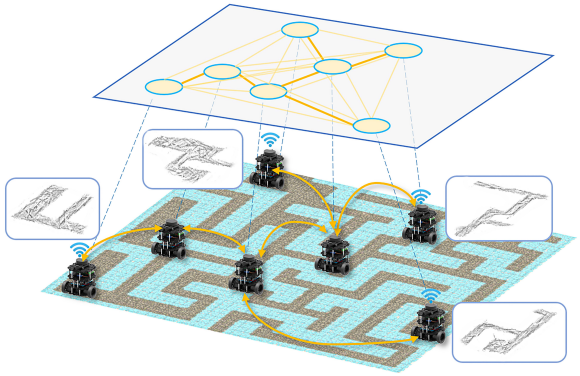


Fig. 1. The diagram of a distributed multi-robotic system in collaborative navigation tasks. Robots explore and map the surrounding environment, then exchange their own sketch map with others. These communication processes can be modeled in the form of a weighted undirected complete graph, where the edge weights can be used to represent communication frequency. From this graph, the communication priority of each robot can be explicitly displayed.

We notice that in social animal colonies, such as bees and ants, individuals are always in dynamic substitution and balance. Although individuals are not exactly consistent in type and size, most newborn individuals can quickly learn how to communicate with partners and integrate into existing colonies. Researches indicate that it is mainly because genes can record innate instincts, so that some basic and common skills can be inherited from one generation to another. These skills come from the accumulated experience of ancestors,

The authors are with the College of Computer, National University of Defense Technology, Changsha, China. E-mails: {jiahongda17, gaozijian19, yangcheng12, dingbo, yuanzhaozhai}@nudt.edu.cn, whm_w@163.com

This work was supported by Major Science and Technology Innovation 2030 "New Generation Artificial Intelligence" project under Grant 2020AAA0104803.

including social collaboration and communication ability. When new individuals are born, they can have some basic life skills by heredity and fine-tune them by postnatal learning, instead of completely learning from scratch.

Inspired by this genetic mechanism, we apply meta reinforcement learning (MRL) methods to multi-robotic systems. MRL aims at learning how to learn efficiently, and has been proven to have many common points with biological learning mechanisms [4]. Especially, we further consider the interconnected behaviors in the social animal colonies, and introduce graph structures to emphatically describe the collaboration relevance and communication frequency among robots. Taking Fig.1 as an instance, robots can be regarded as vertexes, and the distance and communication frequency between two robots can measure the weight of the corresponding edge. Then the whole multi-robotic system can be modeled as a weighted undirected complete graph structure.

Based on the above analysis, we propose a robust framework, *C3F*, to achieve constant and stable collaboration and communication for distributed multi-robotic systems. It takes advantage of coordination graphs for graph-based representation, and meta learning mechanism for concluding general rules. Firstly, robots use deep coordination graph [5] to learn an effective policy from some sub-tasks of one task sequence. Then, these policies will be integrated into one general meta policy by meta learning mechanism, which is similar to how genes record the accumulated experience from predecessors. Once some robots break down, new ones take over them and get basic skills from this meta policy, just like animals having innate instincts from heredity. After that, new robots have good initial policies and can fast adapt to the new system by fine-tuning only a few episodes, like animals effective postnatal learning.

We take experiments on the Webots simulator and StarCraft II platform. Both quantitative multi-indicator analysis and qualitative trajectory comparison provide convincing evidence, that our methods have better performance compared with some SOTA methods. These results also indicate that *C3F* is efficient and robust to the unexpected single points of failure in dynamic multi-robotic systems. The main contributions of this paper consist of:

- At the algorithm level, we make specific improvements to apply meta learning mechanism to graph-based multi-agent deep reinforcement learning methods, and achieve remarkable promotion compared with baseline works.
- At the application level, this work studies the performance degradation problem which is caused by robots dynamically quitting and being substituted by different others. It can ensure constant and stable learning for dynamic distributed multi-robotic systems.
- At the innovation level, we get inspired by how social animal colonies keep dynamic balance and constant evolution, and achieve this genetic mechanism in multi-robotic systems, providing further exploration in bionics fields.

II. RELATED WORK

A. Graph-Based Representation for Multi-Robotic Systems

In multi-robotic systems, not only each independent robot determines the stability and performance, but also the relevance among robots plays an important role. This relevance mainly consists of the distance, communication frequency, and implicit task assignment, which stands for collective-level policy. By introducing the graph-based representation, it can take both individual-level features (as vertexes) and collective-level information (as edges) into consideration.

One outstanding work on graph-based representation is the graph neural network (GNN) [6]. It has provided feasible and promising ways for node classification [7], edge prediction [8], clustering [9] and visualization [10]. Especially, edge prediction is of great significance to describe the relationship and communication between two robots.

There have been some graph-based representation works in the multi-agent reinforcement learning fields. Most of them aim to explore and exploit more potential relevance, and improve learning efficiency [11]. Graph policy gradients (GPG) [12] takes advantage of the potential graphic symmetry between robots to realize the zero-shot transfer. Besides, graph-based policy gradients can also be introduced to reduce the dimension of action space and state space. It can learn a filter to aggregate information among nearby robots [13], which greatly promotes the scalability of multi-robotic systems. Above all, it is feasible and valuable to introduce graph-based representation into multi-robotic systems.

B. Coordination Graph

The coordination graph has been one of the popular graph-based multi-agent reinforcement learning methods, and follows the CTDE (centralized training with distributed execution) architecture. It applies joint value factoring methods to graph neural networks, and successfully improves the learning efficiency of multi-robotic collaborative tasks. In multi-robotic systems, their coordination graph (CG) is based on the current state, and may dynamically change during the task process [14]. As for formalization, undirected coordination graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ models i-th robot as the vertex $v_i \in \mathscr{V}$, and generates a group of edges $e_{ij} \in \mathscr{E}$ between v_i and v_j [15]. The common factorization of the coordination graph divides action-value functions into vertex-based utility function f^v and edge-based payoff function f^e as:

$$Q(s_t, a) = \sum_{v_i \in \mathscr{V}} f^v(a^i | s_t) + \sum_{e_{ij} \in \mathscr{E}} f^e(a^i, a^j | s_t) \tag{1}$$

DCG [5] uses neural networks to approximately estimate these two functions, and successfully factors the joint value function when dealing with complex situations. Based on this, DICG [16] uses implicit inference to deal with dynamic environments, and makes the coordination graph also get changed correspondingly. Dynamic QCGraph [17] also concentrates on the dynamic changes during multi-robot reinforcement learning. It constructs dynamical coordination graphs based on real-time robot subsets, and achieves improved factorization.

The CG theory emphasizes interaction among robots by graph-based representation, providing further theoretical proof and promising application for traditional value function factoring methods. Existing works make great contributions to coordination-graph-based multi-agent reinforcement learning. However, there is still some improvement in need for specific problems, such as how to transfer this graph-based knowledge for fast adaption to new tasks.

C. Meta Learning and Meta Reinforcement Learning

Human beings have the advantage of fast adaptation to new tasks. It is mainly because they can summarize common experiences from prior tasks and reuse them to guide the following tasks [4]. Inspired by this biological cognitive principle, meta learning is proposed to realize learning to learn in the machine learning field.

Existing meta learning algorithms mainly include three categories. The first one is representing some common features as cross-task meta knowledge, such as RNN-based memory module [18], gradient prediction [19], loss function simulation [20]. Secondly, metric-based methods use kernel function to measure the relevance between each pair of samples, then take the most relevant one for transferring [21]. The third one is optimizing initial models, such as MAML [22]. It will take a good initial model at the beginning of new tasks, rather than random initialization. So it can perform well at the beginning, and achieve fast adaption just after a few training steps.

Meta reinforcement learning (meta-RL) methods introduce the above categories into the reinforcement learning field, which is proven to reveal the mechanism of dopamine and the way mammals recognize new objects [23]. MAML is a common gradient-based method, while PEARL [24] stands for context-based methods. Besides, MQL [25] successfully combines gradient-based methods and context-based methods, and takes both strengths to deal with new tasks better.

Existing multi-agent meta-RL works have made many successful explorations on improving the learning efficiency of multi-agent systems. But they consider less about the relevance between agents, and have not introduced the graph structure to represent and transfer these collective-level features of multi-agent systems. Besides, most related works have not offered convincing verification on the three-dimensional multi-robotic collaboration tasks [26] [27].

III. THE FRAMEWORK OF C3F

As shown in Figure 2, there are three main phases in *C3F*: 1) *The meta-training phase*. Robots explore sub-tasks $T_1, T_2, \ldots, T_N$ of one task sequence, and respectively learn policy $\pi_1, \pi_2, \ldots, \pi_N$ from the corresponding sub-task. These sub-tasks share similar goals but have different settings. And the training on each sub-task is an independent and complete process. 2) *The meta-updating phase*. Taking meta learning methods to integrate these policies and representing them as general meta policy π_m. 3) *The meta-testing phase*. Inheriting from this meta policy π_m, and fine-tuning specific policy for the new sub-task. The details will be described as follows.

A. Preliminaries

Markov Decision Process (MDP) Collaborative multi-agent task is a decentralized partially observable Markov decision process [28] with a tuple $< S, \mathscr{I}, \mathscr{A}, P, r, \mathscr{Z}, O, n, \gamma >$. S stands for the global environment state, and set $\mathscr{I}$ consists of n agents. At step t, the state is s_t. Agent $i \in \mathscr{I}$ takes individual action $a_t^i \in \mathscr{A}$, and the multi-agent system takes joint action $a_t = (a_t^1, a_t^2, \ldots, a_t^i, \ldots, a_t^n) \in \mathscr{A}^n$. Then the next state is $s_{t+1} \in S$, which follows the state transition probability function $P(s_{t+1}|s_t, a_t) : S \times \mathscr{A}^n \times S \to [0,1]$. Meanwhile, agents obtain the same reward function $r(s,a) : S \times \mathscr{A}^n \to R$. As for the partially observable Markov decision process (POMDP), agents only access to local observations $o \in O$ with observation function $\mathscr{Z}(s,i) : S \times \mathscr{I} \to O$, instead of global state. To find the optimal policy π, it is required to maximum the joint action-value function $Q^{\pi}(s_t, a_t) = E_{s_{t+1:\infty}, a_{t+1:\infty}}[R_t|s_t, a_t])$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ and γ denotes the discount factor.

B. The Meta-Training Phase

In *the meta-training phase*, robots will learn how to deal with a sequence of sub-tasks $T_1, T_2, \ldots, T_N$ by trial and error, then conclude the common knowledge as meta policy. These sub-tasks are consistent in main goals, but have diversity in robot types or environment settings. Taking collaboration navigation tasks for instance, all sub-tasks require robots to reach more target areas in a short time, but the team compositions of robots in T_1 are different from those in T_2. The training process in each sub-task is independent.

In this phase, the training algorithm for each sub-task basically refers to the DCG method [5] and its improved version [29]. In Fig.2, there are three modules in DCG methods including the historical trajectory encoder, utility function module, and payoff function module. These modules all involve neural networks which are connected together.

For each sub-task, DCG firstly uses the RNN-based neural networks to encode the historical trajectories h_t^i for robot i at step t. Then, the utility function module determines all the possible actions that each robot can take, and calculates the corresponding utility values. Finally, the payoff function module determines all the possible joint actions between each pair of robots, and calculates the corresponding payoff values. Based on the above steps, the total value of joint action can be calculated as (2).

$$Q^{DCG}(a|s_t) = \frac{1}{|\mathscr{V}|} \sum_{v_i \in \mathscr{V}} f^v(a^i|h_t^i) + \frac{1}{2|\mathscr{E}|} \sum_{e_{ij} \in \mathscr{E}} (f^e(a^i, a^j|h_t^i, h_t^j) + f^e(a^j, a^i|h_t^j, h_t^i)) \tag{2}$$

By belief propagation process [5], each robot can find the optimal action and estimate the maximum total value. Through minimizing the loss function (3) by gradient descent, the networks of all three modules get optimized.

$$L = \mathbb{E}[\frac{1}{t_{max}} \sum_{t=0}^{t_{max}-1} (R_t + \gamma \max_{a_t} Q^{DCG}(\cdot|s_{t+1}) - Q^{DCG}(a_t|s_t))^2] \tag{3}$$

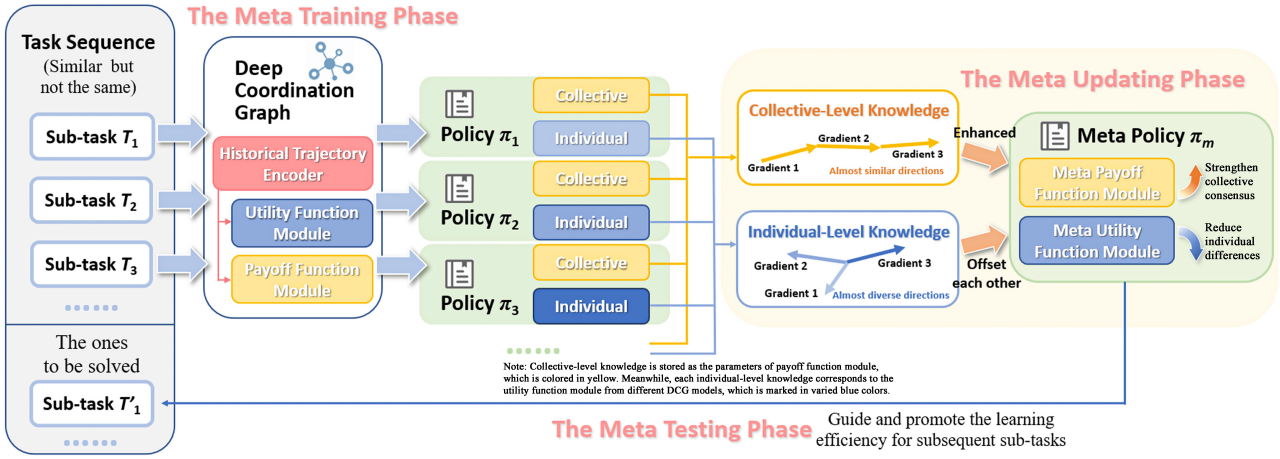


Fig. 2. The main framework of the *C3F* method, including three phases. 1) In *the meta-training phase*, the task sequence is composed of several sub-tasks that have consistent task goals but different robot settings. Robots use the deep coordination graph method to learn a policy from each sub-task. Each policy includes both collective-level and individual-level knowledge, respectively stored as the network models of the utility function module and payoff function module. 2) In *the meta-updating phase*, all these policies will be integrated into the meta-policy by the MAML method. As these sub-tasks share common overall goals, the payoff function modules of all policies have almost similar gradient updating directions, so collective-level knowledge will be constantly enhanced. In contrast, as the specific setting in each sub-task is different, the gradient of each utility function module is diversified. So individual-level knowledge from different sub-tasks will offset each other, and the differences of individuals among sub-tasks are reduced. 3) In *the meta-testing phase*, this meta policy will be used to guide robots have some instinctive basic skills in the subsequent sub-tasks of the same task sequence, improving robustness and learning efficiency.

When reaching the maximum total step or the training performance has converged, the training process for this sub-task can be finished, and the well-trained policy is stored in the form of model parameters for further calculation. This training process is the same for each sub-task, and can be independently taken in parallel. When the training process of all sub-tasks has been finished, this phase ends.

C. The Meta-Updating Phase

After *the meta-training phase*, there are many policies $\pi_1, \pi_2, \ldots, \pi_N$ generated by sub-tasks $T_1, T_2, \ldots, T_N$. Every policy is saved as a group of network models, and covers two kinds of knowledge: payoff function module for the collective-level one, and utility function module for the individual-level one. Then *the meta-updating phase* aims to abstract common knowledge and represent them as the meta policy. It can be an analogy with genes recording the common instincts, which comes from the accumulative experiences of predecessors.

The meta policy π_m is composed of the meta utility function module and the meta payoff function module. Their network structures are the same as that of the utility function module and that of the payoff function module respectively.

The detailed update way of *C3F* refers to the widely-applied meta learning method, MAML [22]. Besides, we design differentiated meta-updating learning rates to better suit the two-module networks in the DCG method. For policy π_i from the sub-task T_i, the parameters of all networks in the utility function module are denoted as θ_{T_i}. And the parameters θ_m of the meta utility function module can be updated as (4) and (5).

$$\theta'_{T_i} = \theta_m - \alpha \nabla_\theta L(\theta_{T_i}) \tag{4}$$

$$\theta_m \leftarrow \theta_m - \beta^{(u)} \nabla_\theta \sum_{T_i \sim p(T)} L(\theta'_{T_i}) \tag{5}$$

where θ'_{T_i} is the adapted parameter, and $L(\theta_{T_i})$ represents the loss function on the sub-task T_i. The sub-task T_i obeys the distribution $p(T)$ of this task sequence. α is the step size for updating specific tasks, and $\beta^{(u)}$ is that for meta-updating.

The meta payoff function module gets updated from the payoff function modules of policies $\pi_1, \ldots, \pi_N$. The updating way also follows (4) and (5), but the meta-updating step size is replaced by $\beta^{(p)}$. $\beta^{(p)}$ is different from $\beta^{(u)}$, and can vary with different sub-tasks. Considering the collective consensus will be more obvious in the meta policy but the individual difference will gradually disappear, the $\beta^{(p)}$ should be a little smaller than $\beta^{(u)}$ for the stable training process.

This phase can be intuitively explained in the following two aspects. On the one hand, as these sub-tasks are from one task sequence, it can be assumed that they have similar requirements for robots on collective-level cognition, such as overall goals and communication rules. For example, the main goal of navigation tasks is to explore more areas in a short time, which is common and similar in the following sub-tasks with different environments. Besides, in distributed multi-robotic systems, the task assignment always follows some general principles, such as the proximity principle or greedy strategy. These kinds of knowledge can be concluded for transfer.

On the other hand, the individual-level knowledge from each sub-task may be different, because robots are required to deal with diverse situations in each sub-task. For example, in one navigation sub-task, robots learn how to avoid obstacles at first and then find a way to the nearest target area. But in another sub-task where the position of this robot and environment settings get changed, the robots can directly go to the targets without rounding obstacles. Besides, the relative positions between robots also dynamically change. For a robot, another robot that was closely connected in the

last sub-task, may be very remote and rarely connected in the next sub-task. These phenomena indicate that individual-level knowledge from prior tasks seems unsuitable for new robots in subsequent tasks.

By the "summation and average" process of *the meta-updating phase*, the collective consensuses from all payoff function modules are enhanced, but the individual differences among the utility function module of each policy are reduced. Therefore, the meta policy is universal and suitable for making robots have remarkable adaptability on collaboration and communication when some individuals dynamically change.

D. The Meta-Testing Phase

As shown in the bottom of Figure 2, the multi-robotic system is required to learn how to solve the upcoming sub-task $T_1', T_2', \ldots, T_N'$, which is similar but not identical to the above $T_1, T_2, \ldots, T_N$. For instance, there are the same robots and the same environment in navigation tasks T_1 and T_1', but one of the robots suddenly crashes in T_1' and has to be replaced with another robot of different types.

At the beginning of sub-task T_i', the neural networks of the payoff function module and the utility function module will load corresponding network parameters from meta policy, instead of random initialization. It is just like newborn animals getting basic survival skills by heredity. Then, all networks will get fine-tuned by using the DCG method again to learn some task-specific knowledge, which is similar to animals' postnatal learning. Finally, the performance of robots can efficiently tend to optimization after fine-tuning a few episodes. At the same time, the centralized training process will end and robots only take distributed execution by their own observation and policy.

The network structures and detailed workflow of fine-tuning are similar to that in *the meta-training phase*. In step t, robot i gets local observation and the received messages, and uses the historical trajectory encoder to get historical trajectories h_t^i. Then the utility function module generates individual action value $f^v(a_t^i|h_t^i)$, and payoff function module generates joint action values $f^e(a^i, a^j|h_t^i, h_t^j)$ and $f^e(a^j, a^i|h_t^j, h_t^i)$, followed by minimizing the loss function as (3).

In *the meta-testing phase*, the learning efficiency during fine-tuning can be used to measure the performance of each method. That is to say, effecitve methods always require few episodes for fine-tuning and fast adaption to new sub-tasks.

IV. EXPERIMENT AND RESULTS

There are two kinds of experiments in this section. First, we set multi-robotic collaborative navigation tasks on the Webots[30] platform, and provide quantitative results and intuitive trajectory comparison for qualitative analysis. Then, we make another group of experiments on the StarCraft II [31] platform, in order to further verify the robustness under the more complex type-changing challenge. These experiments are based on Ubuntu 22.04 operating system and Pytorch 1.13.1 framework. CUDA 11.7 and cuDNN are used to improve training efficiency.

A. Experiment on Robotic Simulator

We conducted experiments on Webots, one 3D robotic simulator. Taking an example in Fig.3, there are two types of robots, including TurtleBot3 and E-puck, which are different in size, speed, and other attributes. They are required to avoid obstacles (i.e. sofa, oil barrels, and plants), and collaboratively explore as many targets (marked with the flags each of which has a base) as possible. This environment is discrete. At every step, each robot can choose one of the four directions (up, down, left, and right) to move or take no action. All robots can get the position of all targets from a global view, while they detect nearby obstacles and other robots by their loaded lidar. Besides, communication is also allowed between two robots.

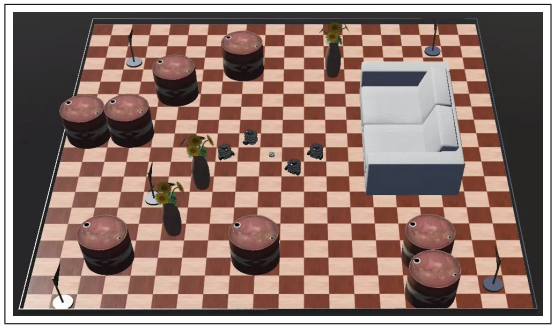


Fig. 3. The Webots simulation environment of task IV. There are four TurtleBot 3 robots, one E-puck robot, five targets, and some obstacles. Robots are required to collaboratively explore as many targets as possible, and avoid colliding with obstacles.

There are four task sequences, and the detailed settings are shown in the upper part of table I. Each task sequence consists of three meta-training tasks and one meta-testing task. Robots first learn 200 episodes in each of the meta-training tasks, then the current model will be tested in another different meta-testing task. During this process, all the network structures are fixed. At the beginning of each episode, robots are reset back to the initial positions, while targets are reset to different random positions. At every step, every robot decides its own actions according to the current information, then gets a reward as (6) from the environment for updating. When reaching the maximum step, this episode ends and the next one starts.

$$R = R_0 - \frac{1}{N_{max}} \sum_{t=1}^{N_{max}} \left(\sum_{j=1}^{N_{tar}} \min_i dist(i, j|t) + \lambda N_{col}(t) \right) \qquad (6)$$

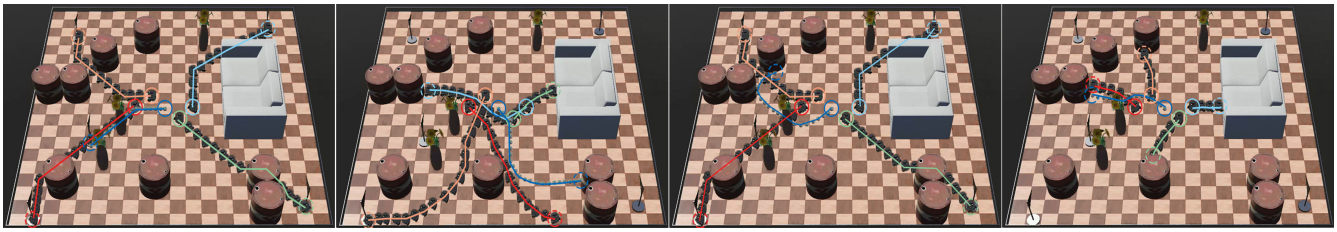
where $dist(i, j|t)$ denotes the Euclidean distance between the i-th robot and the j-th target at step t. R_0 denotes the basic reward for every robot's legal operation. N_{tar} is the number of targets, and N_{max} is the maximum step number of one episode. $N_{col}(t)$ is the number of collisions between a robot and an obstacle or between two robots during step t, and λ is a discount factor. In the following four task sequences, $N_{max} = 100$, $R_0 = 5000$, and $\lambda = 200$.

As the compared groups, we choose several multi-agent reinforcement learning baselines, including QMIX, QTRAN, DCG, and UPDeT[32]. All methods have the same 200 training episodes in each task of *the meta-training phase*, and

TABLE I

THE TASK SETTINGS AND AVERAGE PERFORMANCE OF DIFFERENT METHODS ON WEBOTS SIMULATOR

	Task Sequence 1				Task Sequence 2				Task Sequence 3				Task Sequence 4			
Phase	Three meta-training tasks			The meta-testing task	Three meta-training tasks			The meta-testing task	Three meta-training tasks			The meta-testing task	Three meta-training tasks			The meta-testing task
Task No.	I	II	III	IV	I	III	IV	II	V	VI	VII	VIII	V	VII	VIII	VI
TurtleBot3	1	2	3	4	1	3	4	2	1	3	5	7	1	5	7	3
E-puck	4	3	2	1	4	2	1	3	7	5	3	1	7	3	1	5
Targets	5				5				8				8			
The average computation time (ACT) spent at each episode of three meta-training tasks & the performance of the 50th, 100th, and 150th episodes in the meta-testing task of tha same task sequence.																
Episode	ACT(s)	50th	100th	150th	ACT(s)	50th	100th	150th	ACT(s)	50th	100th	150th	ACT(s)	50th	100th	150th
QMIX	31.49	1064.19	2225.73	3009.30	30.76	946.70	1619.25	2234.93	40.23	1397.20	2693.91	4137.38	40.31	1183.31	2263.24	3443.22
QTRAN	31.98	923.39	1713.71	2755.35	31.78	905.97	1189.17	1959.73	40.60	1309.93	2525.12	3982.13	41.34	1097.58	1836.48	3816.28
DCG	32.76	808.13	1890.04	2765.59	31.45	816.97	1412.61	1946.64	40.84	1332.23	2820.45	3363.65	40.01	1000.80	2206.34	3633.45
UPDeT	33.11	1451.03	2564.05	3278.60	34.24	1370.18	1746.43	1912.38	43.74	1769.20	**3179.08**	3909.59	44.01	1452.26	2234.87	**4210.62**
C3F(ours)	32.31	**1701.75**	**2567.76**	**3313.98**	32.77	**1470.16**	**2010.68**	**2451.23**	41.46	**2025.28**	3074.34	**4338.03**	42.20	**1745.35**	**2608.34**	4195.70



(a) C3F (b) QMIX (c) UPDeT (d) DCG

Fig. 4. The navigation trajectory comparison among four methods at the 50th episode of *Task IV*, *Task Sequence 1*. There are four sub-figures respectively standing for C3F, QMIX, UPDeT, and DCG. In each sub-figure, robots are marked with different colors: the solid-line circle denotes the starting point, while the dotted-line circle for the final position. During the navigation process, the real-time position of each robot will be recorded at every step. By drawing these positions in turn, the robot's discrete position sequence can be used to approximately estimate its navigation trajectory (marked with a colored line). These trajectories of all robots can intuitively measure the overall performance.

the well-trained models will be kept as the initial model at the beginning of *the meta-testing phase*. The only difference is that our method has *the meta-updating phase*. The results are shown by qualitative trajectory comparison (as Fig. 4) and quantitative learning performance (as table I).

In the lower part of table I, the average computation time (ACT) spent at each episode in *the meta-training phase* and the average performance of *the meta-testing phase* are shown by repeating 30 times. The performance is represented by the rewards at the 50th, 100th, and 150th episodes respectively. The best performance is emphasized in bold font.

The results illustrate that all methods can gradually learn a good task assignment policy and collaboratively explore as many targets as possible, which leads to better performance. But on the whole, our method outperforms almost all the time, especially at the beginning episodes. The common collective-level meta policy of *C3F* can guide robots on how to assign tasks and approach their intended targets, so that they can fast adapt to the new multi-robotic system. Besides, the similar ACTs of these methods also prove our method does not obviously increase training time for meta-updating.

Besides, we take the *Task IV*, *Task Sequence 1* for example, and present intuitive navigation trajectory comparison among four methods in Fig.4. At the 50th episode of *the meta-testing phase*, robots' positions at every step are recorded. For every robot, its discrete positions can be used to approximatively estimate the navigation trajectory. And in each sub-figure, the trajectories of different robots are marked in different colors. Please refer to the attached video for more details.

From Fig. 4, we can find our *C3F* in sub-figure (a) can help robots quickly learn to collaboratively approach all targets without collision. In sub-figure (c), four TurtleBot 3 robots in UPDeT find the clear division of work and plan optimal paths to different targets, while only the E-puck has not learned where to go and finally collides with an obstacle. QMIX in sub-figure (b) has not made robots reach an agreed task assignment and learned how to effectively avoid obstacles, so that there are intended-target conflicts as well as collisions. Unfortunately, all the robots of DCG in sub-figure (d) suffer from terrible collisions not long after the task starts.

It indicates that our method can make the multi-robot system with new composition has strong adaptability. These robots can learn an effective policy on task assignment, navigation, and obstacle avoidance just after a few episodes of training. In contrast, robots of other methods learn only a portion of even almost none of these skills after the same training time, which leads to bad performance such as collisions or intended-target conflicts.

B. Experiments on StarCraft II Platform

Fig. 5. One sub-task in the StarCraft II experiment platform. The left red team is our alliance which has one stalker and four zealots. The right blue team is the enemy team which has two stalkers and three zealots. Our team aims to destroy more enemies with minimal alliance loss.

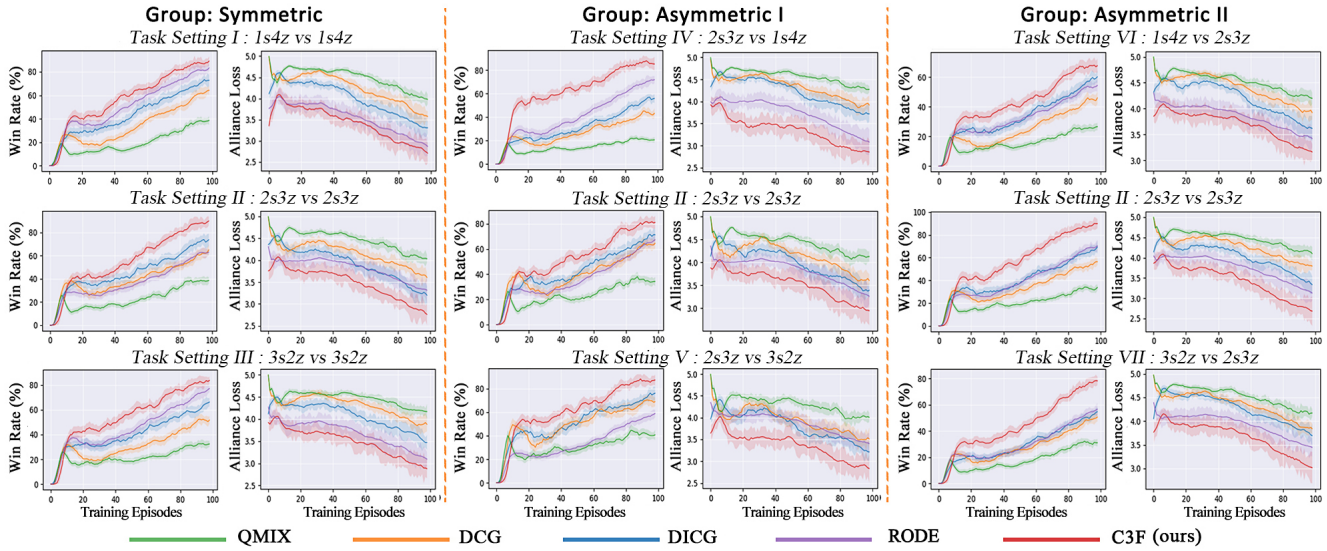


Fig. 6. The average experiment results and 95% CI of *the meta-testing phase* on StarCraft II. There are three groups from left to right, separated by orange dotted lines. Each group involves three task settings, and the performance of each task setting is measured by win rate and alliance loss. In the specific task situation and indicator, the sub-figure shows the performance (y-axis) of five methods with training episodes (x-axis). Each method is marked with a different color. The higher win rate and smaller alliance loss mean better performance.

StarCraft II is one of the typical real-time strategy games, and SMAC provides several pre-set multi-agent task settings as well as custom options. There are two teams consisting of stalkers and zealots, and the composition of each team is varied for each sub-task. Taking Fig.5 as an example, two teams are placed on the opposite side of a closed area. The left red team has five alliance agents, including one stalker and four zealots (abbreviation: 1s4z), which are controlled by the player. The right blue team has two stalkers and three zealots (abbreviation: 2s3z), which follows the policy of game AI and no longer gets updated.

In the distributed execution phase, each agent can observe and communicate with other agents whose relative distance is no more than 9 units. These observations include the relative coordinate, health, shield, and agent type, as well as the last action of alliance agents. However, the above information of all agents can be obtained in the centralized training phase. At every step, each agent can independently decide to move to a nearby discrete location, attack (or heal) other agents, or take no action. Considering agents can only hear from their teammates within a range of 9 units, the real-time topology of communication in the multi-agent system may dynamically change during the task process.

The main task goal for the alliance is to cause more damage to enemies, even completely eliminate them. Besides, they are also required to protect themselves and reduce alliance HP loss. Alliance agents continuously explore and learn to defeat enemies episode by episode, and each episode consists of at most 10000 steps. Once either team is completely eliminated or the maximum step is reached, the current episode ends and the next one starts. At the beginning of each episode, both teams are reset back to the initial state. The reward at each step is based on the difference that enemy HP loss subtracts alliance HP loss, which may be a negative value. Besides, alliance agents will get extra rewards if they eliminate all enemies, while extra punishment for being completely eliminated.

As shown in Fig.6, we set three groups in this experiment: Symmetric, Asymmetric I, and Asymmetric II. The symmetric task settings mean that both teams have the same type of agents, while asymmetric task settings do not. Each group can be regarded as one task sequence, including three sub-tasks with similar goals but different settings. When one of the sub-tasks is selected as the target sub-task in *the meta-testing phase* (marked in the corresponding sub-title), the other two sub-tasks in the same group will be the pre-training sub-tasks in *the meta-training phase*. Each sub-task will be selected in turn for cross-verification, so there are nine experiments of three groups in total.

It takes the QMIX, DCG, DICG, and Rode[33] methods as baselines, and each method has the same experiment conditions, including the training episodes in *the meta-training phase*. The well-trained networks of baselines are reused as the initial model at the beginning of *the meta-testing phase*, while *C3F* uses the meta policy model. The performance of different learning methods is verified by win rate and alliance HP loss. The higher win rate and lower alliance HP loss mean better performance. In *the meta-testing phase*, the changing trend of these two indicators with training episodes can measure the performance of methods. We repeat these experiments 100 times, and obtain their average performance and 95% confidence interval. The results are shown in Fig.6.

The results illustrate that all the methods can gradually learn how to get higher win rates and less alliance loss, but our method outperforms almost all the time. Besides, our method can quickly form some intuitive team strategies, such as focusing on attacking one enemy at the same time for numerical superiority. These quantitative indicators and qualitative analysis prove that *C3F* can help agents quickly adapt to most sub-tasks in *the meta-testing phase*, whether

it is symmetric or asymmetric. One possible explanation is that, although the agent types in *the meta-training phase* are different from that in *the meta-testing phase*, the prior individual-level knowledge may be unsuitable for the new agents. But some collective-level policies, such as task assignment, may still work. It can help agents quickly reach agreements on roles and responsibilities.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a constant and stable collaboration and communication framework, *C3F*, to deal with the performance degradation problem in dynamic multi-robotic systems. It applies meta learning mechanism to the deep coordination graph method, and concludes general collective-level knowledge and de-differentiated individual-level knowledge. This knowledge can be transferred and reused in the subsequent tasks, and guide the new robots fast adapt to their roles and task assignments. The experiment results provide convincing evidence that our methods can make robots have significant advantages in overall performance, compared with other baselines.

There are still some issues to be concerned about in future works. We follow the basic assumption of the traditional coordination graph, so the current *C3F* framework is only suitable for the pair-wise collaboration now. Although this pair-wise way is fundamental and can deal with most cases in collaborative navigation tasks, the collaboration among multiple agents also needs more concerns in the future to reveal more complex relevance in reality. Besides, how to apply it to practical tasks is our long-term interest and focus.

REFERENCES

[1] T. Cieslewski, S. Choudhary, and D. Scaramuzza, "Data-efficient decentralized visual slam," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 2466–2473.

[2] L. Paull, G. Huang, M. Seto, and J. J. Leonard, "Communication-constrained multi-auv cooperative slam," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 509–516.

[3] P. Schmuck and M. Chli, "Multi-uav collaborative monocular slam," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3863–3870.

[4] R. Dubey, P. Agrawal, D. Pathak, T. Griffiths, and A. Efros, "Investigating human priors for playing video games," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1349–1357.

[5] W. Böhmer, V. Kurin, and S. Whiteson, "Deep coordination graphs," in *International Conference on Machine Learning*. PMLR, 2020, pp. 980–991.

[6] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2. IEEE, 2005, pp. 729–734.

[7] H. Park and J. Neville, "Exploiting interaction links for node classification with deep graph neural networks." in *IJCAI*, 2019, pp. 3223–3230.

[8] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, "Neural relational inference for interacting systems," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2688–2697.

[9] X. Xiao and L. Wei, "Robust subspace clustering via latent smooth representation clustering," *Neural Processing Letters*, vol. 52, no. 2, pp. 1317–1337, 2020.

[10] W. L. Chang, K. M. Tay, and C. P. Lim, "A new evolving tree-based model with local re-learning for document clustering and visualization," *Neural Processing Letters*, vol. 46, no. 2, pp. 379–409, 2017.

[11] J. Jiang, C. Dun, T. Huang, and Z. Lu, "Graph convolutional reinforcement learning," *International Conference on Learning Representations (ICLR)*, pp. 1–13, 2020.

[12] A. Khan, E. Tolstaya, A. Ribeiro, and V. Kumar, "Graph policy gradients for large scale robot control," in *Conference on robot learning*. PMLR, 2020, pp. 823–834.

[13] A. Khan, V. Kumar, and A. Ribeiro, "Large scale distributed collaborative unlabeled motion planning with graph policy gradients," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5340–5347, 2021.

[14] C. Guestrin, S. Venkataraman, and D. Koller, "Context-specific multiagent coordination and planning with factored mdps," in *AAAI/IAAI*, 2002, pp. 253–259.

[15] C. Guestrin, M. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *ICML*, vol. 2. Citeseer, 2002, pp. 227–234.

[16] S. Li, J. K. Gupta, P. Morales, R. Allen, and M. J. Kochenderfer, "Deep implicit coordination graphs for multi-agent reinforcement learning," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 2021, pp. 764–772.

[17] C. Siu, J. Traish, and R. Y. Da Xu, "Dynamic coordination graph for cooperative multi-agent reinforcement learning," in *Asian Conference on Machine Learning*. PMLR, 2021, pp. 438–453.

[18] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International conference on machine learning*. PMLR, 2016, pp. 1842–1850.

[19] T. Xu, Q. Liu, L. Zhao, and J. Peng, "Learning to explore via meta-policy gradient," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5463–5472.

[20] R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. Jonathan Ho, and P. Abbeel, "Evolved policy gradients," *Advances in Neural Information Processing Systems*, vol. 31, pp. 5405–5414, 2018.

[21] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, pp. 3630–3638, 2016.

[22] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1126–1135.

[23] J. X. Wang, Z. Kurth-Nelson, D. Kumaran, D. Tirumala, H. Soyer, J. Z. Leibo, D. Hassabis, and M. Botvinick, "Prefrontal cortex as a meta-reinforcement learning system," *Nature neuroscience*, vol. 21, no. 6, pp. 860–868, 2018.

[24] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *International conference on machine learning*. PMLR, 2019, pp. 5331–5340.

[25] R. Fakoor, P. Chaudhari, S. Soatto, and A. J. Smola, "Meta-q-learning," *International Conference on Learning Representations (ICLR)*, pp. 1–17, 2020.

[26] D. K. Kim, M. Liu, M. D. Riemer, C. Sun, M. Abdulhai, G. Habibi, S. Lopez-Cot, G. Tesauro, and J. How, "A policy gradient algorithm for learning to learn in multiagent reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5541–5550.

[27] M. Kayaalp, S. Vlaski, and A. H. Sayed, "Dif-maml: Decentralized multi-agent meta-learning," *IEEE Open Journal of Signal Processing*, vol. 3, pp. 71–93, 2022.

[28] F. A. Oliehoek, C. Amato *et al.*, *A concise introduction to decentralized POMDPs*. Springer, 2016, vol. 1.

[29] T. Wang, L. Zeng, W. Dong, Q. Yang, Y. Yu, and C. Zhang, "Context-aware sparse deep coordination graphs," *International Conference on Learning Representations (ICLR)*, pp. 1–23, 2022.

[30] O. Michel, "Cyberbotics ltd. webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[31] M. Samvelyan, T. Rashid, C. Schroeder de Witt, G. Farquhar, N. Nardelli, T. G. Rudner, C.-M. Hung, P. H. Torr, J. Foerster, and S. Whiteson, "The starcraft multi-agent challenge," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 2186–2188.

[32] S. Hu, F. Zhu, X. Chang, and X. Liang, "Updet: Universal multi-agent reinforcement learning via policy decoupling with transformers," *International Conference on Learning Representations (ICLR)*, pp. 1–15, 2021.

[33] T. Wang, T. Gupta, A. Mahajan, B. Peng, S. Whiteson, and C. Zhang, "Rode: Learning roles to decompose multi-agent tasks," *International Conference on Learning Representations (ICLR)*, pp. 1–24, 2021.