Efficient Domain Coverage for Vehicles with Second-Order Dynamics via Multi-Agent Reinforcement Learning

Xinyu Zhao, Razvan C. Fetecau, Mo Chen

Abstract— Collaborative autonomous multi-agent systems covering a specified area have many potential applications, such as UAV search and rescue, forest fire fighting, and real-time high-resolution monitoring. Traditional approaches for such coverage problems involve designing a model-based control policy based on sensor data. However, designing model-based controllers is challenging, and the state-of-the-art classical control policy still exhibits a large degree of sub-optimality. In this paper, we present a reinforcement learning (RL) approach for the multi-agent efficient domain coverage problem involving agents with second-order dynamics. Our approach is based on the Multi-Agent Proximal Policy Optimization Algorithm (MAPPO). Our proposed network architecture includes the incorporation of LSTM and self-attention, which allows the trained policy to adapt to a variable number of agents. Our trained policy significantly outperforms the state-of-the-art classical control policy. We demonstrate our proposed method in a variety of simulated experiments.

I. INTRODUCTION

Multi-agent cooperative area coverage is an active research field, with the objective being to deploy a set of agents over a domain of interest to achieve optimal sensing. Applications include monitoring [1], [2], target detection [3], [4], and search and rescue [5], [6]. In the past decades, researchers have proposed various approaches, including both classical mathematical model-based methods and learning-based methods, to solve coverage problems.

Classical control presents many sophisticated control laws for coverage problems under certain assumptions. Common approaches include solving an optimization problem that involves Voronoi tesselations [7], [8], potential field methods [9], [10], and scalar field mapping [11], [12]. Although those classical approaches have proven effective in many applications, designing a suitable model under complex scenarios is always challenging. Thus, the learning-based method has drawn increasing attention in recent years.

For the learning-based approach, the recent deep reinforcement learning (RL) algorithm could leverage the advantages of neural networks (NNs) and adaptive learning capabilities to obtain a policy that optimizes the performance metric over the trajectory. NNs have the ability to approximate a broad class of functions, and through RL, a control policy parameterized by a NN can be directly learned through real-time interactive feedback from the environment, hence avoiding the challenge of choosing an accurate model.

X. Zhao and R. C. Fetecau are with the Department of Mathematics, Simon Fraser University, Burnaby, BC, Canada (SFU) {xza261@, van@math.}sfu.ca). M. Chen is with the School of Computing Science, SFU, mochen@cs.sfu.ca.
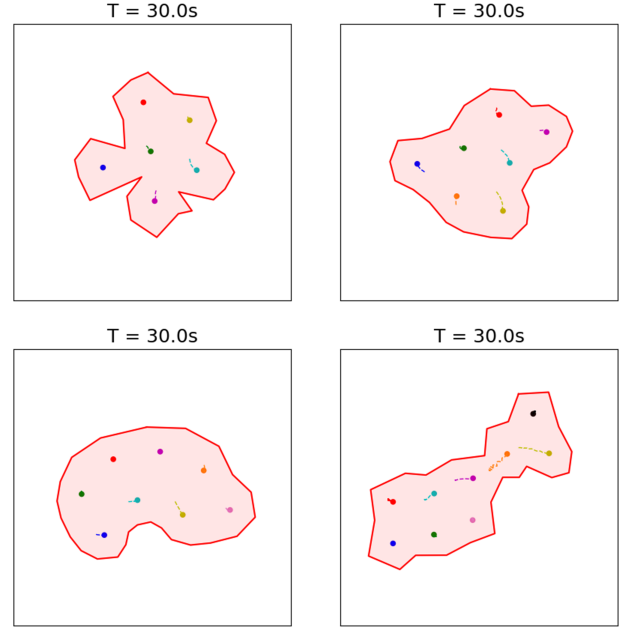
Fig. 1: The final coverage configuration of our policy for 6 to 9 agents in non-convex polygon domains. The simulation duration is the 30s and the tails in all plots represent the agents' trajectories in the last 15s of the simulation. Small perturbations over the last 15s indicate reliable persistent coverage performance.

For most previous works that applied RL methods to the coverage problem [13]–[15], the major limitation is the requirement of discrete action space or even discrete state space. Works that apply the multi-agent RL technique to coverage problems with continuous action space are still limited. One related work [16] applies an actor-critic method involving a specific value function formulation to multi-agent continuous coverage control. In [17], the authors used MADDPG [18] to learn a coverage policy for the ocean monitoring task. Another limitation in previous work is that the NN policy can only handle a fixed number of agents due to the typical multiple-layer perceptron (MLP) networks used in this domain, which require a fixed-dimension input. In addition, besides achieving coverage, another performance metric, such as the time efficiency of achieving coverage, has not been considered.

In this work, we propose a novel RL method to address some of the limitations of the previous work. Our target is to learn a policy for continuous state-action space, which 1) allows the agents to achieve the final coverage configuration in a time-efficient manner and 2) generalizes to scenarios with a variety of domains and a varying number of agents. For

illustration, Fig. 1 shows the final coverage configurations achieved with our policy, with 6 to 9 agents, for non-convex domains. To our best knowledge, this is the first work on adaptive, efficient coverage control using RL for variable numbers of agents. In summary, our contributions are as follows:

- We propose an RL approach for continuous control of area coverage with second-order dynamics through potential-based reward shaping.
- Our approach offers a more effective policy allowing agents to achieve coverage objectives with superiority in time efficiency.
- We propose a new training scheme involving an LSTM-based policy and self-attention-based value function estimation to allow the agent to generalize to scenarios involving varying numbers of agents.

The rest of the paper is organized as follows. Section II states the problem formulation for domain coverage. Section III provides background information. Section IV details our method for solving the time-efficient coverage control across different numbers of agents. Section V illustrates our approach with simulations.

II. PROBLEM FORMULATION

We consider a set of n homogeneous agents indexed by $I = \{1, \ldots, n\}$, with dynamics given by

$$\begin{cases} \dot{p}^i = v^i, & \|v^i\| \leq v_{max}, \\ \dot{v}^i = a^i, & \|a^i\| \leq a_{max}, \end{cases} \tag{1}$$

where $p^i = (p_x^i, p_y^i)$, $v^i = (v_x^i, v_y^i)$ and $a^i = (a_x^i, a_y^i)$ are the position, velocity and the control input of the ith agent, respectively. Also, v_{max} and a_{max} are bounds on velocities and control inputs, and $\|\cdot\|$ denotes the Euclidean distance.

The problem definition of multi-agent area coverage can vary under different scenarios and assumptions. In this paper, we are interested in deploying the set of agents into a target domain to achieve uniform area coverage. Specifically, we adopt the concept of coverage presented in [10], [19].

Definition II.1. We say that the group of agents I is an *l-subcover* configuration for a compact domain $\Omega \subset \mathbb{R}^2$ if

1. $B_{\frac{l}{2}}(i) \subset \Omega$ for all $i \in I$,
2. $\|p^i - p^j\| \geq l$ for all pairs $i, j \in I$, $i \neq j$.

where $B_{\frac{l}{2}}(i)$ denotes the ball of radius $\frac{l}{2}$ centered at p^i.

The main interest in [10] is in l_d-*subcover* configurations, where $l_d = \sqrt{\frac{Area(\Omega)}{n}}$, based on the assumption that each agent covers the same amount of square area. Aside from reaching an l_d-*subcover* configuration, we are also interested in the time efficiency of achieving domain coverage. Based on Def. II.1 and the time efficiency concerns, the primary interest of this work is defined as follows.

Multi-Agent Efficient Domain Coverage Control: We aim to find a control policy that drives a set of n agents from any initial positions to an l_d-*subcover* of some compact domain Ω as quickly as possible.

III. BACKGROUND

In this section, we review some important background information for this work. We consider multi-agent domain coverage control as a fully cooperative task with homogeneous agents, formulated as a decentralized partially observable Markov decision process (Dec-POMDP) [20] of n agents consisting of a tuple of $\langle S, A, O, P, R, n, \gamma \rangle$. Here, S is the joint state space of all agents, A is the joint action space, O is the joint observation space, $P = \mathbb{P}(s_{t+1}|s_t, a)$ is the transition probability function from current joint state s_t to the next state s_{t+1} given the joint action $a = (a^1, \ldots, a^n)$, $R(s_t, a, s_{t+1}) \colon S \times A \times S \longrightarrow \mathbb{R}$ is the global reward function, n is the number of agents, and $\gamma \in [0, 1)$ is a constant discount factor.

The joint state s is only partially observable to each agent, who draws the local observation o^i according to its own observation function $\eta^i \colon S \to O$. At time t, each agent samples an action a_t^i from a stochastic policy $\pi^i(a_t^i|o_t^i)$ given o_t^i. By executing the joint action $a_t = (a_t^1, \ldots, a_t^n)$, the joint state evolves from s_t to s_{t+1} according to the transition probability function P, and an immediate reward $R_t = R(s_t, a_t, s_{t+1})$ is given by the environment. The return G_t is defined as the discounted cumulative future reward starting from state s_t, given by $G_t = \sum_{k=0}^{T} \gamma^k R_{t+k}$. The objective for all agents is to find a joint policy $\pi = (\pi^1, \ldots, \pi^n)$ that maximizes the corresponding state value function $V^\pi(s) = \mathbb{E}_\pi[G_t|s_t = s]$ for any given state s.

A. Multi-Agent Proximal Policy Optimization

Our approach is built upon multi-agent proximal policy optimization (MAPPO) [21], an actor-critic algorithm for the MARL problem. The main feature of MAPPO is to extend the PPO algorithm [22] under the centralized training and decentralized execution (CTDE) framework [18]. In particular, the CTDE extension for any actor-critic architecture allows the critic network to access the environment's global state, while the policy network for every agent still chooses the action based on the agent's local observation only.

Consider an n-agent environment described by Dec-POMDP, with a set of policies $\pi_\theta = (\pi_{\theta^1}, \ldots, \pi_{\theta^n})$ parametrized by $\theta = (\theta^1, \ldots, \theta^n)$, and a centralized value function V_ϕ parametrized by ϕ. The MAPPO algorithm samples trajectories from an old joint policy $\pi_{\theta_{old}}$, and updates the policy parameter for each agent by maximizing the following surrogate objective

$$J(\theta^i) = \mathbb{E}_{s \sim \rho_{\pi_{\theta_{old}}}, a^i \sim \pi_{\theta_{old}^i}} \Big[\min\big(r_t(\theta^i) A_{\pi_{\theta_{old}}}(s_t, a_t), \\ \text{clip}(r_t(\theta^i), 1+\epsilon, 1-\epsilon) A_{\pi_{\theta_{old}}}(s_t, a_t)\big) \Big],$$

where $\rho_{\pi_{\theta_{old}}}$ is the state visitation frequency induced by old policy $\pi_{\theta_{old}}$, $r_t(\theta^i)$ is the probability ratio calculated as

$$r_t(\theta^i) = \frac{\pi_{\theta^i}\big(a_t^i|\eta^i(s_t)\big)}{\pi_{\theta_{old}^i}\big(a_t^i|\eta^i(s_t)\big)},$$

$A_{\pi_{\theta_{old}}}(s_t, a_t)$ is the advantage value at time t, and $\epsilon \in [0, 1)$ is a hyperparameter controlling the clip rate.
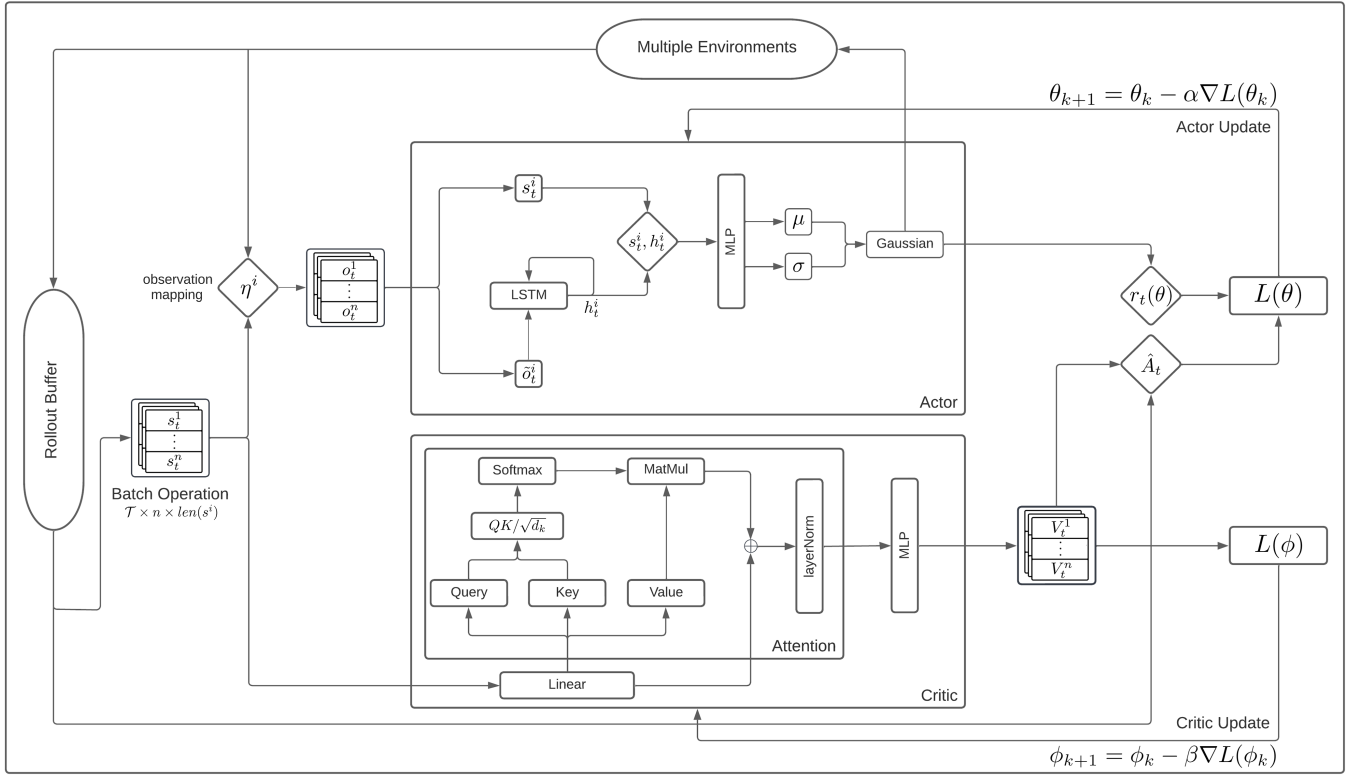
Fig. 2: An overview of the architecture of our approach.

The advantage function $A_{\pi_{old}}(s_t, a_t)$ can be estimated in several ways as in [23] based on the centralized value function V_ϕ, where V_ϕ is found by minimizing the standard mean squared error loss (MSE) between $V_\phi(s_t)$ and a value target $\hat{V}_t$ such as a Monte-Carlo estimation or temporal-difference target. For more details about the PPO and MAPPO objective, we refer the reader to [21], [22].

B. Behaviour Cloning

The behavior cloning (BC) technique is also used to help the training in this work. The main idea of BC is to copy an expert's policy using supervised learning. For a stochastic policy π_θ, BC minimizes the error between the expert's action and the maximum likelihood action over the current policy. This can be done via several different objective functions. Our approach uses a simple version of BC, where the objective is given by

$$L^{BC}(\pi) = \frac{1}{\mathcal{T}} \sum_{t=0}^{\mathcal{T}} \left\| \hat{a}_t^i - \mu_{\pi_\theta}(o_t^i) \right\|^2,$$

where $\hat{a}_t^i$ is the expert's action under the observation o_t^i, and $\mu_{\pi_\theta}(o_t^i)$ is the mean of stochastic policy π_θ.

IV. APPROACH

In this section, we explain the details of our approach for applying MARL to solve the multi-agent efficient domain coverage control problem defined in Sec. II. Two major challenges need to be addressed: The learned policies need to 1) produce a time-efficient trajectory while completing the coverage task, and 2) be able to control different numbers of agents in the environment.

To address the two challenges above, we build our method based on the MAPPO algorithm with newly designed policy and value network architectures. In order to guide the agent to learn a suitable control policy that solves the multi-agent coverage problem efficiently, a proper reward function needs to be designed. We accomplish this by choosing a suitable potential function from the classical coverage control. With an adequately shaped reward, agents learning to maximize the return also improve time efficiency.

For variable numbers of agents, the input to the policy network varies in size and hence causes issues to the MLP network. To enable adaptive control of our policy over variable numbers of agents, we take the idea from natural language processing to reshape the structure of both actor and critic network by leveraging the long short-term memory (LSTM) [24] and the self-attention [25] mechanism to encode the state information into an ideal format as input for MLP networks. Meanwhile, the value decomposition (VD) [26] is applied as a natural complement to the self-attention structures to facilitate the training further. The new architecture enables the training process to happen concurrently across multiple environments with a variable number of agents, and hence the policy could adapt to an environment with an arbitrary number of agents.

An overview of our approach is illustrated in Fig. 2. The "Actor" is the policy network that uses an LSTM to handle the variable length part in the input state. The "Critic"

network is the centralized value function estimation involving self-attention and VD structure. The "Actor" and "Critic" parts of the figure will be explained in detail in Sections IV-C and IV-D, respectively.

A. Action Space, Agent Observation and Environment State

In this work, we aim to directly output the continuous control force following the system (1). Namely, the action space in this work is defined as the agent's acceleration. The assumptions on the environment state and agent observation in this work are as follows. We assume that every agent knows its position p^i and velocity v^i. We further assume that every agent can measure its position relative to the domain boundary $\partial\Omega$, given by the vector

$$d^i := p^i - P_{\partial\Omega}(p^i),$$

where $P_{\partial\Omega}$ denotes the projection operator to $\partial\Omega$. With these assumptions, we define the agent's internal state s^i to be

$$s^i = \left(p^i,\, v^i,\, \hat{d}^i,\, \|d^i\|,\, \mathbb{I}(p^i)\right),$$

where $\hat{d}^i = d^i/\|d^i\|$ and $\mathbb{I}(p^i)$ is given by

$$\mathbb{I}(p^i) = \begin{cases} 1, & \text{if } p^i \notin \Omega, \\ -1, & \text{if } p^i \in \Omega. \end{cases}$$

We further construct the environment full state s by stacking all s^i as

$$s = \begin{bmatrix} s^1 \\ \vdots \\ s^n \end{bmatrix} = \begin{bmatrix} p^1 & v^1 & \hat{d}^1 & \|d^1\| & \mathbb{I}(p^1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p^n & v^n & \hat{d}^n & \|d^n\| & \mathbb{I}(p^n) \end{bmatrix}.$$

The expression of the state as a matrix will be useful in Sec. IV-D.

Partial observability in this paper means that agents do not know all the information about the other agents. However, we assume that each agent knows its position relative to the other agents, represented by $p^{i,j} = p^i - p^j$ for $i \neq j$. Therefore, we consider the agent local observation $o^i = (s^i, \tilde{o}^i)$ to consist of two parts, where the first part is the agent's internal state s^i and the second part is the observation of other agents relative to itself, given by

$$\tilde{o}^i = \left(p^{i,1}, p^{i,2}, \ldots, p^{i,i-1}, p^{i,i+1}, \ldots, p^{i,n}\right).$$

In practice, such an observation can be obtained by ranging sensors such as LIDAR.

B. Reward Shaping

We now discuss reward shaping for the domain coverage problem stated in Sec. II. We design the reward function to address the coverage goal, as well as time efficiency. We base our reward function on the artificial potentials used in [10], [19] for the dynamics of (1). Specifically, the potential functions in [10], [19] for agent-domain and inter-agent interactions, are given respectively by

$$U_h(p^i) = \begin{cases} 0, & \text{for } [\![d^i]\!] \leq -\frac{l_d}{2}, \\ \frac{1}{2}([\![d^i]\!] + \frac{l_d}{2})^2, & \text{for } [\![d^i]\!] > -\frac{l_d}{2}, \end{cases} \quad (2)$$

and

$$U_I(p^{i,j}) = \begin{cases} \frac{1}{2}(\|p^{i,j}\| - l_d)^2, & \text{for } \|p^{i,j}\| < l_d, \\ 0, & \text{for } \|p^{i,j}\| \geq l_d. \end{cases} \quad (3)$$

Here, $[\![d^i]\!]$ denotes the signed distance of p^i from $\partial\Omega$, i.e.,

$$[\![d^i]\!] = \mathbb{I}(p^i) \cdot \|p^i - P_{\partial\Omega}(p^i)\|.$$

Note that the expressions of U_h and U_I correspond directly to the two conditions in Def. II.1 (for $l = l_d$).

The total potential energy of the system is given by

$$\Phi = \sum_{i}^{n} \Phi^i, \quad (4)$$

where Φ^i is the individual potential obtained by summing up U_h and U_I as

$$\Phi^i = 2U_h\left(p^i\right) + \sum_{j \neq i}^{n} U_I\left(p^{i,j}\right).$$

The global minimum of Φ is zero and attained at l_d-*subcover* configurations, and is a measure for coverage performance. In our method, we shape the reward by transforming the potential energy slightly. We define the agent's individual reward $R^i(s_t, a_t, s_{t+1})$ as

$$R^i(s_t, a_t, s_{t+1}) = \begin{cases} -M, & \text{if agent } i \text{ is outside } \Omega, \\ -\Phi^i_{t+1}, & \text{if agent } i \text{ is inside } \Omega. \end{cases}$$

where M is a positive constant, and Φ^i_{t+1} denotes the individual potential for agent indexed by i at time step $t+1$. Here, a constant reward $-M$ is assigned to an agent that is outside the domain, as the potential could be arbitrarily large in such case. We set the constant M to be

$$M = \sup\{\Phi : \text{all agents are inside } \Omega\}.$$

We further define R as the global reward for the Dec-POMDP setup, which is calculated by taking the sum of all agents' individual rewards, i.e.,

$$R(s_t, a_t, s_{t+1}) = \sum_{i=1}^{n} R^i(s_t, a_t, s_{t+1}). \quad (5)$$

Let $R^i_t = R^i(s_t, a_t, s_{t+1})$, and $R_t = R(s_t, a_t, s_{t+1})$. The formulation in (5) allows each agent to receive a different reward, while simultaneously knowing its contribution to the team. We remark that the additive reward formulation allows us to use direct VD, which facilitates the training process in our approach. Details on using the reward (5) will be presented in Sec. IV-D.

The reward in (5) also naturally encourages time efficiency. Indeed, the maximum of R is zero and can be attained at any l_d-*subcover* configuration; hence, a trajectory that attains the coverage goal faster will gain higher returns.

C. Actor Architecture for Variable Length Input

A typical policy network architecture in MARL uses the MLP networks for each agent. However, such architecture

has difficulties to adapt into the scenario of a variable number of agents. In this work, we utilize the parameter sharing (PS) technique [27], [28], where all homogeneous agents share a single policy network $\pi_{\theta^1} = \cdots = \pi_{\theta^n}$, resulting in the policy objective being reduced to follows:

$$J(\theta) = \mathbb{E}_\pi \big[\min(r_t(\theta)\hat{A}_t^i), \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^i \big].$$

Note that the PS technique naturally fits into the purpose of training a single policy to handle a variable number of agents, as the shared policy offers different actions for different agents based on observation from each agent.

Although a shared policy makes the policy suitable for a variable number of agents, the MLP cannot handle the variable length input of observation caused by the variable number of agents. Therefore, for the policy network, we use the LSTM architecture, a popular recurrent NN [24], which accepts a variable length input and produces a fixed-length output.

Besides being able to take as input vectors of varying length, another advantage of the LSTM for multi-agent coverage control is that it takes an *ordered* sequence as input. In our problem definition, the variable length part of the input is $\tilde{o}^i$, an agent's relative observation, which can be naturally treated as a sequence ordered according to the relative importance of other agents. In our application of the LSTM architecture, for every time step t, the first entry of $\tilde{o}_t^i$ will be fed first and produce a hidden state inside the LSTM. When the next entry is fed in, the LSTM will combine both the current entry and the hidden vector from the last entry to produce the next hidden vector. Eventually, the entire input $\tilde{o}^i$ is mapped to the final hidden vector, denoted as h_t^i, which represents an encoded vector that stores all the important information in $\tilde{o}^i$. Later entries have a more direct influence on the LSTM output.

Therefore, we arrange the entries in $\tilde{o}^i$ in descending order of the norm. Such an ordering puts the relative position of the farthest agent first, and of the nearest agent last, and captures the intuition that nearby agents should have a stronger effect on the current agent's action. This ordering is also consistent with the agent interaction potential U_I, where two agents interact more strongly if they are closer to each other.

The LSTM output h_t^i is concatenated with the agents' internal state s^i and fed into the standard MLP to output the policy distribution. The LSTM-based policy network, showing as the "Actor" part of Fig. 2.

D. Self-Attention Based Value Decomposition Network

With the CTDE framework, the shared critic network takes the global state s as input to estimate the true state value, where s is of variable length according to the number of agents. While one can attempt a similar procedure as in Sec. IV-C for the environment state and treat s as a sequence of the agents' internal states s^i, the LSTM structure is not the most appropriate for the *centralized* critic network since there is no natural agent ordering. We propose instead a self-attention based value decomposition(VD) structure, illustrated in the "Critic" part of Fig. 2, which is able to

estimate the state value by equally assessing all agents' internal states s^i.

The self-attention was originally introduced in the Transformer architecture [25], which is used to compute a sequence representation that associates the element at different positions in a sequence. For input a length-n sequence $X \in \mathbb{R}^{n \times d_m}$ of vectors $x_i \in \mathbb{R}^{d_m}$, the self-attention of X outputs

$$\mathrm{attention}(X) = \mathrm{softmax}\left(\frac{(XW_q)(XW_k)^\top}{\sqrt{d_k}}\right)(XW_v), \quad (6)$$

where $W_q \in \mathbb{R}^{d_m \times d_k}, W_k \in \mathbb{R}^{d_m \times d_k}$, and $W_v \in \mathbb{R}^{d_m \times d_v}$ are learnable matrices that project every $x_i \in X$ from $\mathbb{R}^{d_m}$ onto $\mathbb{R}^{d_k}$, $\mathbb{R}^{d_k}$ and $\mathbb{R}^{d_v}$ respectively, and $\sqrt{d_k}$ is a scaling factor depending on the subspace dimension. Each row in the output of $\mathrm{attention}(X) \in \mathbb{R}^{n \times d_v}$ can be viewed as a new vector representation for the i-th element x_i in the sequence X, which contains the correlated information to another element in a different position.

In our work, we use self-attention to encode the global state into a new representation for each agent and connect with a VD structure constructed by shared MLP. This idea is visualized in the "Critic" part of Fig. 2, where we treat the environment full state s as a sequence of length n. For the critic input, the state s_t at every time step is first sent into a linear layer that maps the internal state representation for each agent onto a higher dimensional space. Then, the output from the linear layer, of dimension $\mathbb{R}^{n \times d_m}$, is used to compute the attention output using the matrix equation (6). The final output from attention, denoted as $z_t = \mathrm{attention}(s_t)$, is considered a new representation of the environment. In particular, each row of z_t, denoted as z_t^i, is a new representation of s_t^i that contains not only information of the intrinsic state for the ith agent, but also the information about other agents relevant to coordination in the coverage task. In practice, the attention is often followed by residual connections and layer normalization to facilitate the training, and our implementation preserves this structure as well.

Since the attention output fuses all agents' information, while giving each agent different internal state representations, the shared MLP network can output different values according to the new representation z^i for different agents. Such structure fits well with the VD approach [26], where we explicitly formulate the state value function estimation at time step t as the summation of local state values estimated from every z_t^i, i.e.,

$$V^{\pi_\theta}(s_t) \approx \sum_{i=i}^{n} V_\phi(z_t^i).$$

We remark that the additive VD is legitimate in our setup due to the explicit additive reward formulation and the true value function, defined as $V^{\pi_\theta}(s_t) = \sum_{i=1}^{n} \mathbb{E}_{\pi_\theta}[\sum_{k=0}^{T} \gamma^k R_{t+k}^i | s = s_t]$. The above setup yields the critic loss to be:

$$L^{Critic}(\phi) = \frac{1}{\mathcal{T}} \sum_{t=0}^{\mathcal{T}} \left(\hat{V}^i - V_\phi(z_t^i)\right)^2, \forall\, i \in \{1, \ldots, n\}, \quad (7)$$

where the value target $\hat{V}^i$ is chosen to be the Monte-Carlo estimation of agents' individual returns at every time. Note that the loss (7) is different from the normal VD method, where the shared critic network directly estimates the true agent's individual state value instead of the global state value since the agent's individual rewards can be directly accessed in our work.

The main advantage of using the VD is to alleviate the credit assignment problem compared to using the global value function. VD allows us to decompose the advantage function $A_\pi(s, a)$ as well, which accelerates policy learning further. The structure of the shared value network is still an MLP, where the z_t is fed to compute the final local state value for each agent. Note that the new structure of the critic network still agrees with the main feature of the CTDE framework, where a centralized critic network estimates the state value via global information.

E. Training the Policy

To smooth the policy training, we apply the strategy of two-phase training. The first phase is the pre-training step, and both the policy network and critic network could be fitted into the pre-training phase. For the policy network, the pre-training is applied using the BC method with a set of trajectories sampled from an expert policy f. Meanwhile, the critic network could be pre-trained with sampled trajectories using MSE loss with Monte-Carlo estimation of agents' individual returns.

After the pre-training step, we continue the training with the MAPPO algorithm based on the pre-trained initial model. In order to train a policy that can adapt to variable numbers of agents, the second phase requires sampling multiple environments across different numbers of agents. This is done by paralleled sampling across multiple environments with different numbers of agents in each training iteration. Furthermore, for the possibility of transferring the policy into various domains, each paralleled environment is encouraged to use a different domain to ensure the training covers a variety of shapes.

Since our reward function is partially sparse, most of the exploration in the early stage of training would only provide unimportant samples because sparse rewards do not provide a meaningful signal to agents for most states. Using BC for model pre-training allows our policy to be trained toward expert policy at first, avoiding the problem of reward sparsity in the early training stage. Using the MAPPO algorithm to continue training upon the pre-trained initial model, we can leverage the main strengths of RL to obtain a potentially better policy through proper exploration.

Moreover, the expert policy chosen in this work is the control law proposed in [10], [19], which can be directly derived from the two potentials U_h and U_I (see (2) and (3)) by taking the negative gradient with respect to each agent:

$$a^i = -\nabla_i U_h(p^i) - \sum_{i=1}^{n} \nabla_i U_I(p^{i,j}) - cv^i, \qquad (8)$$

where c is a positive constant. We remark that this controller drives system (1) into an equilibrium configuration which is a critical point of the total system potential energy Φ from (4); note that critical points of Φ are l_d-*subcover* configurations. For more discussion on the asymptotic behavior of system (1) with this control law, we refer to [10], [19].

V. SIMULATED EXPERIMENTS

In this section, we present simulation results of the policy obtained by our approach in various domains and across different numbers of agents. We will compare the performance of our policy and the state-of-the-art classical controller (8) and show the remarkable adaptability of our policy across a different number of agents.

In our experiment, the policy can adapt from 6 to 9 agents across various domains. Per our discussion in Sec. IV-E, the training process is split into two stages. In the pre-training stage, an initial policy is obtained using 10 thousand trajectories sampled from the classical controller (8) over different agent counts. The RL stage continues the training on the initial policy with multiple environments across different numbers of agents with different types of domains. The domain in each environment is either a randomly generated polygon (examples shown in Fig. 1) or an equilateral polygon. We run 16 parallel environments across 6 to 9 agents, each using a different domain shape. In every RL training iteration, we sample 4 trajectories from each environment, and the policy takes about 300 iterations to converge.

All simulations presented in this section are 30 seconds long. For all environments, the agent group's initial position is randomized but follows an approximately horizontal line configuration outside the domain (with only a slight offset from a perfect horizontal line) and with zero initial velocity.

A. Time Efficiency over State-of-the-Art Classical Controller

We compare our policy with the state-of-the-art controller in the 9-agent environment. Take the simple square domain and note that the l_d-*subcover* configuration for the square domain with 9 agents is simple and unique, in which case all agents are arranged in a square formation. Fig. 3 shows the agents' positions and trajectories generated by the classical controller (8) (left column) and by our policy (right column). The colored dots represent the agents' positions at the current time, and the dashed tails represent the trajectories for the past 10 seconds.

In the first 10 seconds of the simulation, our policy behaves very differently from the controller (8). For the trajectories generated by the classical controller, an overshooting behavior can be observed. All agents initially enter the domain, but some pass through the domain after entering it. Such overshooting behavior reduces the time efficiency for achieving the final coverage configuration. In contrast, the trajectories from our policy show that all agents directly approach the final l_d-*subcover* configuration.

For the next 20 seconds, the classical controller brings the overshooting agents back into the domain, then holds all agents within the domain and drives them slowly to the

final configuration. Meanwhile, our policy maintains the l_d-*subcover* configuration with only small perturbations over time.
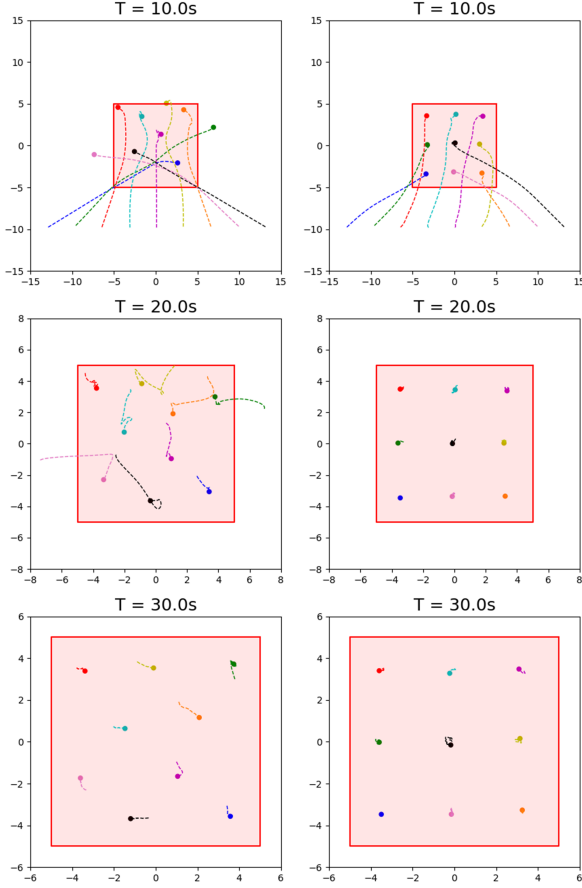

Fig. 3: Square domain coverage of 9 agents for system (1) with classical controller (8) (left) and for our policy (right). Note the superior time efficiency of our policy, by which the agents converge to the correct l_d-*subcover* configuration in only about 10 seconds.

We also measure the coverage performance using the system potential Φ. Fig. 4 shows the evolution of Φ over time, for the square domain simulation, where the red dashed curve corresponds to the classical controller (8) and the solid blue curve is for our policy. The left plot is for the whole simulation duration, while the right plot gives a zoomed-in view of the last 20 seconds. During approximately the first 5 seconds, the system potential Φ decreases rapidly in both the classical controller (8) and our policy.

The two bumps of the potential energy for the classical controller (at around $t \approx 6$ and $t \approx 10$) can be explained as follows. The first bump occurs when some agents get too close to each other inside the domain, resulting in strong repulsion forces. Such repulsive forces push the agents to get far away from each other so strongly that some agents are pushed out of the domain. This overshoot causes the second bump in Φ – see also Fig. 3 (top left).

In contrast, the solid blue curve shows that for our policy the potential decreases quickly and stays at a low level without any increase. Note that the potential level of our

policy is always below that of the classical controller (8), which does not converge to the l_d-*subcover* configuration within 30s duration.
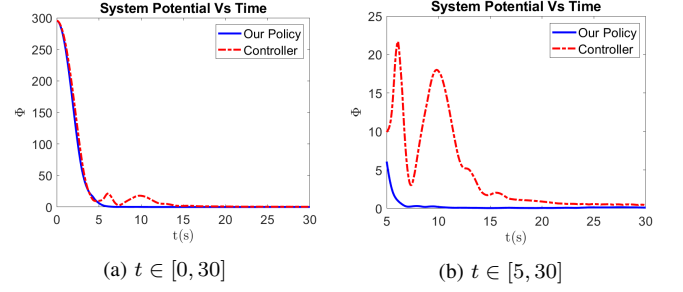

Fig. 4: Time evolution of the total potential energy Φ from (4) for the classical controller (dashed red) and our policy (solid blue). The left plot shows the entire duration and the right plot zooms in on the last 25 seconds. The two bumps of the red curve in the left plot are due, respectively, to strong repulsive forces and the overshooting behavior present in the classical controller. The right plot shows that our policy maintains a low potential in the last 20 seconds. Note that the corresponding potential level of our policy is always below that of the classical controller.

For non-convex domains, our policy preserves the same time-efficient advantages over controller (8). Fig. 5 shows the positions and trajectories of 9 agents that cover a non-convex non-symmetric polygon corresponding to the classical controller (left) and to our policy (right). Both controller and our policy achieve the final coverage configuration within 20 seconds, while our policy only takes about 10 seconds. We note again the overshooting phenomenon present in the controller in the non-convex polygon domain as well.
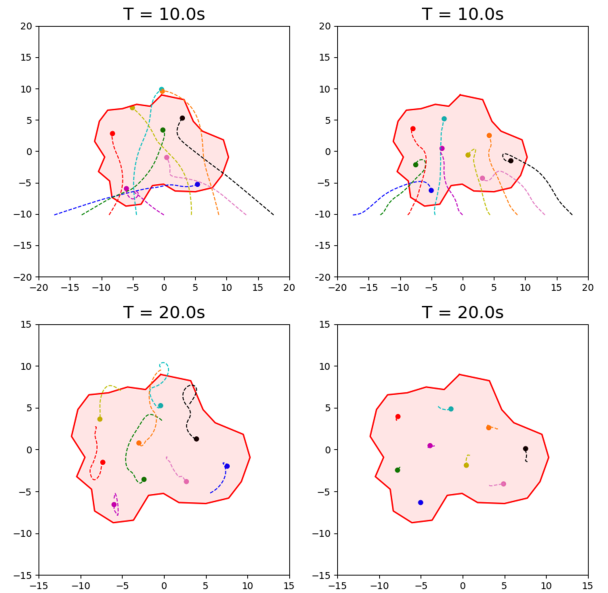

Fig. 5: Non-convex domain coverage of 9 agents for system (1) with classical controller (8) (left) and for our policy (right). Our policy has a superior time efficiency for convergence to an l_d-*subcover* configuration.

B. Generalization to Different Agent Counts

From 6 to 9 agents, we run 1000 simulations for each agent count with both controller (8) and our policy in different domains and measure the system potential Φ at every time step for all sampled trajectories. A trajectory is considered to deliver a successful coverage result if the system potential level is below 0.15 (low enough to be an approximate l_d-subcover configuration) before 30 seconds and lasts for the rest of the simulation time. Table I shows the success rate of different agent counts over all simulations and the average convergence time over all samples that provided a successful coverage result. Our policy performs better than the controller across different agent counts, with significant savings in the convergence times. Animated simulations are available on YouTube at https://youtu.be/OgF62-NSbc0.

Agents	Success Rate		Convergence Time	
	Controller	Policy	Controller	Policy
6	83.4%	88.6%	25.87s	13.35s
7	88.1%	88.8%	26.14s	15.28s
8	90.8%	94.1%	28.62s	16.76s
9	80.6%	83.6%	29.72s	23.99s

TABLE I: Success rate and average convergence time for controller (8) and our policy. The success rate only considers the simulations that achieve a successful coverage result within 30 seconds. A "failed" simulation can possibly achieve the desired coverage result with a longer simulation time. Among all successful samples, our policy offers at least 30% improvements in the convergence speed, for all agent counts.

VI. CONCLUSION

In this work, we present a Self-Attention based reinforcement learning approach for the multi-agent efficient coverage control involving agents in continuous state and action with second-order dynamics. By using the LSTM and self-attention structure, our policy can perform adaptive control tasks with a variable number of agents. Our novel approach is also shown to outperform the existing classical controller in terms of time efficiency while maintaining the same level of coverage performance compared to the classical controller. Future work will investigate the potential of using reinforcement learning to solve coverage control incorporated with collision avoidance.

REFERENCES

[1] G. Zhang, G. K. Fricke, and D. P. Garg, "Spill detection and perimeter surveillance via distributed swarming agents," *IEEE/ASME Trans. Mechatronics*, vol. 18, no. 1, pp. 121–129, 2013.

[2] W. J. Yun, S. Park, J. Kim, M. Shin, S. Jung, D. A. Mohaisen, and J.-H. Kim, "Cooperative multiagent deep reinforcement learning for reliable surveillance via autonomous multi-uav control," *IEEE Trans. Industr. Inform.*, vol. 18, no. 10, pp. 7086–7096, 2022.

[3] X. Dang, C. Shao, and Z. Hao, "Target detection coverage algorithm based on 3d-voronoi partition for three-dimensional wireless sensor networks," *Mobile Information Systems*, vol. 2019, pp. 1–15, 03 2019.

[4] W. Wang, V. Srinivasan, K.-C. Chua, and B. Wang, "Energy-efficient coverage for target detection in wireless sensor networks," in *Int. Symposium on Information Processing in Sensor Networks*, 2007.

[5] D. Drew, "Multi-agent systems for search and rescue applications," *Current Robotics Reports*, vol. 2, June 2021.

[6] J. P. Queralta, J. Taipalmaa, B. Can Pullinen, V. K. Sarker, T. Nguyen Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund, "Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision," *IEEE Access*, vol. 8, pp. 191617–191643, 2020.

[7] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Trans. Robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[8] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Trans. Industr. Inform.*, vol. 9, no. 1, pp. 427–438, 2013.

[9] S. S. Ge and Y. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonomous Robots*, vol. 13, pp. 207–222, 2002.

[10] J. Chacon, M. Chen, and R. C. Fetecau, "Safe coverage of compact domains for second order dynamical systems," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15167–15173, 2020.

[11] Y. Tan, "Multi-robot swarm for cooperative scalar field mapping," in *Handbook of Research on Design, Control, and Modeling of Swarm Robotics*, pp. 383–395, IGI Global, 2015.

[12] M. T. Nguyen, H. M. La, and K. A. Teague, "Collaborative and compressed mobile sensing for data collection in distributed robotic networks," *IEEE Trans. Control Netw. Syst.*, vol. 5, no. 4, pp. 1729–1740, 2018.

[13] H. X. Pham, H. M. La, D. Feil-Seifer, and A. Nefian, "Cooperative and distributed reinforcement learning of drones for field coverage," 2018. arXiv preprint:1803.07250.

[14] J. Xiao, G. Wang, Y. Zhang, and L. Cheng, "A distributed multi-agent dynamic area coverage algorithm based on reinforcement learning," *IEEE Access*, vol. 8, pp. 33511–33521, 2020.

[15] J. Heydari, O. Saha, and V. Ganapathy, "Reinforcement learning-based coverage path planning with implicit cellular decomposition," 2021. arXiv preprint: 2110.09018.

[16] A. A. Adepegba, S. Miah, and D. Spinello, "Multi-agent area coverage control using reinforcement learning," in *Int. Florida Artificial Intelligence Research Society Conf.*, 2016.

[17] M. Kouzehgar, M. Meghjani, and R. Bouffanais, "Multi-agent reinforcement learning for dynamic ocean monitoring by a swarm of buoys," in *Global Oceans 2020: Singapore – U.S. Gulf Coast*, 2020.

[18] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Int. Conf. Neural Information Processing Systems*, December 2017.

[19] J. Chacon, M. Chen, and R. C. Fetecau, "Safe coverage of moving domains for vehicles with second order dynamics," *IEEE Trans. Autom. Control*, 2022. Early access.

[20] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*. Springer, 1st ed., 2016.

[21] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of PPO in cooperative, multi-agent games," 2022. arXiv preprint: 2103.01955.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. arXiv preprint: 1707.06347.

[23] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Int. Conf. Learning Representations (ICLR)*, 2016.

[24] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Int. Conf. Neural Information Processing Systems*, 2017.

[26] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Int. Conf. Autonomous Agents and MultiAgent Systems*, 2018.

[27] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems*, 2017.

[28] J. K. Terry, N. Grammel, S. Son, and B. Black, "Parameter sharing for heterogeneous agents in multi-agent reinforcement learning," 2020. arXiv preprint: 2005.13625.