IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

LPAC: Learnable Perception-Action-Communication Loops with Applications to Coverage Control

Saurav Agarwal, Ramya Muthukrishnan, Walker Gosrich, Vijay Kumar, and Alejandro Ribeiro

***Abstract*—Coverage control is the problem of navigating a robot swarm to collaboratively monitor features or a phenomenon of interest not known *a priori*. The problem is challenging in decentralized settings with robots that have limited communication and sensing capabilities. We propose a learnable Perception-Action-Communication (LPAC) architecture for the problem, wherein a convolutional neural network (CNN) processes localized perception; a graph neural network (GNN) facilitates robot communications; finally, a shallow multi-layer perceptron (MLP) computes robot actions. The GNN enables collaboration in the robot swarm by computing *what* information to communicate with nearby robots and *how* to incorporate received information. Evaluations show that the LPAC models—trained using imitation learning—outperform standard decentralized and centralized coverage control algorithms. The learned policy generalizes to environments different from the training dataset, transfers to larger environments with more robots, and is robust to noisy position estimates. The results indicate the suitability of LPAC architectures for decentralized navigation in robot swarms to achieve collaborative behavior.**

***Index Terms*—Graph Neural Networks, Coverage Control, Distributed Robot Systems, Swarm Robotics, Deep Learning Methods**

I. Introduction

NAVIGATING a swarm of robots through an environment to achieve a common collaborative goal is a challenging problem, especially when the sensing and communication capabilities of the robots are limited. These problems require systems with high-fidelity algorithms comprising three key capabilities: perception, action, and communication, which are executed in a feedback loop, i.e., the Perception-Action-Communication (PAC) loop. To seamlessly scale the deployment of such systems across vast environments with large robot swarms, it is imperative to consider a decentralized system wherein each robot autonomously makes decisions, drawing upon its own observations and information received from neighboring robots.

However, designing a navigation algorithm for a decentralized system is challenging. The robots perform perception and action independently, while the communication module is the only component that can facilitate robot collaboration. Under limited communication capabilities, the robots must decide *what* information to communicate to their neighbors and *how* to use the received information to take appropriate actions. This article studies the coverage control problem [1], [2] as a canonical problem for the decentralized navigation of robot swarms. We propose a learnable PAC (LPAC) architecture that can learn to process sensor observations, communicate relevant information, and take appropriate actions.

Coverage control is the problem of collaboration in a robot swarm to provide optimal sensor coverage to a set of features or a phenomenon of interest in an environment [2]. The relative importance of the features is modeled as an underlying scalar field known as the *importance density field* (IDF) [3]. Coverage control with unknown features of interest has applications in various domains, including search and rescue, surveillance [4], and target tracking [5]. Consider a critical search and rescue scenario where a robot swarm assists and monitors survivors in a disaster area. A simple solution could be to distribute the robots in a manner that keeps them further away from each other and covers the entire environment. However, if one or more survivors are detected, it is desirable to have a robot close to each survivor to provide vigilance, which is not captured by this simple solution. The fundamental question, then, is how the robots should collaborate to cover the environment efficiently while accounting for the features of interest, i.e., the survivors in this example. Similarly, coverage control can be used to monitor a continuous phenomenon of interest, such as a wildfire or soil moisture.

This article considers a decentralized multi-robot system with limited communication and sensing capabilities. Furthermore, the environment is not known *a priori*, and the robots use their sensors to make localized observations of the IDF in the environment. Decentralized algorithms based on centroidal Voronoi tessellation (CVT), e.g., Lloyd's method, have been developed for robots with limited communication capabilities [2], [6]. However, in the absence of any prior knowledge of the IDF, such algorithms exhibit poor performance compared to their centralized counterpart. The primary reason for this is that the robots communicate only the relative positions to their neighbors, which does not directly capture the observations of the robots. In contrast, communicating the entire observation of a robot to its neighbors scales poorly with team size, observation size, and time. The LPAC architecture proposed in this article addresses this issue by learning an abstract representation of the observations and communicating the relevant information to nearby robots.

The primary *contribution* of the article is a learnable

This work was supported in part by grants ARL DCIST CRA W911NF-17-2-0181, NSF 2415249, and ONR N00014-20-1-2822.

The source code for the architecture implementation and simulation platform is available at https://github.com/KumarRobotics/CoverageControl.

S. Agarwal, W. Gosrich, and V. Kumar are with the GRASP Laboratory, University of Pennsylvania, USA.
E-mail: {sauravag,gosrich,kumar}@seas.upenn.edu

R. Muthukrishnan is with the CSAIL, Massachusetts Institute of Technology, USA. Her contributions were made while she was affiliated with the University of Pennsylvania. E-mail: ramyamut@mit.edu

A. Ribeiro is with the Department of Electrical and Systems Engineering, University of Pennsylvania, USA. E-mail: aribeiro@seas.upenn.edu

©2026 IEEE

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

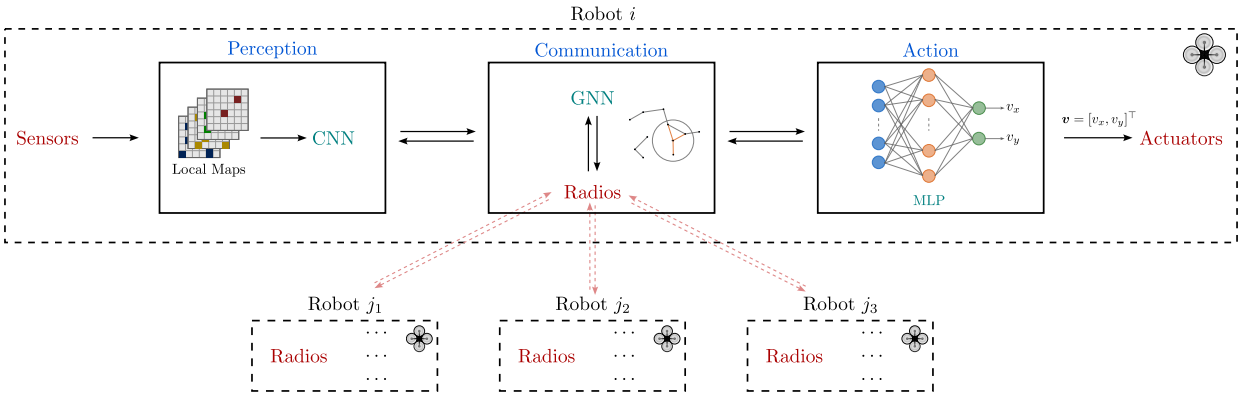


Fig. 1. The proposed learnable Perception-Action-Communication (LPAC) architecture for decentralized navigation of robot swarms: (1) In the perception module, a convolutional neural network (CNN) processes maps representing localized observations and generates an abstract representation. (2) In the communication module, a graph neural network (GNN) performs computations on the output of the perception module and the messages received from neighboring robots. It generates a fixed-size message to share with nearby robots and aggregates the received information to generate a feature vector for the action module of the robot. (3) In the action module, a shallow multilayer perceptron (MLP) computes the control actions for the robot based on the output generated by the GNN. The three modules are executed on each robot independently, with the GNN in the communication module facilitating collaboration between robots.

Perception-Action-Communication (LPAC) architecture for the decentralized coverage control problem (Fig. 1). The architecture is composed of three different types of neural networks, one for each module of the PAC system. (1) In the perception module, a convolutional neural network (CNN) processes localized IDF observations and generates an abstract representation. (2) In the communication module, a GNN performs computation on the output of the perception module and the messages received from neighboring robots. It generates a fixed-size message to communicate with the neighbors and aggregates the received information to generate a feature vector for the action module of the robot. (3) In the action module, a shallow multilayer perceptron (MLP) predicts the control actions of the robot based on the feature vector generated by the GNN.

The LPAC architecture is trained using imitation learning with a clairvoyant centralized planner, which has access to the entire IDF and the positions of the robots in the environment. We extensively evaluate the performance of the architecture and establish the following: (1) The LPAC architecture can learn to communicate the relevant information to achieve performance significantly better than both decentralized and centralized CVT-based algorithms. (2) The learned policy generalizes to a wide range of features and sizes of the robot swarm. (3) The models transfer well to larger environments with bigger robot swarms without any degradation in performance. (4) The policy is robust to noisy position estimates. (5) The model performs well on a real-world traffic light dataset without further training or fine-tuning. These results indicate that the LPAC architecture, with a GNN as a collaboration unit, is a promising learning-based approach for the decentralized navigation of robot swarms.

The rest of the article is organized as follows. Section II discusses the related work focusing on the graph neural networks for robot navigation and the coverage control problem. The problem statement, in Section III, formalizes the decentralized navigation of robot swarms and the coverage control problem. The LPAC architecture is presented in Section IV. The environment setup, dataset generation, and imitation learning are discussed in Section V. Extensive simulation results are presented in Section VI. Finally, Section VII provides concluding remarks and discusses future work.

II. Related Work

Algorithms for the navigation of robots, a fundamental problem in multi-robot systems, often need to execute a PAC loop to achieve a collective goal. The perception module is responsible for acquiring localized sensor observations and processing them to generate a representation of the environment. Deep learning, in particular convolutional neural networks (CNNs), has been successfully used for processing sensor observations for tasks such as mapping [7], place recognition [8], and visual odometry [9]. However, designing a navigation algorithm for a decentralized system is challenging as the robots perform perception and action independently, while the communication module is the only component that can facilitate robot collaboration. Graph neural networks (GNNs) have been shown to be effective in learning decentralized controllers for multi-robot systems. We focus the related work on *(i)* GNNs for navigation of multi-robot systems and *(ii)* the coverage control problem as a canonical example of a decentralized navigation problem.

A. Graph Neural Networks for Navigation Problems

The main challenge in decentralized navigation problems with robots that have limited sensing and communication capabilities is to design a function that computes the information to be communicated to neighboring robots and a policy that can incorporate the received information with the local observations to make decisions. Graph neural networks (GNNs) [10] are particularly suitable for this task as they can operate on a communication graph—the graph imposed by the communication topology of the robots—and can learn to aggregate information from neighboring robots to make decisions [11], [12]. Furthermore, GNNs exhibit several desirable properties

for decentralized systems [13]: *(i) transferability* to new graph topologies not seen in the training set, *(ii) scalability* to large teams of robots, and *(iii) stability* to graph deformations.

Tolstaya et al. [11] established the use of GNNs for learning decentralized controllers for robot swarms and demonstrated their effectiveness in the flocking problem. They use aggregation GNNs, along with aggregated messages, as the primary architecture for learning a controller that can exploit information from distant robots using only local communications with neighboring robots. Gama et al. [14] proposed a framework for synthesizing GNN-based decentralized controllers using imitation learning. They illustrate the potential of the framework by learning decentralized controllers for the flocking and path planning problems. Building on these results, we use GNNs in the LPAC architecture as the primary module for collaboration in the coverage control problem.

GNNs have been demonstrated to be effective in several multi-robot tasks including flocking [11], [15], [14], [16], path planning [17], [18], target tracking [19], perimeter defense [20], and information acquisition [21]. Gosrich et al. [3] recently demonstrated the use of GNNs for the coverage control problem with robots that have limited sensing but assumed full communication capabilities. A common theme in these works is the use of imitation learning to train the GNNs. Most works use carefully hand-crafted features as the input to the GNNs [3], [11], [14], [12], [18], which, in general, can be challenging to design for complex tasks.

Similar to our work, Li et al. [17] and Hu et al. [16] use CNNs in conjunction with GNNs for path planning and flocking problems, respectively. In [17], CNNs are used to process a local map maintained individually by each robot using the local observations, while in [16], the processing is done on raw images in simulated environments. Using raw images is not always desirable, as a CNN trained in a simulated environment may not generalize to real-world environments. Furthermore, raw images capture the current state of the environment but not the history of the environment. Since it is common to maintain a local map in navigation problems, using a CNN to process local maps [17] is more generalizable and can be applied to different hardware platforms and sensors.

Recently, ROS-based frameworks have been developed for testing and deploying decentralized GNN-based policies [22], [23]. While Blumenkamp et al. [22] focus on GNNs and different types of network setups, Agarwal et al. [23] generalize to asynchronous and decentralized implementation of LPAC architectures. Despite several important applications of GNNs in multi-robot systems, detailed study of the transferability, generalizability, and robustness properties of GNN-based architectures has been limited, especially when combined with a CNN. In this article, we study these properties of the LPAC architecture in the context of the coverage control problem.

B. Multi-Agent Reinforcement Learning (MARL)

A joint space representation does not scale well with a large number of robots and makes it challenging to train and deploy deep neural network policies. Foerster et al. [24] provided an RL framework to explicitly learn communication protocols. The approach uses Q-learning with the help of lookup tables, and is demonstrated on switch riddle and MNIST games. Gómez et al. [25] provided an MARL framework for multi-robot systems where they learn to decide on which agents to communicate with. The framework is integrated with non-linear model predictive control to solve collision avoidance. Although the robots have independent trajectories, the system assumes that the robots have complete observation of the positions and velocities of all other robots. MARL has also been used for coverage control [26] using multi-agent deep deterministic policy gradient. However, they assume positions of all robots are known to each agent, do not perform explicit communication, and are limited to 10×10 environments. In these MARL works, there has been a lack of extensive study on the scalability of the policies to a large number of agents.

The theoretical properties of GNNs on transferability, scalability and stability render them more suitable for decentralized systems than the MARL framework. Furthermore, imitation learning often helps in converging to a good policy faster than exploring a large action space in the reinforcement learning framework. An interesting approach would be to use imitation learning for an initial policy and then fine-tune using RL frameworks to improve performance.

C. Coverage Control in Multi-Robot Systems

The coverage control problem is a well-studied problem in multi-robot systems and is often used as a benchmark problem for evaluating decentralized controllers. The problem is closely related to the sensor coverage problem [27] and the facility location problem [28]. Cortés et al. [2] were the first to propose decentralized control algorithms for the problem with robots that have limited sensing and communication capabilities. The algorithms iteratively use the centroidal Voronoi tessellation (CVT) [6], [29] of the environment to assign each robot to a region of dominance and perform gradient descent on a cost function to converge to a local minimum. The algorithms are based on the Lloyd algorithm [30] from quantization theory.

Several standard approaches build upon the work of Cortés et al. [1], [2] and use CVT-based algorithms for the coverage problem. Different sensor models have been used in the literature, e.g., limited sensor radius identical among all agents [2], [31], heterogeneous sensing capabilities [32], and heterogeneous disk sector shaped sensor models [33], [34]. This article considers robots with a limited sensor radius identical to all robots. Sensor models with a footprint in the shape of a disk sector need to include the orientation of the sensor in the state space. Although we do not consider such a model, the LPAC architecture can be extended to include the sensor orientation.

Similar to most of the work in the literature, we assume that the robots have sensors or processing capabilities to obtain the IDF in the sensor field of view. However, IDF estimation techniques based on consensus learning [35] and using Gaussian processes [36], [37] have been studied. Including such techniques in the LPAC architecture is an interesting direction for extending the work.

Incorporating explicit exploration strategies can improve the performance of coverage control algorithms at the cost of addi-

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

tional time spent strategically exploring the environment [38], [39]. Exploration and coverage are opposing objectives, and the trade-off between the two objectives is a challenging problem. A dual LPAC architecture that can switch between exploration and coverage modes may be a promising direction for this problem. Other works have considered time-varying density fields [40], [41] for which incorporating an attention-based architecture [42] in the LPAC architecture may be beneficial. In this article, we restrict to the standard coverage control problem with no explicit exploration and a static density field.

Coverage control has been used in real-world applications such as monitoring of algae blooms [43], agricultural fields [44], indoor environments [45], surveillance [4], and target tracking [5]. In this work, we evaluate the LPAC architecture in coverage control of semantic-derived real-world data; we use traffic signal data from 50 cities in the United States to model the importance density field of the environment. The wide range of applications of coverage control has been the primary motivation for the development of decentralized controllers for the problem. Hence, we use the coverage control problem to evaluate the LPAC architecture.

III. Problem Statement

The multi-robot coverage control problem belongs to the class of navigation control problems. A general navigation control problem is defined on a d-dimensional environment $\mathcal{W} \subset \mathbb{R}^d$. We are given a homogeneous team of N robots $\mathcal{V} = \{1, \dots, N\}$ with $\mathbf{x}_i(t) \in \mathbb{R}^{d_x}$ representing the state of robot i at time t. Along with the position of the robot, the state can also include other information such as velocity, acceleration, and sensor observations. Similarly, the control input for robot i at time t is denoted by $\mathbf{u}_i(t) \in \mathbb{R}^{d_u}$ in a d_u-dimensional control space. The collective state and control for the entire system can be represented as:

$$\mathbf{X}(t) = \begin{bmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \\ \vdots \\ \mathbf{x}_N(t) \end{bmatrix} \in \mathbb{R}^{N \times d_x}, \quad \mathbf{U}(t) = \begin{bmatrix} \mathbf{u}_1(t) \\ \mathbf{u}_2(t) \\ \vdots \\ \mathbf{u}_N(t) \end{bmatrix} \in \mathbb{R}^{N \times d_u}.$$

Assuming a static environment, given a state $\mathbf{X}(t)$ and a control $\mathbf{U}(t)$, the state of the system evolves according to the following Markov model $\mathbb{P}$ for time step Δt:

$$\mathbf{X}(t + \Delta t) = \mathbb{P}(\mathbf{X} \mid \mathbf{X}(t), \mathbf{U}(t)).$$

In a centralized setting, the navigation control problem is posed as an optimization problem with a cost function $\mathcal{J}(\mathbf{X}(t), \mathbf{U}(t))$. Its goal is to find a centralized control policy Π_c^* that minimizes the expected cost [14]:

$$\Pi_c^* = \arg\min_{\Pi_c} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \mathcal{J}(\mathbf{X}(t), \mathbf{U}(t))\right].$$

Here, the control actions are drawn from the policy as $\mathbf{U}(t) = \Pi_c(\mathbf{U} \mid \mathbf{X}(t))$. The discount factor is given by $\gamma \in [0, 1)$, and T is the total time for which the policy is executed. A centralized policy requires complete knowledge of the state of the system $\mathbf{X}(t)$, and all robots need to communicate with the central controller, resulting in a bottleneck in the system. The computation of the control actions is also centralized, which can be computationally expensive and does not scale well with a large number of robots. Furthermore, a central controller is not robust because of a single point of failure. This motivates us to study the decentralized navigation control problem.

A. Decentralized Navigation Control

In a decentralized setting, each robot computes its own control action based on its own state and the states of nearby robots. The robots can only communicate with other robots that are within a limited communication radius r_c. The problem can now be formulated on a *communication graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices representing the robots and $\mathcal{E}$ is the set of edges representing the communication topology. An edge $(i, j) \in \mathcal{E}$ exists if and only if robots i and j can communicate with each other, i.e., when $\|\mathbf{p}_i(t) - \mathbf{p}_j(t)\| \leq r_c$. The communication graph is assumed to be undirected, i.e., $(i, j) \in \mathcal{E} \iff (j, i) \in \mathcal{E}$. The neighbors of robot i are the robots that it can communicate with and are denoted by $\mathcal{N}(i) = \{j \mid (i, j) \in \mathcal{E}\}$.

Let $\mathcal{X}_i(t)$ be the total information acquired by robot i through its own observations and through communication with its neighbors. In general, a robot may communicate an abstract representation of its state to its neighbors, i.e., $\mathcal{X}_i(t)$ need not be a simple aggregation of the states. A decentralized policy leverages only this locally available information, $\mathcal{X}_i(t)$. Unlike the centralized policy, the decentralized policy is executed by each robot independently and does not require communication with a central controller, thereby mitigating the issues of robustness and scalability.

B. Coverage Control Problem

The coverage control problem is a navigation control problem where the goal is to provide sensor coverage based on the importance of information at each point in the environment. An importance density field (IDF) $\Phi : \mathcal{W} \to \mathbb{R}_{\geq 0}$ is defined over a 2-D environment $\mathcal{W} \subset \mathbb{R}^2$. The IDF represents a non-negative measure of importance at each point in the environment. With the state of a robot i given by its position $\mathbf{p}_i \in \mathcal{W}$ in the environment, the control actions given by the velocity $\dot{\mathbf{p}}_i(t)$, and Δt as the time step, we use the following model for the state evolution:

$$\mathbf{p}_i(t + \Delta t) = \mathbf{p}_i(t) + \dot{\mathbf{p}}_i(t)\Delta t. \tag{1}$$

The cost function for the coverage control problem is defined as:

$$\mathcal{J}(\mathbf{X}) = \int_{\mathbf{q} \in \mathcal{W}} \min_{i \in \mathcal{V}} f(\|\mathbf{p}_i - \mathbf{q}\|) \Phi(\mathbf{q})\, d\mathbf{q}. \tag{2}$$

Here, f is a non-decreasing function, and a common choice is $f(x) = x^2$ as it is a smooth function and is easy to optimize. We drop the time index t here and in the rest of the article for convenience.

Assuming that no two robots can occupy the same point in the environment, the Voronoi partition [46] can be used

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

to assign each robot a distinct portion of the environment to cover. The Voronoi partition $\mathcal{P}$ is defined as:

$$\begin{aligned} &\mathcal{P} = \{P_i \mid i \in \mathcal{V}\}, \quad \text{where} \\ &P_i = \{\mathbf{q} \in \mathcal{W} \mid \|\mathbf{p}_i - \mathbf{q}\| \leq \|\mathbf{p}_j - \mathbf{q}\|, \forall j \in \mathcal{V}\}. \end{aligned} \tag{3}$$

All points in P_i are closer to robot i than any other robot. The cost function (2) can now be expressed in terms of the Voronoi partition as:

$$\mathcal{J}(\mathbf{X}) = \sum_{i=1}^{N} \int_{\mathbf{q} \in P_i} f(\|\mathbf{p}_i - \mathbf{q}\|) \Phi(\mathbf{q}) \, d\mathbf{q}. \tag{4}$$

The cost function (4) is a sum of integrals over disjoint regions and is much easier to compute and optimize than the original function (2). Furthermore, if the Voronoi partition is known, the cost function can be computed in a decentralized manner, as each robot only needs to compute the integral over its own region P_i.

We can now define the coverage control problem in the context of the navigation control problem: Find a decentralized control policy Π that minimizes the expected cost $\mathcal{J}(\mathbf{X})$ (4). The policy Π is defined over a space of all possible velocities, and each robot independently executes the same policy.

Fig. 2 shows a near-optimal solution, along with Voronoi partition, to an instance of the coverage control problem with 32 robots and 32 features in a $1024\,\mathrm{m} \times 1024\,\mathrm{m}$ environment.

In this article, we consider the decentralized coverage control problem with the following restrictions: (R1) At any time, each robot can make localized observations of the IDF within a sensor field of view (FoV). (R2) The IDF Φ is static and is not known *a priori*. (R3) Each robot can maintain only its own localized observations of the IDF aggregated over time. (R4) The robots can only communicate with other robots that are within a limited communication radius.

Additionally, our simulation environment assumes, without loss of generality: (R5) The boundary of the environment is convex to allow the use of centroidal Voronoi tessellation (CVT) for imitation learning and evaluation. This can be relaxed by using coverage control algorithms for non-convex environments [47]. (R6) The IDFs are generated using standard random 2D Gaussian functions truncated at 2σ (see Section V-B). This is representative of applications where the importance of information is localized around certain points in the environment and is reduced with distance from these points. However, the overall approach is general and can potentially be applied to other types of IDFs.

In such a setting, a coverage control algorithm needs to provide the following based on the state of robot i and the information received from its neighbors $\mathcal{N}(i)$:

1) A function $\mathcal{I}$ that computes the information, in the form of messages, to be communicated by robot i to its neighbors, and
2) A common policy Π that computes the control action $\mathbf{u}_i = \dot{\mathbf{p}}_i$ for any robot $i \in \mathcal{V}$.

Designing such decentralized algorithms is challenging and can be intractable for complex systems [48]. This motivates us to use a learning-based approach to design a decentralized coverage control algorithm. As we will see in Section IV, the LPAC architecture with GNN addresses the above challenges and provides a scalable and robust solution to the problem.

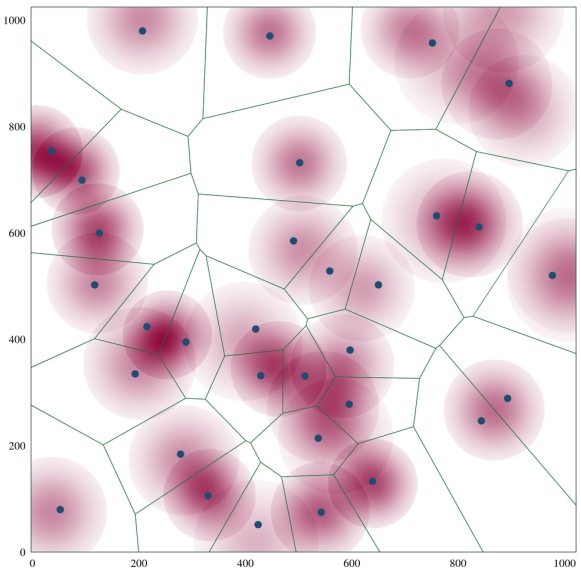


Fig. 2. A near-optimal solution to the coverage control problem: A team of 32 robots is deployed in an environment of size $1024\,\mathrm{m} \times 1024\,\mathrm{m}$. There are 32 features represented as Gaussians to represent the importance density field (IDF). Robots position themselves to provide sensor coverage to the features of interest. The green lines represent the Voronoi partition of the environment with respect to the robot positions. A robot is closer to all points in its Voronoi region than any other robot.

Remark. Let $A_i \subset \mathcal{W}$ denote the current sensor FoV. Then, the cost function can be formulated such that only the current observation is considered, following [1], [2], [3], [31]:

$$\mathcal{J}(\mathbf{X}) = \sum_{i=1}^{N} \int_{\mathbf{q} \in (P_i \cap A_i)} f(\|\mathbf{p}_i - \mathbf{q}\|) \Phi(\mathbf{q}) \, d\mathbf{q}. \tag{5}$$

Alternatively, under the restrictions of a static IDF (R2) and each robot maintaining its own observations (R3), the cost function can be stated in terms of the observed workspace, $\mathcal{W}_o \subseteq \mathcal{W}$:

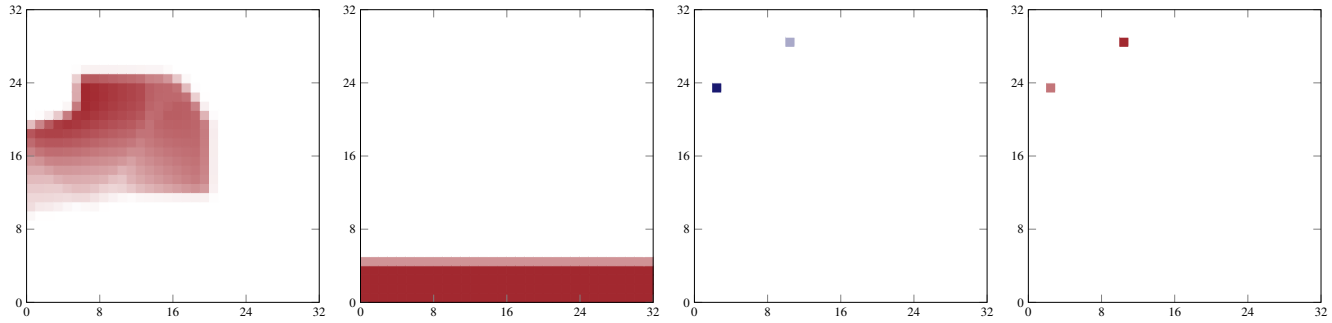
$$\mathcal{J}(\mathbf{X}) = \sum_{i=1}^{N} \int_{\mathbf{q} \in (P_i \cap \mathcal{W}_o)} f(\|\mathbf{p}_i - \mathbf{q}\|) \Phi(\mathbf{q}) \, d\mathbf{q}. \tag{6}$$

However, the above cost functions ignore the underlying IDF, which hasn't been completely observed. Although the robots have access to only the observed IDF, they can still learn to provide more efficient coverage by exploiting its underlying structure. Hence, we evaluate the proposed LPAC architecture on the entire IDF, as given by (4). As discussed in Section V-A, we use a clairvoyant algorithm that has knowledge of the entire IDF for imitation learning, and in Section VI, the evaluations show that the LPAC architecture performs significantly better than CVT algorithms in terms of the global cost function (4).

IV. Learnable PAC Architecture

We propose a learnable Perception-Action-Communication (LPAC) architecture with a Graph Neural Network (GNN) for

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.



(a) Observed IDF (b) Boundary (c) Neighbors (x) (d) Neighbors (y)

Fig. 3. The four channels of the CNN input image in the perception module. All channels are ego-centric to the robot and are of size 32×32. The first channel (a) represents the IDF observed by the robot in its local vicinity. The second channel (b) represents the boundary of the environment. It has non-zero values only when the robot is close to the boundary. The third (c) and fourth (d) channels represent the positions of the neighbors of the robot. For each neighbor, the pixels in the channels corresponding to the relative position of the neighbor have non-zero values. The channel (c) represents the x-coordinates of the neighbors, and the other channel (d) represents the y-coordinates, both normalized by the communication range.

the coverage control problem. The three modules of the LPAC architecture are executed on each robot in a decentralized manner. The input to the architecture is the local IDF observed by the robot, and the output is the velocity control actions for the robot. The collaboration in the robot swarm is achieved using the communication module, which is responsible for exchanging information between robots. The following subsections describe the three modules of the LPAC architecture for the coverage control problem in detail.

A. Perception Module

The perception module is composed of a convolutional neural network (CNN) that takes a four-channel image as input and generates a 32-dimensional feature vector. The channels are grids of pre-defined size (32×32) and are centered around the current position of the robot. The first and the second channels are local representations of the importance and boundary maps, respectively. The other two channels, neighbor maps, encode the relative positions of the neighboring robots. Fig. 3 shows an example of the four-channel image, which forms the input to the CNN.

The perception module is responsible for processing data acquired by robot sensors with a limited field of view. We maintain an *importance map* to represent the IDF sensed by the robot. Each pixel in the importance map is assigned an importance value if it has been sensed by the robot and is assigned a value of 0 otherwise. The first channel of the input image is generated by extracting a local map of a pre-defined size (256×256) from the importance map centered around the current position of the robot and, thereafter, downsampling it to the size of the channel (32×32) using bilinear interpolation.

The perception module also maintains a *boundary map* to represent the boundaries of the environment. Regions outside the environment boundaries are assigned a value of 1, and regions inside the environment boundaries are assigned a value of 0. The boundary map is used so that the policy can learn to avoid collisions with the environment boundaries. Similar to the importance map, the boundary map represents a local region of a predefined size (256×256) centered at the current position of the robot. This map is also downsampled to the size of the channel (32×32) using bilinear interpolation. In general, an obstacle map, similar to the boundary map, can be maintained to represent the obstacles in the environment. Such a map can be generated using SLAM or other obstacle detection techniques.

The other two channels, *neighbor maps*, encode the relative positions of the neighboring robots. These relative positions can be obtained by the sensors on the robot or by exchanging information with the neighboring robots using the communication module. The resolution of the channel is set to the ratio of the communication range and the size of the channel. A pixel, subject to the resolution constraint, is assigned a value if a neighboring robot is present in the corresponding region and is assigned a value of 0 otherwise. A large communication range relative to the channel size makes the resolution very coarse, which, in turn, results in a loss of accuracy of the relative positions of neighboring robots. To mitigate this issue, we have two channels for the neighbor maps; one channel encodes the relative x coordinate, while the other is for the relative y coordinate. We normalize the relative coordinates by the communication range. If two or more neighboring robots are present in the same region of a pixel, the pixel is assigned the sum of the normalized relative coordinates of the neighboring robots. The neighbor maps are designed in this way to ensure that they are *permutation invariant*, i.e., the relative positions of the neighboring robots are independent of the order in which they are encoded in the map.

These four channels are concatenated to form the input to the CNN. The CNN is composed of three sequences of a convolutional layer followed by batch normalization [49] and a leaky ReLU pointwise nonlinearity. The convolution layers have a kernel size of 3×3 and a stride of one with zero padding and generate 32 output channels. The output of the last sequence is flattened and passed through a linear layer with a leaky ReLU activation to generate a 32-dimensional feature vector. The feature vector is then sent to the communication module for further processing.

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

B. Communication with Graph Neural Networks

The communication module is responsible for exchanging information between robots and enabling collaboration in the robot swarm. The main component of the module is a GNN, which is a layered information processing architecture that operates on graphs and makes inferences by diffusing information across the graph. In the LPAC architecture, the graph for GNN is defined by the communication graph of the robot swarm. The vertices $\mathcal{V}$ of the graph correspond to the robots, and the edges $\mathcal{E}$ represent the communication links between the robots, as discussed in Section III-A.

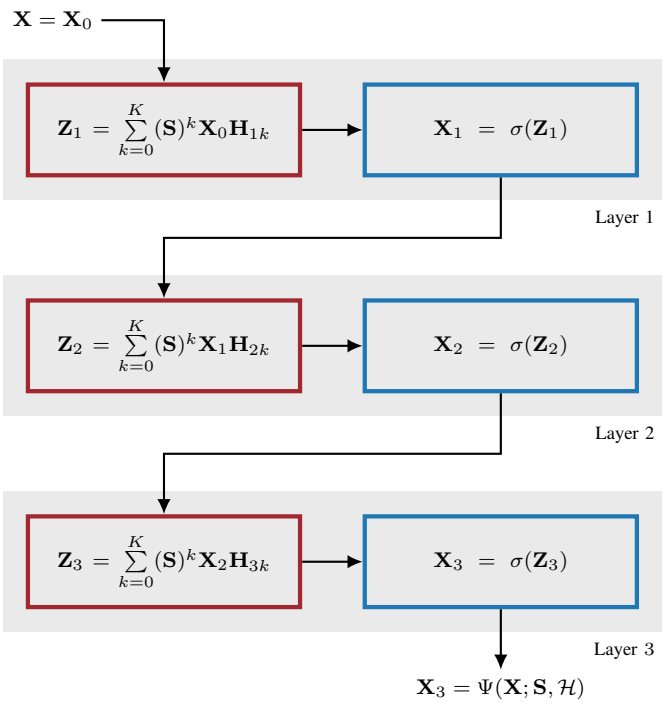


Fig. 4. An example of the GNN architecture used in the communication module. The architecture is composed of $L = 3$ layers of graph convolution filters (red boxes) followed by pointwise nonlinearities (blue boxes).

Our GNN architecture is a layered composition of L *graph convolution filters* [13] with ReLU as the pointwise nonlinearity, as shown in Fig. 4. Each graph convolution filter is parameterized by K hops of message diffusion and is a polynomial function of the *shift operator* $\mathbf{S} \in \mathbb{R}^{N \times N}$, which is a matrix representation of the communication graph for N robots. The elements $[\mathbf{S}]_{ij}$ can be non-zero only if $(i, j) \in \mathcal{E}$. There are many ways to define the shift operator $\mathbf{S}$, such as the adjacency matrix, the Laplacian matrix, or the normalized Laplacian matrix of the communication graph. In our models, we use the normalized adjacency matrix as the shift operator, which is defined as:

$$\mathbf{S} = \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}. \tag{7}$$

Here, $\mathbf{A}$ is the adjacency matrix, and $\mathbf{D}$ is the diagonal degree matrix.

The input to the GNN is a collection of features $\mathbf{X}_0 \in \mathbb{R}^{N \times d_0}$, where each row $\mathbf{x}_i$ is the output of the perception module, augmented with the position of the robot normalized by the environment size, for robot $i \in \mathcal{V}$ and d_0 is the dimension of the feature vector. Let the dimension of layer l be d_l, then the learning weight parameters of the GNN are given by:

$$\mathbf{H}_{lk} \in \mathbb{R}^{d_{(l-1)} \times d_l}, \forall l \in \{1, \cdots, L\}, \forall k \in \{0, \cdots, K\} \tag{8}$$

The output $\mathbf{Z}_l$ of the convolution layer is a polynomial function of the shift operator $\mathbf{S}$ and is processed by the pointwise nonlinearity to generate the input to the next layer:

$$\mathbf{Z}_l = \sum_{k=0}^{K} (\mathbf{S})^k \mathbf{X}_{l-1} \mathbf{H}_{lk}, \quad \mathbf{X}_l = \sigma(\mathbf{Z}_l). \tag{9}$$

Here, we set $(\mathbf{S})^0$ as the identity matrix of size $N \times N$. The final output of the GNN is $\mathbf{X}_L$, and the architecture is denoted by $\Psi(\mathbf{X}; \mathbf{S}, \mathcal{H})$. In our models, we use $L = 5$ graph convolution layers with $d_0 = 34$ and $d_l = 256$, $\forall l \in \{1, \cdots, L\}$.

Distributed Implementation: The GNN architecture inherently allows a distributed implementation of the communication module. To see this, consider the computation $\mathbf{Y}_{lk} = (\mathbf{S})^k \mathbf{X}_{l-1}$, where $\mathbf{X}_{l-1}$ is the input to the l-th layer of the GNN. The computation can be written recursively as $\mathbf{Y}_{lk} = \mathbf{S}\mathbf{Y}_{l(k-1)}$, with $\mathbf{Y}_{l0} = \mathbf{X}_{l-1}$. For a robot i, the corresponding vector in $\mathbf{Y}_{lk}$ is given by:

$$(\mathbf{y}_i)_{lk} = [\mathbf{Y}_{lk}]_i = \sum_{j \in \mathcal{N}(i)} s_{ij} (\mathbf{y}_j)_{l(k-1)}, \tag{10}$$

where $\mathcal{N}(i)$ is the set of neighbors of robot i. The above equation uses the fact that the shift operator $\mathbf{S}$ is a matrix representation of the communication graph, and hence, $s_{ij} = [\mathbf{S}]_{ij}$ is non-zero only if $(i, j) \in \mathcal{E}$ or equivalently, $j \in \mathcal{N}(i)$. The equation also precisely defines the information to be exchanged between neighboring robots. Note that the computation of $(\mathbf{y}_i)_{lk}$ requires $(\mathbf{y}_j)_{l(k-1)}$ for all $j \in \mathcal{N}(i)$. The information sent by the robot i, also known as *aggregated message* [11], is then a collection of vectors $\mathbf{Y}_i$:

$$\mathbf{Y}_i = \{(\mathbf{y}_i)_{lk}\}, \quad \forall k \in \{0, \ldots, K-1\}, \; l \in \{1, \ldots, L\}. \tag{11}$$

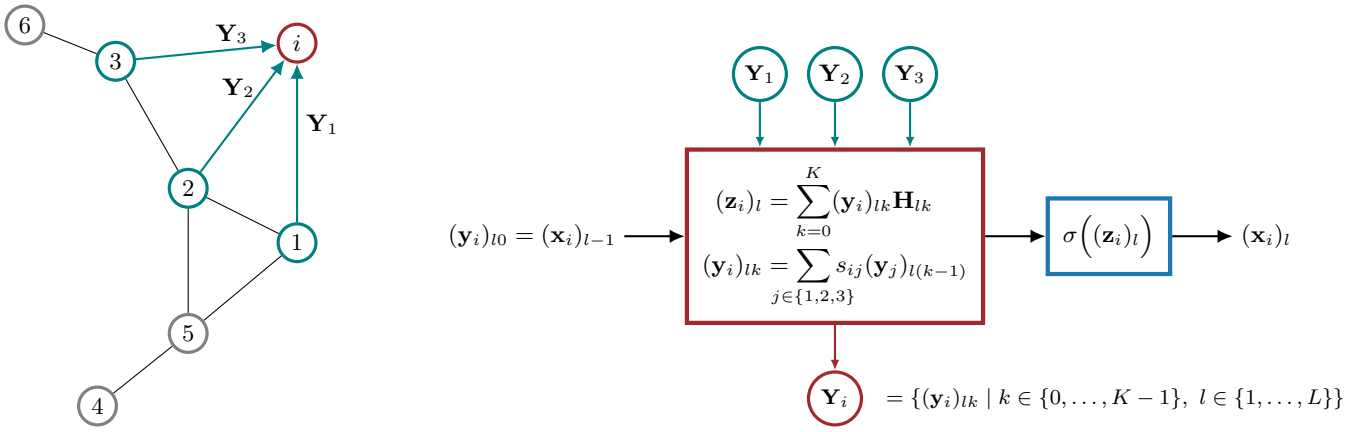
The output of the graph convolution filter for robot i is then given by:

$$(\mathbf{z}_i)_l = \sum_{k=0}^{K} (\mathbf{y}_i)_{lk} \mathbf{H}_{lk}. \tag{12}$$

Finally, pointwise nonlinearity is applied to generate the output $(\mathbf{x}_i)_l$ of the l-th layer for robot i. The distributed implementation of the GNN architecture is illustrated in Fig. 5.

In the actual deployment of the architecture, the module additionally maintains buffers to send and receive information from the neighboring robots using radio communication. In general, the communication is asynchronous, i.e., the robots may receive messages from the neighboring robots at different time steps. Furthermore, the communication module may also run concurrently with the perception module, such that the robots may receive messages from the neighboring robots while processing the sensor data. We refer the reader to [23] for a detailed discussion on asynchronous and distributed implementations of the GNN architecture.

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.



(a) Communication graph (b) Graph convolution filter (red box) with pointwise nonlinearity (blue box)

Fig. 5. Distributed implementation of the GNN architecture for a robot i. (a) The communication graph highlights the neighboring robots, i.e., $\mathcal{N}(i) = \{1, 2, 3\}$. The robot i receives aggregated messages $\mathbf{Y}_j = \{(\mathbf{y}_j)_{lk}\}, \forall j \in \{1, 2, 3\}$ from the neighboring robots. (b) For a layer l of the GNN, the output of the previous layer $(\mathbf{x}_i)_{l-1}$ and the aggregated messages are processed by the graph convolution filter to generate the output $(\mathbf{z}_i)_l$, which is then processed by the pointwise nonlinearity to generate the output $(\mathbf{x}_i)_l$. The convolution filter also generates the aggregated message $\mathbf{Y}_i$ to be sent to the neighboring robots.

C. Action Module

The action module is responsible for generating velocity control actions for the robot. It uses a shallow multi-layer perceptron (MLP) with ReLU activations to process the feature vector generated by the communication module. The MLP has two layers with 32 output channels each, and the final output of the MLP is processed by a linear layer to generate the velocity control actions for the robot. The linear layer has two output channels corresponding to the x and y components of the velocity control action; it is used as the final layer of the MLP to scale the output to the range of the velocity control action. The velocity control actions are sent to a low-level controller to generate the actual control commands for the robot actuators.

The action module may, in general, be composed of additional functionalities that modify the velocity actions before sending them to the low-level controller. For example, depending on the type of robot, such as a ground robot or a quadrotor, the velocity commands may need to be modified to account for the dynamics of the robot. The velocity commands may also need to be modified to ensure that the robots do not collide with each other and the boundaries of the environment.

V. Environment and Imitation Learning

In this section, we discuss the clairvoyant algorithm, based on the centroidal Voronoi tessellation, used to generate the dataset for imitation learning of the LPAC architecture. We also detail baseline algorithms that operate with limited sensing and communication capabilities used to evaluate the performance of the LPAC architecture. Finally, we discuss the simulation environment and the dataset generation process for imitation learning.

A. Centroidal Voronoi Tessellation

We now discuss an iterative gradient descent algorithm that uses the centroidal Voronoi tessellation (CVT) to generate a robot configuration that provides good coverage of the environment. The algorithm, also referred to as Lloyd's algorithm [30], [1], is widely used to solve the coverage control problem. It relies on computing the Voronoi partition (or tessellation) of the region with respect to the locations of the robots. For N robots, the Voronoi partition can be computed in $\mathcal{O}(N \log N)$ time using the sweep line algorithm [46], and efficient implementations [50] are available. We restate the definition of the Voronoi partition $\mathcal{P}$ (Section III):

$$\mathcal{P} = \{P_i \mid i \in \mathcal{V}\}, \quad \text{where}$$
$$P_i = \{\mathbf{q} \in \mathcal{W} \mid \|\mathbf{p}_i - \mathbf{q}\| \leq \|\mathbf{p}_j - \mathbf{q}\|, \forall j \in \mathcal{V}\}.$$

Given N robots operating in a workspace $\mathcal{W}$ with a convex boundary, an observed subset of the workspace $\mathcal{W}_o \subseteq \mathcal{W}$, an importance density field (IDF) $\Phi : \mathcal{W} \to \mathbb{R}_{\geq 0}$, and a Voronoi cell P_i for robot i, we can compute the generalized mass, centroid, and the polar moment of inertia as:

$$\begin{aligned} m_i &= \int_{\mathbf{q} \in (P_i \cap \mathcal{W}_o)} \Phi(\mathbf{q})\, d\mathbf{q}, \\ \mathbf{c}_i &= \frac{1}{m_i} \int_{\mathbf{q} \in (P_i \cap \mathcal{W}_o)} \mathbf{q} \Phi(\mathbf{q})\, d\mathbf{q}, \quad \text{and} \\ I_i &= \int_{\mathbf{q} \in (P_i \cap \mathcal{W}_o)} \|\mathbf{q} - \mathbf{c}_i\|^2 \Phi(\mathbf{q})\, d\mathbf{q}. \end{aligned} \tag{13}$$

The objective function (4) for the coverage control problem, with $f(\|\mathbf{p}_i - \mathbf{q}\|) = \|\mathbf{p}_i - \mathbf{q}\|^2$, can be rewritten as:

$$\begin{aligned} \mathcal{J}(\mathcal{P}) &= \sum_{i \in \mathcal{V}} \int_{\mathbf{q} \in (P_i \cap \mathcal{W}_o)} \|\mathbf{p}_i - \mathbf{q}\|^2 \Phi(\mathbf{q})\, d\mathbf{q} \\ &= \sum_{i \in \mathcal{V}} I_i + \sum_{i \in \mathcal{V}} m_i \|\mathbf{p}_i - \mathbf{c}_i\|^2. \end{aligned} \tag{14}$$

Taking the partial derivative of the objective function (14) with respect to the location of the robot i, we get:

$$\frac{\partial \mathcal{J}(\mathcal{P})}{\partial \mathbf{p}_i} = 2 m_i (\mathbf{p}_i - \mathbf{c}_i) \tag{15}$$

The partial derivatives vanish at the centroid of the Voronoi cell, i.e., $\mathbf{p}_i = \mathbf{c}_i$; thus, the centroid of the Voronoi cell is

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

the local minimum of the objective function (14). Hence, we can write a control law [1] that drives the robot towards the centroid of the Voronoi cell as:

$$\mathbf{u}_i = \dot{\mathbf{p}}_i = -k(\mathbf{p}_i - \mathbf{c}_i). \tag{16}$$

Here, k is a positive gain for the control law. The control law in (16) has nice convergence properties; it is guaranteed to converge to a local minimum of the objective function [1].

The algorithm can now be expressed as an iteration of the following steps until convergence or for a maximum number of iterations:

1) Compute the Voronoi partition $\mathcal{P}$ of the region with respect to the locations of the robots.
2) Compute the mass centroids $\mathbf{c}_i$ for each Voronoi cell P_i.
3) Move each robot towards the centroid of the Voronoi cell using the control law (16).

We can define different variants of the above algorithm with the same control law (16), depending on the observed workspace $\mathcal{W}_o$ and information available to the robots for computing the Voronoi partition and the centroids. In this article, we refer to the algorithms as variants of CVT.

Clairvoyant: The clairvoyant is a centralized algorithm based on CVT. It has perfect knowledge of the positions of all the robots at all times. Thus, it can compute the exact Voronoi partition. It also has complete knowledge of the IDF for the centroid computation for each Voronoi cell, i.e., $\mathcal{W}_o = \mathcal{W}$. Although the algorithm is not optimal, it generally computes solutions that are very close to the optimal solution.

Centralized CVT (C-CVT): The C-CVT is also a centralized algorithm based on CVT. Similar to the clairvoyant algorithm, it knows the positions of all the robots and computes the exact Voronoi partition. However, unlike the clairvoyant algorithm, the C-CVT can access a limited IDF. It operates on the cumulative knowledge of the IDF of all the robots, sensed up to the current time step, i.e.,

$$\mathcal{W}_o = \bigcup_{i \in \mathcal{V}} \mathcal{W}_o^{(i)} \subseteq \mathcal{W},$$

where $\mathcal{W}_o^{(i)} \subseteq \mathcal{W}$ is the workspace observed by robot i along its entire trajectory.

Thus, the amount of information available to the C-CVT is dependent on the sensor radius of the robots and the trajectory taken by the robots.

Decentralized CVT (D-CVT): The D-CVT is the decentralized version of the C-CVT algorithm. Here, each robot uses the positions of neighboring robots, i.e., the robots within its communication range, to compute the Voronoi partition. Furthermore, each robot has restricted access to the IDF sensed by itself up to the current time, i.e., $\mathcal{W}_o = \mathcal{W}_o^{(i)}$. Thus, the D-CVT algorithm uses only the local information available to each robot to take control actions.

As one can expect from the amount of information available to each algorithm, the clairvoyant algorithm is the best-performing algorithm, followed by the C-CVT and the D-CVT algorithms. The clairvoyant algorithm is used to generate the dataset for training, and the C-CVT and D-CVT baseline algorithms are used to evaluate the performance of the LPAC architecture.

B. Coverage Control Environment

The environment is a 2D grid world of size 1024×1024 cells with a resolution of $1\,\mathrm{m} \times 1\,\mathrm{m}$ per cell. The environment is populated with M features of interest that are randomly placed in the environment. The IDF is generated by defining a 2-D Gaussian distribution as a probability density function centered at the feature of interest, with a randomly generated standard deviation $\sigma \in [40, 60]$ and a randomly generated scale factor $\alpha \in [6, 10]$. The Gaussian distribution is set to 0 beyond a distance of 2σ from the center of the feature of interest to avoid any detection of features from a faraway location. The value of a cell in the IDF is set to the sum of the integrals of the Gaussian distributions over the area of the cell for all features of interest. Finally, the values are normalized to have a maximum value of one.

The environment is populated with N robots that are randomly placed in the environment according to a uniform probability distribution. The robots have a square sensor field of view with a side length of $64\,\mathrm{m}$. The sensor field of view is centered at the location of the robot. A disk-shaped sensor field-of-view can be approximated by a square sensor field-of-view with a side length of the diameter of the disk and setting the values outside the disk to zero. The robots have a communication range of $128\,\mathrm{m}$, i.e., they are able to communicate with each other if within this range. The maximum speed of the robots is set to $5\,\mathrm{m\,s^{-1}}$.

C. Data Generation and Imitation Learning

The learning pipeline comprises data generation, imitation learning, and evaluation. Imitation learning is used to train the LPAC architecture to mimic the behavior of the clairvoyant algorithm, which serves two main purposes: (i) drive the simulation by taking actions during data generation, and (ii) provide near-optimal velocity actions for the LPAC architecture to learn from. Although the clairvoyant algorithm has knowledge of the entire IDF, the input to the LPAC architecture for both training and evaluation is always limited to the observed IDF with the limited sensing capabilities of the robots. Each robot i independently runs the same LPAC policy using only the observed workspace $\mathcal{W}_o^{(i)}$ made by the robot i and GNN-based abstract information communicated by the neighboring robots. There are two primary advantages to this approach: First, designing highly specialized algorithms with limited sensing capabilities is challenging, which is circumvented by using the clairvoyant algorithm. Second, as the results show in Section VI, the LPAC policy is able to outperform the centralized algorithm (C-CVT) with limited observations, which would not have been possible if the LPAC architecture had been trained using the C-CVT algorithm.

Using the clairvoyant algorithm, the simulation environment evolves in discrete time steps of $0.2\,\mathrm{s}$, i.e., with a frequency of $5\,\mathrm{Hz}$, and the maximum distance that the robots are able to move is $1\,\mathrm{m}$ in each time step. The algorithm is run until convergence or for a maximum of 1000 iterations. At each iteration, for each robot, the four-channel input to the CNN of the perception module (Section IV-A), along with the control actions and positions of robots, are stored as a state-action

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

pair. Such state-action pairs are stored every five iterations of the algorithm. We additionally generate configurations at the converged state of the CVT algorithm to help the imitation learning algorithm learn the converged state. In total, 100,000 data points are generated, each containing state-action pairs for 32 robots.

The training is performed using the Adam optimizer [51] in Python using PyTorch [52] and PyTorch Geometric [53]. We use a batch size of 750 and train the network for 100 epochs. The learning rate is set to 10^{-4}, and the weight decay is set to 10^{-3}. The mean squared error (MSE) loss is used as the loss function, where the target is the output of the clairvoyant algorithm, and the prediction is the output of the LPAC architecture. The training and validation loss curves are shown in Fig. 6. The model with the lowest validation loss is selected as the final model.

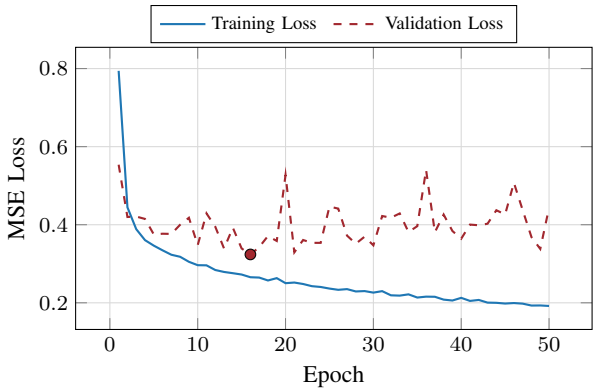


Fig. 6. Evolution of the training and validation Mean Squared Error (MSE) loss over epochs for the primary LPAC-K3 model trained on 1024×1024 grid world with 32 robots. The training loss steadily decreases, indicating the model is fitting the data more closely, while the validation loss remains higher and fluctuates, indicating the model is not overfitting. The highlighted point marks the epoch (#16) at which the validation loss is minimized, which is used to select the final model.

VI. Results

This section presents empirical evaluations of the proposed LPAC architecture for the coverage control problem. Our experiments establish the following:

1) The model **outperforms the baseline** centralized and decentralized CVT algorithms. It learns to share relevant abstract information, which is generally challenging to design for decentralized algorithms.
2) The **ablation study** shows: (a) The model with $K = 3$ in the GNN architecture performs better than the ones with $K = 1$ and $K = 2$. Thus, the architecture is able to exploit the propagation of information over a larger portion of the communication graph. (b) The neighbor maps and the normalized robot positions are essential for improving the performance of the model.
3) The model **generalizes** to environments with varying numbers of robots and features, especially when they are larger than the training distribution.
4) The model **transfers** well to larger environments with a larger number of robots while keeping the ratio of robots to environment size the same.
5) The model is **robust to noisy position estimates**, indicating that the model can be deployed on real-world robots with noisy sensors.
6) Models trained with different communication ranges perform well for up to half the environment size, and the performance is better than the CVT algorithms for all ranges.
7) Even though the model is trained on synthetic datasets with randomly generated features, it performs well on **real-world datasets** without further training.

We implemented an open-source platform[1] for simulating the coverage control problem with the LPAC architecture. The platform is implemented in `C++`, `CUDA`, and `Python` using the `PyTorch` library. The `Python` interface is used for training and rapid prototyping, while the evaluation and deployment are done using the `C++` interface.

A. Comparison to Baseline Algorithms

We evaluate our learned LPAC models against the decentralized and centralized centroidal Voronoi tessellation (CVT) algorithms, as discussed in Section V-A. We also compare the models against the clairvoyant algorithm, which has knowledge of the entire IDF and the positions of all robots, and it can compute near-optimal actions for each robot.

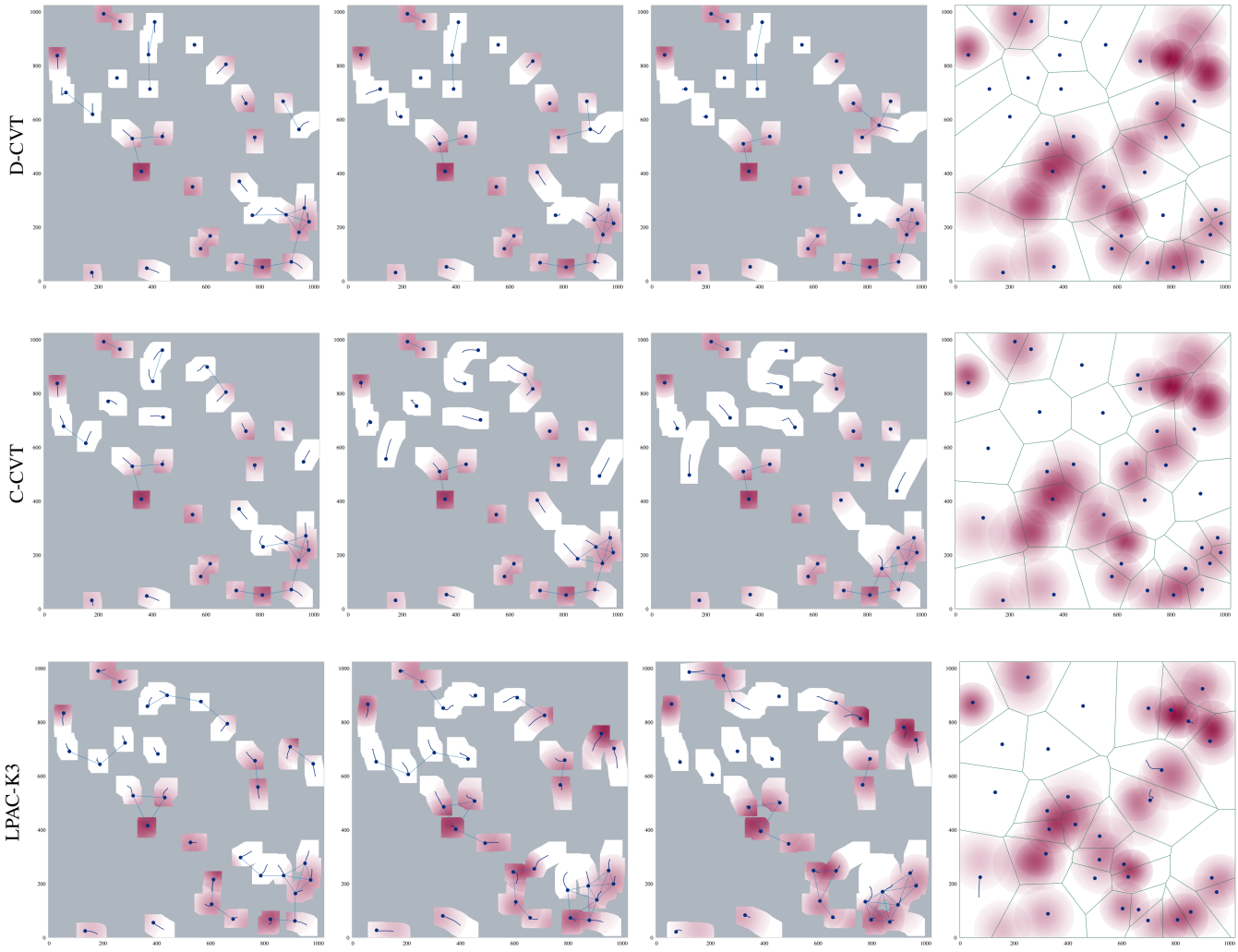
We evaluated the performance of the LPAC models and the baseline algorithms on 100 environments, each of size $1024\,\mathrm{m} \times 1024\,\mathrm{m}$ with 32 robots. The initial positions of the robots are sampled uniformly at random in each environment. The IDF in each environment is generated by 32 Gaussian distributions at random locations in the environment. Fig. 7 shows the progression of the controllers for various time steps in a random environment. Since the CVT-based algorithms converge very fast, we show the state of the system for the beginning of the episode, i.e., time steps 60, 120 and 180. For this environment, the LPAC-K3 controller performs better than decentralized CVT at all timesteps and better than centralized CVT after 12 time steps.

Fig. 8 shows the performance of the LPAC models and the baseline algorithms over time for 100 environments. The features used for the IDF and the initial positions of the robots are uniformly sampled at random. For performance evaluation, we measure the coverage cost relative to the initial state of the environment, i.e., we normalize the coverage cost at each timestep by the initial coverage cost. This provides a consistent baseline that highlights how effectively each method reduces the coverage cost from its starting conditions, thereby reducing the impact of the initial conditions on the comparison.

The solid lines in the figure show the normalized coverage cost averaged over all environments, and the shaded regions show the standard deviation of the coverage cost. The average performance of the LPAC model, with $K = 3$ hops in the GNN architecture, is significantly better than both the decentralized and centralized CVT algorithms. The standard deviation of the LPAC model is slightly higher than the CVT algorithms, but the spread of the standard deviation is mostly below that of the decentralized CVT algorithm.

[1] https://github.com/KumarRobotics/CoverageControl

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.



(a) Time Step = 60 (b) Time Step = 120 (c) Time Step = 180 (d) Time Step = 900

Fig. 7. Progression of the decentralized and centralized CVT algorithms (D-CVT and C-CVT) and the LPAC-K3 model for a sample environment. The features and the initial positions of the robots are sampled uniformly at random. The first three columns show the cumulative observations of all robots up to time steps 60, 120, and 180, respectively. The robots are shown as blue discs with blue lines showing the trajectory for the past 40 time steps. The light blue lines show communication links between robots based on a range of 128 m. The last column shows the positions of the robots at time step 900 with the Voronoi cells of the robots. The entire IDF is shown in the final time step.

Fig. 9 shows the percentage of the environment observed by the robots over time. There is a correlation between the observed area and the coverage cost, shown in Fig. 8. The decentralized and centralized CVT algorithms observe only a small portion of the environment as they quickly converge to local minima based on their immediate local observations. In contrast, the clairvoyant algorithm, which has complete knowledge of the IDF, strategically spreads the robots across the environment to reach near-optimal locations, resulting in a higher observed area. Note that the observed area for the clairvoyant algorithm is calculated based on what the robots would have observed during their motion, as the algorithm itself has complete knowledge of the IDF. Initially, the observed area of the LPAC model is lower than that of the clairvoyant algorithm, but it increases over time since the LPAC model does not have an explicit convergence criterion. Similar to the clairvoyant algorithm, the LPAC-K3 policy effectively distributes the robots and explores more of the environment, achieving a greater observed area compared to the CVT algorithms.

Fig. 10 shows the number of environments, out of the 100 environments, for which each controller performs the best. The LPAC-K3 model performs the best for about 50% of the environments within the first few time steps and reaches above 85% after 500 time steps. This indicates that the LPAC model can learn a policy that performs well in most environments over time.

These results establish that the LPAC architecture is suitable for the coverage control problem and outperforms the decentralized CVT algorithm. Even though the centralized CVT algorithm has knowledge of all the observations of the robots, the LPAC model still outperforms the centralized CVT algorithm. This indicates that the LPAC model learns to share abstract information about the environment with other

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

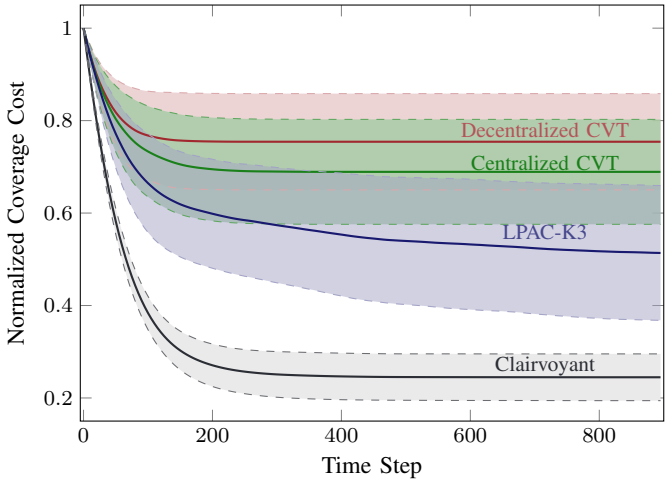


Fig. 8. Evaluation of the LPAC model with respect to CVT-based algorithms: The coverage cost is normalized by the cost at the initialization of the environment. Hence, the cost is 1 at time step 0. The controllers are run for 100 environments for 900 time steps each. For each controller, the mean performance over all environments is shown as a solid line, and the standard deviation is shown as a shaded region. The LPAC model outperforms both the centralized and the decentralized CVT algorithms.

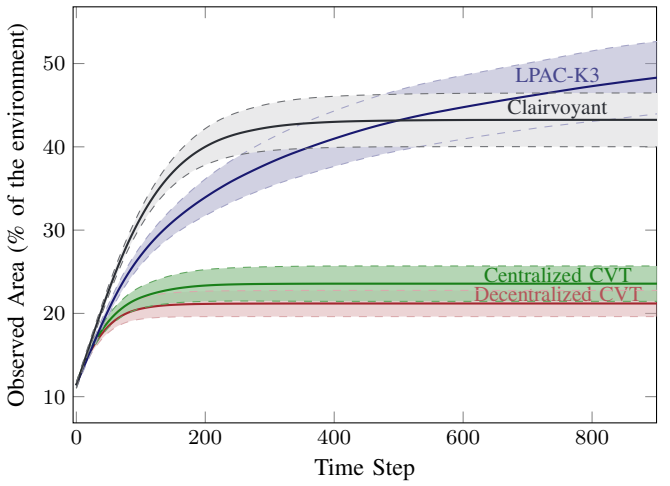


Fig. 9. Percentage of the environment observed over time: The decentralized and centralized CVT algorithms observe a small portion of the environment as they quickly converge to local minima based on local observations. The clairvoyant algorithm, leveraging complete knowledge of the importance density function (IDF), spreads the robots effectively across the environment, resulting in a higher observed area. The LPAC-K3 policy, though initially observing less than the clairvoyant algorithm, gradually increases the observed area over time by distributing the robots more effectively, ultimately achieving a higher observed area compared to the CVT algorithms.

robots and propagates this information over the communication graph, resulting in improved performance.

B. Ablation Study

We performed an ablation study of the LPAC architecture to understand the importance of the different components of the architecture. For the first ablation study, we trained LPAC models with different numbers of hops in the GNN architecture, ranging from 1 to 3. A $K = 3$ hop model

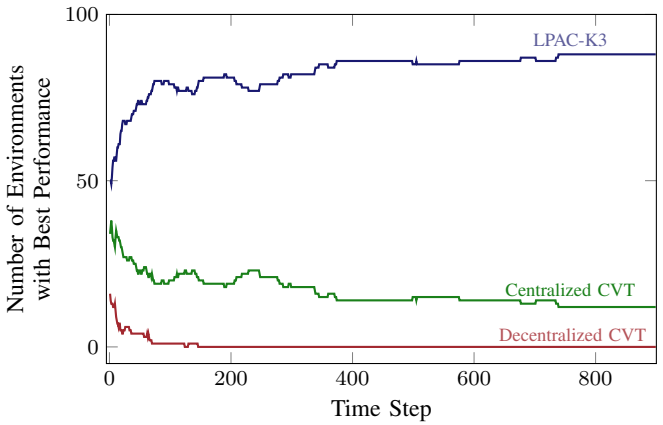


Fig. 10. Number of environments for which each controller performs the best (after the first time step): The controllers are run for 100 environments for 900 time steps each. The environments are randomly generated—the center of the IDF features and the initial positions of the robots are sampled uniformly at random. The LPAC-K3 model performs best for about 50% of the environments within the first few time steps and reaches above 85% after 500 time steps.

enables the diffusion of information over a larger portion of the communication graph, whereas a $K = 1$ hop model limits diffusion to the immediate neighbors of a robot. Fig. 11 shows the performance of the LPAC models with different numbers of hops in the GNN architecture and compares them with the decentralized and centralized CVT algorithms. The LPAC-K1 model performs worse than the other LPAC models and is unstable over time. This is expected, as the model is not able to collaborate with robots that are not within its communication range. The LPAC-K2 and LPAC-K3 models, on the other hand, significantly outperform the CVT algorithms. A large number of hops in the GNN architecture is not necessarily advantageous, as it requires more computation and memory and may become unsuitable for real-time applications.

For the second ablation study, we trained LPAC models without the neighbor maps as inputs to the CNN layer (LPAC-NoNeighborMaps), and without the normalized robot positions as additional features to the GNN layer (LPAC-NoPos). Fig. 12 shows the performance of these ablated LPAC models with respect to the primary LPAC model (LPAC-K3) and the CVT algorithms. The LPAC-NoNeighborMaps model performs significantly worse than the LPAC-K3 model. This indicates that the neighbor maps are essential for the LPAC model to perform well. The LPAC-NoPos model performs slightly worse than the LPAC-K3 model, but the performance is still better than the CVT algorithms. The normalized position of the robot as an additional feature to the GNN layer is, therefore, helpful in improving the performance of the LPAC model. However, the LPAC model can still perform well even when the position of the robot is not available, as in the case of a robot without a GPS or good state estimation.

The ablation study shows that the proposed LPAC architecture is suitable for the coverage control problem, and the different components of the architecture are essential for the performance of the model. In the following subsections, we evaluate the performance of the LPAC architecture, focusing

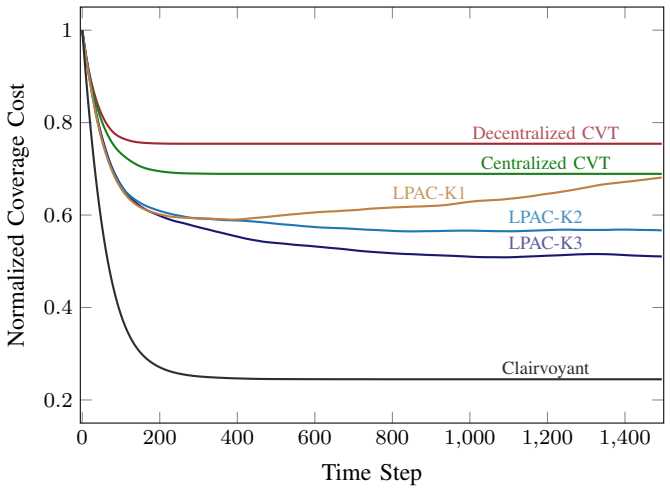


Fig. 11. Performance of LPAC architecture with different number of hops K in the GNN layer: The controllers are run for 100 environments for 1500 time steps each. All the LPAC models perform well in the first 200 time steps, but the LPAC-K3 model with 3 hops in the GNN layer outperforms the other LPAC models in the long run. The LPAC-K1 model with a single hop in the GNN layer is particularly unstable, as it is not able to learn to collaborate with the robots that are not in its communication range. The LPAC-K2 and LPAC-K3 models outperform both the centralized and decentralized CVT algorithms.

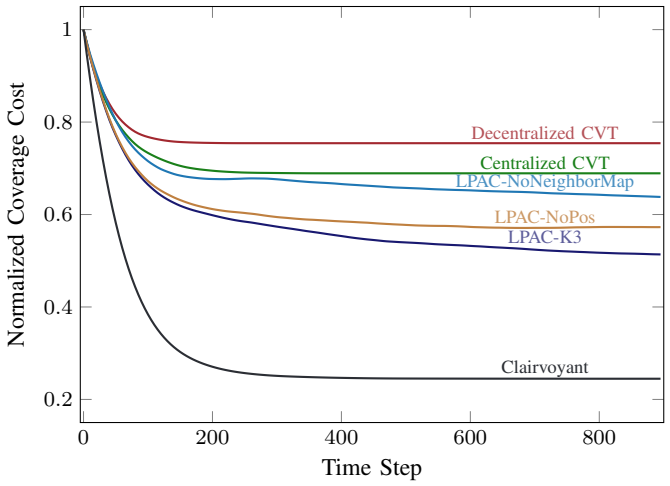


Fig. 12. Ablation study of the LPAC model: The LPAC architecture without the neighbor maps (LPAC-NoNeighborMaps) as inputs to the CNN layer performs significantly worse than the full LPAC model, even though the average performance is better than both the centralized and decentralized CVT algorithms. The LPAC architecture (LPAC-NoPos), without the normalized robot positions as additional features to the GNN layer, performs slightly worse than the full LPAC model. This indicates that even when a global position estimate is not available, the LPAC model can still perform well.

on $K = 3$ hops (LPAC-K3) in the GNN architecture as the primary model.

C. Generalization to Varying Number of Robots and Features

The primary LPAC model (LPAC-K3) was trained on environments with 32 robots and 32 features in a $1024\,\mathrm{m} \times 1024\,\mathrm{m}$ environment. We evaluate the performance of the model on environments with varying numbers of robots and features

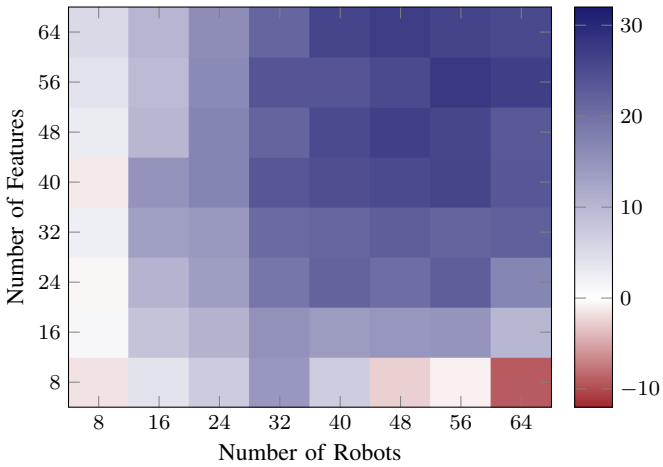


Fig. 13. Evaluation of the LPAC-K3 model on environments with varying numbers of features and robots. The controllers are evaluated in 100 environments for each combination of the number of robots and features. The performance is computed as the percentage improvement of the average cost over the decentralized CVT algorithm. The model performs worse than the decentralized CVT algorithm when the number of robots or features is very small (8 for each). However, the model performs significantly better when the number of robots and features increases.

from 8 to 64, in steps of 8, in the same size environment. For each combination of a number of robots and features, we generated 100 environments with the locations of the features and the initial positions of the robots sampled uniformly at random. The controllers are run for 1500 time steps for each environment. Fig. 13 shows the performance of the LPAC model, with $K = 3$ hops in the GNN architecture, with respect to the decentralized CVT algorithm. The performance is computed as the percentage improvement of the average cost over the decentralized CVT (D-CVT) algorithm:

$$\frac{(\text{Avg. Cost D-CVT}) - (\text{Avg. Cost LPAC-K3})}{(\text{Avg. Cost D-CVT})} \times 100 \quad (17)$$

The LPAC model performs worse than the decentralized CVT algorithm when the number of robots or features is very small (8 for each). The sensor field-of-view of the robots is 64×64, whereas the environment size is 1024×1024. This represents a very small fraction of the environment that the robots can observe, with a ratio of $1 : 256$. When the number of features is very small, large empty regions are problematic as the robots may not see any features, especially in the first few steps, and thus, the LPAC policy fails to take appropriate actions. The LPAC policy does not often see such scenarios during training, and hence, the policy may not perform well in such cases. For example, the LPAC model performs 9% worse than the decentralized CVT algorithm with 8 features, even with 64 robots.

A similar case arises when the number of robots is very low, resulting in a sparse disconnected communication graph. The GNN-based LPAC policy is not effective in such scenarios as the robots are not able to communicate with each other.

However, the model performs significantly better for a larger number of robots and features. Interestingly, the performance of the LPAC model improves for cases beyond the training

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

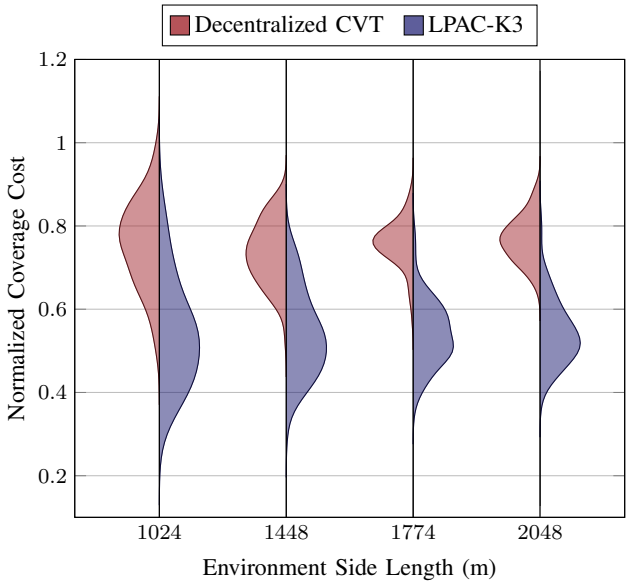


Fig. 14. Scalability of LPAC models: The LPAC-K3 model, trained on environments of size $1024\,\mathrm{m} \times 1024\,\mathrm{m}$ with 32 robots, is executed on larger environments with a larger number of robots while keeping the ratio of robots to environment size the same. In the violin plots, the LPAC-K3 model (blue) consistently outperforms the decentralized CVT algorithm (red) in all cases. The width of the violin plots is proportional to the number of environments for the average coverage cost.

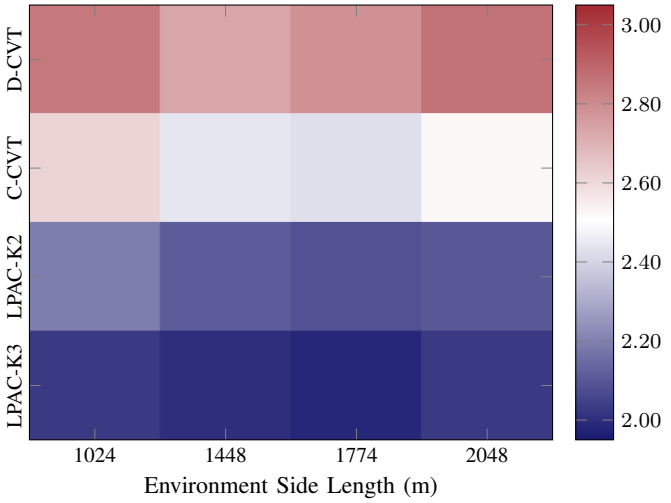


Fig. 15. Performance of coverage control algorithms as a ratio of average costs with respect to the clairvoyant algorithm: The decentralized CVT (D-CVT) and centralized CVT (C-CVT) algorithms and the LPAC-K2 and LPAC-K3 models are evaluated on increasing environment size while keeping the ratio of robots to environment size the same. The LPAC models outperform both the decentralized and centralized CVT algorithms in all cases. The average costs for the D-CVT and the C-CVT algorithms are around 2.8 and 2.5 times that of the clairvoyant algorithm, respectively. The average costs for the LPAC-K3 model are about twice the clairvoyant algorithm, even though the model is decentralized and the IDF is not known *a priori* to the model.

distribution, i.e., when the number of robots and features is larger than 32. With 32 robots and 32 features and beyond, the LPAC model performs at least 20% better than the decentralized CVT algorithm.

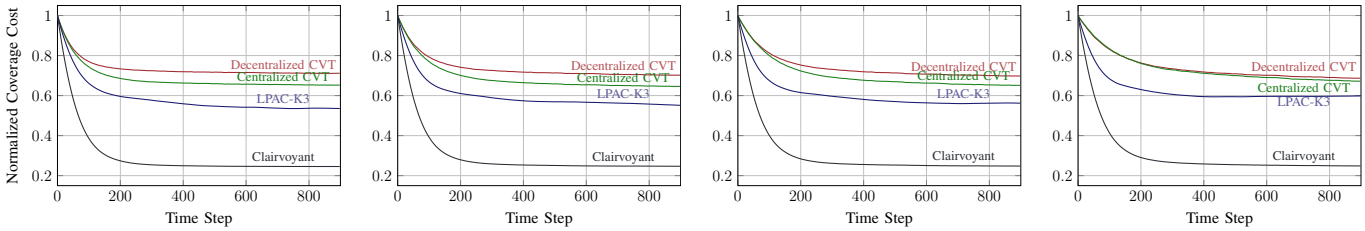
These results demonstrate that the LPAC model can generalize effectively to scenarios featuring substantially more robots and features than those used during training. This capability is significant, as it allows the model that was trained on a sufficiently large, but finite set of conditions, to be reliably applied in environments where the number of robots and features is not fixed in advance and may far surpass the levels encountered during training. These empirical results correlate with the theoretical results on transferability [13], i.e., policies trained on sufficiently large number of robots can be transferred to even larger teams.

D. Transferability to Larger Environments

One of the advantages of using a GNN-based architecture is that it scales well to larger environments and a larger number of robots. We evaluate the primary LPAC model (LPAC-K3), which was trained on environments with 32 robots and 32 features in a $1024\,\mathrm{m} \times 1024\,\mathrm{m}$ environment, on larger environments while keeping the number of robots and features per unit of environment area the same. The model is not retrained or fine-tuned on the larger environments, and the robot parameters, such as the communication range ($128\,\mathrm{m}$) and the sensor field of view ($64\,\mathrm{m} \times 64\,\mathrm{m}$), are kept the same. Fig. 14 shows the performance of the LPAC model with respect to the decentralized CVT algorithm on environments ranging from $1024\,\mathrm{m} \times 1024\,\mathrm{m}$ to $2048\,\mathrm{m} \times 2048\,\mathrm{m}$ with 32 to 128 robots. The shaded regions in the figure show the distribution of the normalized coverage cost over 100 environments. Note that the peaks of the distributions for the LPAC model are at a significantly lower coverage cost than those of the decentralized CVT algorithm. Interestingly, the spread of the distributions for both controllers decreases as the size of the environment increases, even though the ratio of robots to environment size is kept the same. This could be attributed to the fact that the ratio of environment size and the number of robots does not exactly match with the expected degree of a robot in the communication graph—the expected degree of a robot is 1.41 in the $1024\,\mathrm{m} \times 1024\,\mathrm{m}$ environment, and 1.49 in the $2048\,\mathrm{m} \times 2048\,\mathrm{m}$ environment, as computed using a Monte Carlo simulation. The discrepancy is caused because of the boundary limits of the environment, which change the probability of a robot being connected to other robots from a linear relationship to a more complex one.

We also evaluated the centralized CVT algorithm and the LPAC model with $K = 2$ hops in the GNN architecture in the same environments. Similar to LPAC-K3, the LPAC-K2 model was trained on environments with 32 robots and 32 features in a $1024\,\mathrm{m} \times 1024\,\mathrm{m}$ environment. Fig. 15 shows the performance of the controllers in the larger environments. The performance is measured as the ratio of the average coverage cost with respect to the clairvoyant algorithm, i.e., the average cost of the clairvoyant algorithm is 1. Since the clairvoyant algorithm has the knowledge of the entire IDF, the coverage costs of other algorithms are always worse than the clairvoyant algorithm, and the ratio is always greater than 1. The results show that both LPAC-K2 and LPAC-K3 models outperform the decentralized and centralized CVT algorithms for all environment sizes, and the models transfer well to larger

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.



(a) $\epsilon \sim \mathcal{N}(0, 5^2)$ (b) $\epsilon \sim \mathcal{N}(0, 10^2)$ (c) $\epsilon \sim \mathcal{N}(0, 15^2)$ (d) $\epsilon \sim \mathcal{N}(0, 20^2)$

Fig. 16. Evaluation of coverage algorithms with a Gaussian noise ϵ added to the position of each robot, i.e., the sensed position $\bar{\mathbf{p}}_i = \mathbf{p}_i + \epsilon$. The performance of the algorithms, measured as normalized coverage cost in the y-axis, is not significantly affected by the noise, except for the very large noise with a standard deviation of $20\,\mathrm{m}$.

environments. The performance of the LPAC-K3 model is the best of all the controllers, whereas the performance of the LPAC-K2 model is slightly worse than that of the LPAC-K3 model.

These results establish that the LPAC architecture can be trained in a small environment and can be deployed in larger environments with a larger number of robots. The results support the theoretical analysis on the transferability of GNNs [13]. The LPAC architecture utilizes CNN and GNN modules, and the inputs to these modules are designed such that both rely primarily on local, compositional structures rather than fixed global representations. As the number of robots and the environment size grow, each robot's decision-making process remains focused on its immediate neighbors, ensuring that the learning process is independent of global scale. Since the LPAC architecture, particularly GNN, is invariant to the number of vertices (robots), the same learned rules apply seamlessly to larger teams. Moreover, as we scale the number of robots and maintain a constant density, the local neighborhood structure encountered by each robot remains essentially the same, creating a form of regularity. This regularity in the local graph structure ensures that the learned local policies generalize effectively as the system expands.

E. Robustness to Noisy Positions

Real-world robots are subject to noise in their position estimates. This is particularly true for robots that use GPS for localization, as the GPS signal can be affected by the environment, such as tall buildings and the weather. Thus, we evaluate the performance of the LPAC-K3 model with respect to the CVT algorithms with simulated noise in the position of the robots. A Gaussian noise is added to the position of each robot in the simulator. The standard deviation of the Gaussian noise is varied from 5 to 20 meters in steps of 5 meters. Fig. 16 shows the performance of the controllers for different noise levels. The controllers, including LPAC-K3, are not significantly affected by the noise. However, there is still a degradation in the performance of the controllers with increasing noise, especially for the noise with a standard deviation of $20\,\mathrm{m}$. These results indicate that LPAC models, as well as the CVT algorithms, are robust and can be deployed on real-world robots with noisy position estimates.

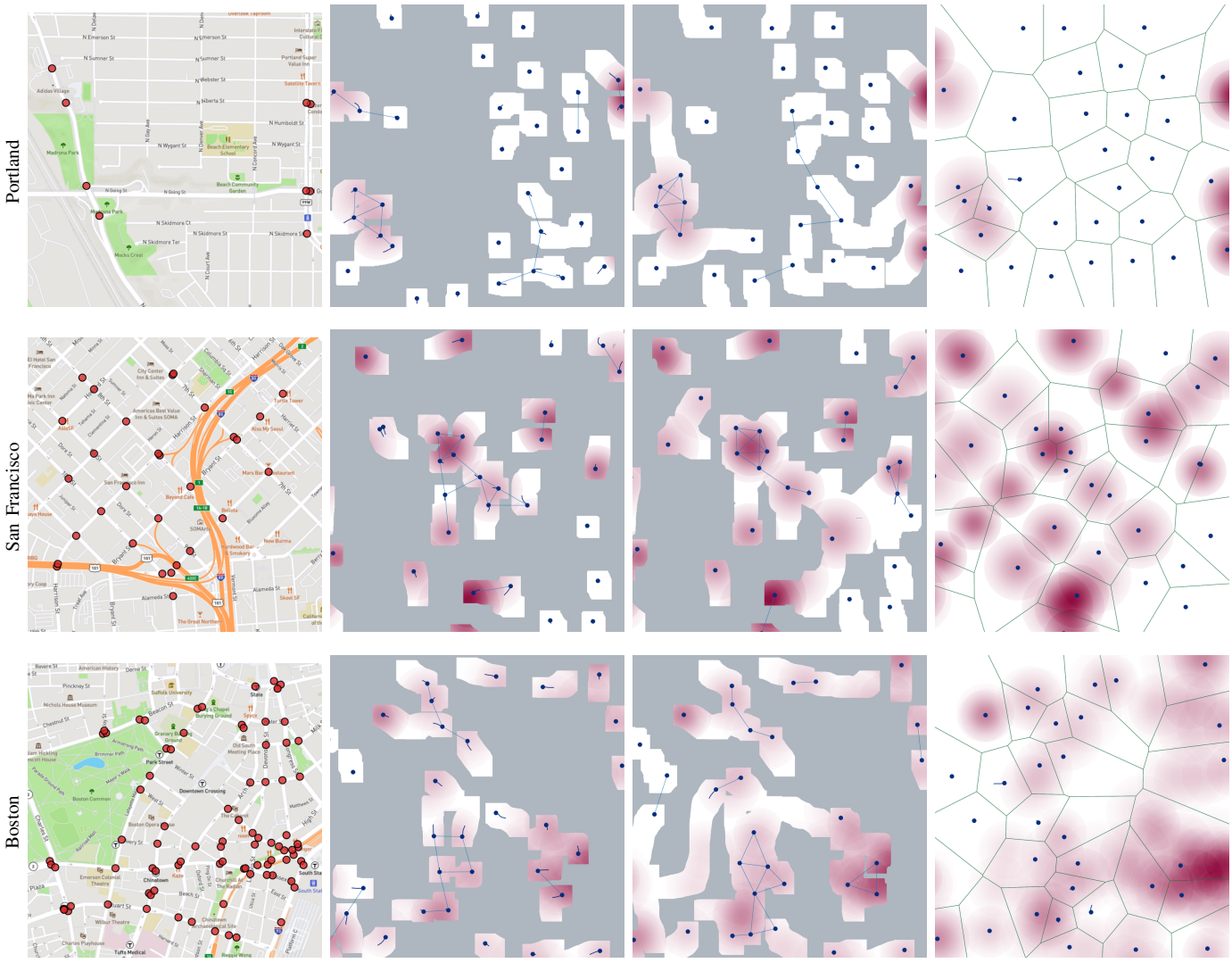
F. Performance on Real-World Datasets

In addition to testing on uniform, randomly generated feature distributions, we also evaluate our LPAC architecture using traffic light datasets derived from real-world traffic light locations from 50 cities in the US. An area of $1024\,\mathrm{m} \times 1024\,\mathrm{m}$ was selected in each city, and the locations of the traffic lights were extracted from the OpenStreetMap database. The traffic light locations form the locations of the features in the IDF. The core idea behind using the traffic light dataset is to assess the generalizability of the LPAC policy under non-uniform and realistically structured feature distributions. In our training and baseline evaluations, features (represented by Gaussian peaks in the IDF) are placed uniformly at random. This setup, while controlled, does not reflect spatial patterns or correlations that often arise in real-world scenarios.

In contrast, the traffic light dataset introduces features located at fixed, non-uniform points that mirror real-world urban layouts. Traffic lights are typically clustered along road networks, follow certain urban planning principles, and occur at predictable intervals rather than scattered randomly. This is more than a trivial variation in feature placement as it forces the policy to address structured complexity rather than the uniform randomness it was exposed to during training.

Fig. 17 shows the progression of the LPAC-K3 model in three environments from the dataset. Since the performance of the coverage control algorithms is sensitive to the initial positions of the robots, we evaluated the performance of the algorithms with 10 different initial positions of the robots. Fig. 19 shows the performance of the LPAC-K3 model with respect to the decentralized and centralized CVT algorithms. The performance is measured as the ratio of the average cost with respect to the clairvoyant algorithm. The environments are split into buckets of size 10 based on the number of features in each environment. The results show that the LPAC-K3 model performs poorly, around 3.5 times the clairvoyant algorithm, on environments with a small number of features, as also observed in Section VI-C. The model, however, performs well in environments with more than 20 features, with an average performance of around 1.9 times the clairvoyant algorithm for more than 30 features.

The results show that the LPAC architecture performs well for environments that may be very different from the training distribution. It is important to note that the LPAC-K3 model was not retrained or fine-tuned on the real-world dataset, and



(a) Map (b) Time Step = 100 (c) Time Step = 600 (d) Time Step = 1200

Fig. 17. Progression of the LPAC-K3 model on three real-world environments: The traffic signals, shown in the first column, form the features for generating the IDF. Each such feature is modeled as a 2-D Gaussian distribution with a uniformly sampled standard deviation $\sigma \in [40, 60]$ and scaled with a value sampled uniformly at random from $[6, 10]$. The number of features for Portland, San Francisco, and Boston are 9, 30, and 86, respectively. The robots in the swarm are initialized at random positions in the environment. The second and third columns show the cumulative observations of all robots up to time steps 100 and 600, respectively. The last column shows the positions of the robots at time step 1200 with the Voronoi cells of the robots on the entire IDF.

the model was trained on synthetic datasets with randomly generated features. Furthermore, the clairvoyant algorithm has knowledge of the entire IDF, and the LPAC model is able to perform well with only local observations of the IDF and communication with neighboring robots. The performance of the learned LPAC policy in the traffic light realistic environments further validates that the LPAC architecture does not merely overfit to synthetic uniform conditions, but can effectively address real-world scenarios with non-uniform feature distributions.

G. Demonstration on Realistic Simulator

We demonstrate our approach on a simulation framework that combines PX4, Gazebo, and ROS2 Humble to facilitate a direct and practical approach to prototyping aerial robots. A large number of platforms use PX4, and thus, using PX4 ensures that code developed in simulation can be transferred to most real platforms with minimal modifications. Gazebo introduces physics-based realism, including collision detection and sensor noise, reducing discrepancies between simulated experiments and real-world field tests.

We deployed the Learnable Perception-Action-Communication (LPAC) policy operating within the integrated simulation environment. In this demonstration, the system relies solely on GPS for localization, although the simulation setup allows for introducing additional sensors and more sophisticated perception methods. The environment is a 250×250 square with 8 robots and 8 features. The effective communication range was set to $64\,\mathrm{m}$ with all other parameters the same. Fig. 18 shows the simulation environment and the coverage of the robots at the final time-step. Note that the performance of any coverage control

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

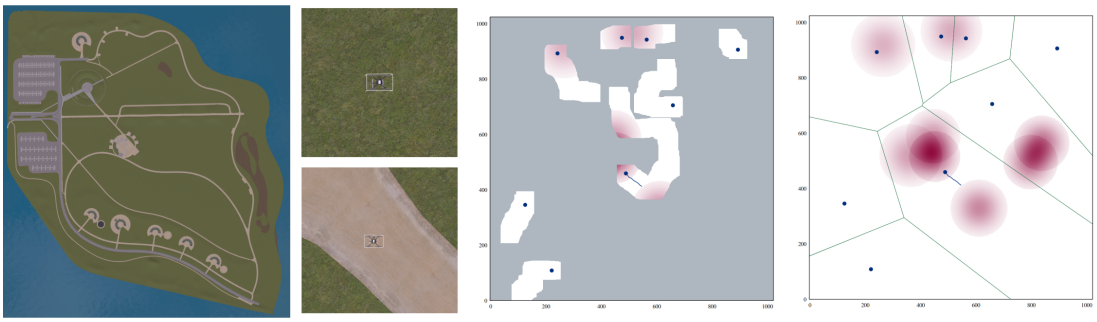


Fig. 18. Gazebo and PX4 based simulation environment using ROS2. The environment is a 250×250 square with 8 robots and 8 simulated features.

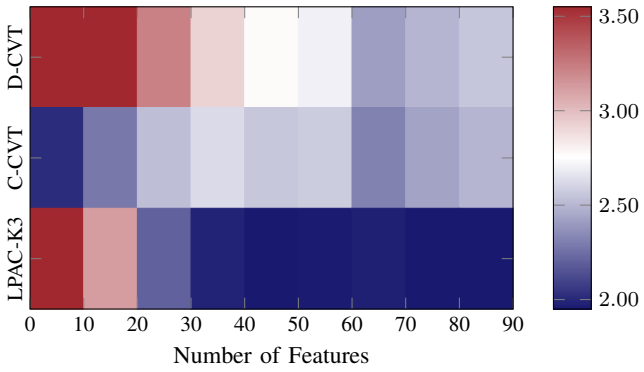


Fig. 19. Performance of coverage algorithms on the real-world traffic light datasets from 50 cities in the US spanning $1024\,\mathrm{m} \times 1024\,\mathrm{m}$: The values in the heatmap are the ratio of the average costs of the coverage algorithms with respect to the clairvoyant algorithm. The 50 environments are organized in the heatmap in buckets of size 10, based on the number of features (traffic signals) in each environment. For each environment, 10 trials with random initial positions of 32 robots are run, and the average cost is computed. The LPAC-K3 model performs poorly on environments with a small number of features, as they are out of distribution for the model. However, the model is able to generalize well to environments with a large number of features, and it outperforms both the decentralized and centralized CVT algorithms, on average, for environments with more than 20 features.

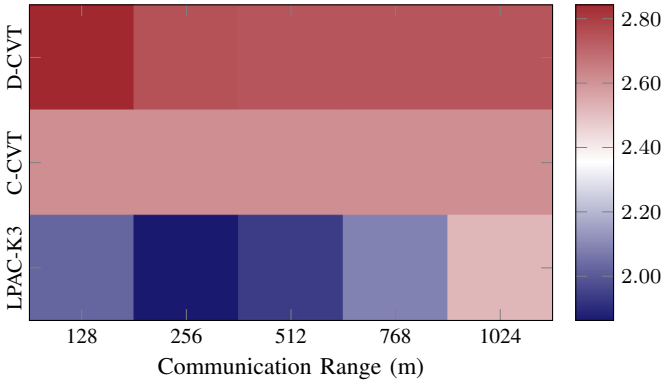


Fig. 20. Performance of the LPAC architecture with different communication ranges: Models with LPAC architecture, with 3 hops of communication, are trained on environments with 32 robots and $1024\,\mathrm{m} \times 1024\,\mathrm{m}$ size with different communication ranges. The models are evaluated on environments with the same size and number of robots but with different communication ranges. Counterintuitively, the LPAC architecture does not perform well when the communication range is closer to the size of the environment. It, however, performs well when the communication range is relatively smaller than the size of the environment, peaking at a communication range of $512\,\mathrm{m}$.

policy is subject to the initial positions of the robots, as the field-of-view is limited. Future work will involve performing large-scale simulations on various environments and more robots, along with real-world experiments, to validate the performance of the LPAC architecture.

H. Communication Analysis

In this section, we analyze the performance of LPAC policies with different communication ranges, and discuss the communication requirements for both centralized and decentralized deployment of such policies.

1) Performance with Different Communication Ranges: We trained models with the LPAC architecture with different communication ranges and compared them with the CVT algorithms. Fig. 20 shows the performance of the controllers as a ratio of the average cost with respect to the clairvoyant algorithm. The same set of environments is used for all controllers and communication ranges. The performance of the centralized CVT algorithm does not change with the communication range, as expected. Whereas the decentralized CVT algorithm improves until a communication range of $512\,\mathrm{m}$ and then stagnates. This is also expected, as the Voronoi cell of a particular robot is generally not affected by the robots that are far away. The performance of the decentralized CVT algorithm is still worse than the centralized one, even with a communication range of $1024\,\mathrm{m}$, as the robots only have knowledge of their own observations of the IDF. In contrast, the centralized algorithm uses the combined observations of all robots.

The LPAC-K3 model outperforms the CVT algorithms for all communication ranges. Interestingly, the performance of the LPAC-K3 model improves until a communication range of $512\,\mathrm{m}$ and then decreases. This is counterintuitive, as one would expect the performance to improve with a larger communication range and the performance to be closer to the clairvoyant algorithm with a communication range of $1024\,\mathrm{m}$. We hypothesize three reasons for this behavior of the LPAC model. First, the neighbor maps in the CNN architecture are of the fixed size of 32×32. For a large communication range, there would be several robots in a cell of the neighbor map due to the low resolution of the map. We take the sum of the values in the cell when we have multiple robots in a cell, leading to a loss of information. Second, in a very similar

way, the GNN architecture uses summation as the aggregation function, which is again a lossy operation. The model may be poorly suited to disambiguating features coming from different robots when the communication range is large. Third, the communication graph becomes almost fully connected as the communication range increases, and GNNs perform better with sparse graphs. Investigating these hypotheses and improving the performance of the LPAC model with a larger communication range is an interesting direction for future work. One possible solution is to select only the closest robot when a cell in the neighbor map has multiple robots and use a more sophisticated aggregation function in the GNN architecture.

Nevertheless, the LPAC model is reliable for communication ranges up to $512\,\mathrm{m}$ and outperforms the CVT algorithms for all communication ranges.

2) Bandwidth Requirements: In the coverage control problem, each robot makes localized observations based on the sensor size and maintains an ego-centric local map over time. In real-world deployment, each robot would need to process raw sensor images and generate importance values. In our simulations, we simulate the generation of importance values to keep the system agnostic to the sensor hardware and the computer vision module. We now analyze the communication bandwidth requirements for the baseline coverage control algorithms and the proposed LPAC-based policies.

D-CVT: The decentralized-CVT algorithm is executed independently on each robot and requires only the positions of the neighboring robots. This means that there is no communication requirement for a robot when there is no neighbor. On the other hand, in the worst-case scenario, from a communication load perspective, a robot would have all other robots as neighbors. Thus, the worst-case requirement is $\mathcal{O}(n)$ messages for a robot. When all robots are within the communication radius of each other, the entire system will be exchanging $\mathcal{O}(n^2)$ messages. The message is only the positions of the robots, which in the simplest case can be represented as a vector of two floating-point numbers. Hence, the communication load for the D-CVT algorithm is very low.

C-CVT: The centralized-CVT requires each robot to send its local maps and positions to a central server. In our implementation, each robot maintains a local map of size 256×256. This requires a message of $2^{16}+2$ floating-point numbers. The central server receives these messages from all n robots. A large-scale system can overload the communication bandwidth due to both large message size and the number of robots.

LPAC: The LPAC policy can be deployed in both centralized and decentralized settings. In the centralized setting, each robot communicates its input to the GNN module, which is the concatenation of the CNN output with the normalized position of the robots. For our LPAC policy, this message is a (2^5+2) size floating-point vector. This is substantially lower than the C-CVT algorithm by a factor of approximately $2^{11} = 2048$. As the results show, the LPAC policy performs significantly better than the C-CVT algorithms, even with a lower communication requirement.

In the decentralized setting, the communication load is strongly tied to the architecture design, i.e., the number of hops, the number of GNN layers, and the latent size of the GNN layers. In our primary LPAC policy, the input to the first GNN layer is a 34-sized vector, and the output of all layers is a $2^8 = 256$-sized vector. As described in Section IV-B and in Equations (10) and (11), the transmitted aggregated message is given by a set of vectors $(\mathbf{y}_i)_{lk}$ for a robot i. Except for $(\mathbf{y}_i)_{00}$, which has a size of 34 and corresponds to the input to the GNN, all other vectors have size $2^8 = 256$. The architecture has $L = 5$ GNN layers with $K = 3$ hops. Thus, the total message size is $14 \times 2^8 + 34 = 3618$ floating-point numbers. This is lower than the C-CVT algorithm by approximately a factor of 18. Similar to the D-CVT algorithm, a robot only communicates directly with its neighbors, and when all robots are neighbors to each other in the worst-case scenario, they make $\mathcal{O}(n)$ communications. In practice, the number of neighbors is relatively low for the coverage control problem. For an environment of size 1024×1024 with 32 robots and 256 communication radius, the average number of neighbors is 5.30 with a standard deviation of 2.35, averaged over simulations of LPAC policy on 100 unique importance density fields and initial robot positions.

We highlight the following features of the GNN-based decentralized LPAC policy: *(i)* The communication is completely decentralized and hence, there is no single point of failure or communication bottleneck. *(ii)* The GNN abstracts the observations and received information into a fixed sized message, which is independent of the number of robots, i.e., it does not increase with the number of neighbors seen by a robot. Hence, the system scales well with the number of robots. *(iii)* The number of message exchanges is always with the neighboring robots. In general, instead of peer-to-peer communication, a broadcasting protocol can also be used to reduce the communication load [23], [22]. *(iv)* The LPAC-policy performs better than the centralized-CVT baseline even though the communication requirement is significantly lower.

VII. Discussion

This article presented a learning-based approach to the decentralized coverage control problem, where a robot swarm is deployed in an environment with an underlying importance density field (IDF), not known *a priori*. The problem requires a robot swarm to be placed optimally in the environment so that robots efficiently cover the IDF with their sensors. The problem is challenging in the decentralized setting as the robots need to decide what to communicate with the neighboring robots and how to incorporate the received information with their own observations. However, in contrast to a centralized system, a decentralized system is robust to individual failures and scales well with large teams in larger environments.

Motivated by the advantages of a decentralized system, the article proposed a learnable Perception-Action-Communication (LPAC) architecture. The perception module uses a convolutional neural network (CNN) to process the local observations of the robot and generates an abstract representation of the local observations. In the communication module, the GNN computes the messages to be sent to other robots and incorporates the received messages with the output of the

perception module. Finally, a shallow multi-layer perceptron (MLP) in the action module computes the velocity controls for each robot. All three modules are executed independently on each robot, and the communication model enables collaboration in the robot swarm with the aid of the GNN.

Models based on the LPAC architecture were trained using imitation learning with a clairvoyant algorithm that uses the centroidal Voronoi tessellation (CVT). We extensively evaluated the LPAC models with the decentralized and the centralized CVT algorithms and showed that the models consistently outperform both CVT algorithms. It is interesting to note that the centralized CVT algorithm knows the positions of all robots to compute the Voronoi partition; it also has the cumulative knowledge of the IDF from all robots to compute the centroids of the Voronoi cell. Yet, the decentralized LPAC models do better, which indicates that they are able to learn features beyond what is used by the CVT-based algorithms.

The evaluations further establish that the models are able to perform well in environments with a larger number of robots and features. The models also transfer to larger environments with bigger robot swarms. The models were robust to noisy estimates of the robot positions, with only a small degradation in performance as the noise levels increased. A real-world dataset was generated from traffic signals as features for the IDF. The models were tested without any further training or fine-tuning. Except for the cases when the number of features was very low, the models outperformed the CVT algorithms, even though these datasets have IDFs that are very different from the training set of randomly and uniformly generated features.

In *conclusion*, the results establish that the LPAC models are transferable to larger environments, scalable to larger robot swarms, and robust to noisy position estimates and varying IDFs for the decentralized coverage control problem. The success of the LPAC architecture indicates that it is a promising approach to solving other decentralized navigation problems. Furthermore, the architecture learns a decentralized policy from a centralized clairvoyant algorithm, thereby alleviating the need for carefully designed decentralized algorithms.

Future work involves testing the models in physical experiments with asynchronous and decentralized communication infrastructure [23]. Additional testing with communication dropouts, latency, and robot failures is essential to characterize the resiliency of the system to failures. Another essential task is to investigate further the poor performance of the models with large communication ranges, which may require minor changes in how aggregation is performed in the GNN architecture. When we do not have any prior knowledge of the IDF, the algorithm can benefit from exploring the environment to discover more features. In contrast, in this article, the robots react to the features they observe without any explicit exploration. However, the exploration is antithetical to the coverage control problem, as the former requires visiting new regions, whereas the latter is focused on providing coverage to the already observed high-importance regions. In future work, we plan to investigate the trade-off between exploration and coverage control. The promising results motivate the theoretical study of the LPAC architecture in relation to transferability, robustness, and scalability. Additionally, the use of the clairvoyant algorithm to generate actions needs further investigation to ensure general applicability to other decentralized navigation problems and to determine the approach's limitations.

The primary motivation for the design of the PAC architecture was to solve decentralized navigation problems with robot swarms where there is a collaborative goal to be achieved. We plan to investigate more navigation problems and assess the generalizability of the LPAC architecture.

Acknowledgments

The map data used to generate the dataset in Section VI-F is copyrighted by OpenStreetMap contributors and is available from https://www.openstreetmap.org.

References

[1] J. Cortés, S. Martìnez, T. Karataş, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Trans. Robot. Autom.*, vol. 20, no. 2, pp. 243–255, 2004.

[2] Cortés, Jorge, Martínez, Sonia, and Bullo, Francesco, "Spatially-distributed coverage optimization and control with limited-range interactions," *ESAIM: Control, Optimisation and Calculus of Variations*, vol. 11, no. 4, pp. 691–719, 2005.

[3] W. Gosrich *et al.*, "Coverage Control in Multi-Robot Systems via Graph Neural Networks," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 8787–8793.

[4] L. Doitsidis *et al.*, "Optimal surveillance coverage for teams of micro aerial vehicles in GPS-denied environments using onboard vision," *Autonomous Robots*, vol. 33, pp. 173–188, 2012.

[5] L. C. Pimenta *et al.*, "Simultaneous coverage and tracking (SCAT) of moving targets with robot networks," in *Algorithmic Foundation of Robotics VIII*. Springer, 2010, pp. 85–99.

[6] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi Tessellations: Applications and Algorithms," *SIAM Review*, vol. 41, no. 4, pp. 637–676, 1999.

[7] K. Tateno, F. Tombari, I. Laina, and N. Navab, "CNN-SLAM: Real-time dense monocular slam with learned depth prediction," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2017, pp. 6243–6252.

[8] A. Oertel, T. Cieslewski, and D. Scaramuzza, "Augmenting Visual Place Recognition With Structural Cues," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 5534–5541, 2020.

[9] N. Yang, L. v. Stumberg, R. Wang, and D. Cremers, "D3VO: Deep Depth, Deep Pose and Deep Uncertainty for Monocular Visual Odometry," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, June 2020.

[10] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[11] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks," in *Proc. Conf. Robot Learning*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, Oct 2020, pp. 671–682.

[12] E. Tolstaya, J. Paulos, V. Kumar, and A. Ribeiro, "Multi-Robot Coverage and Exploration using Spatial Graph Neural Networks," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 8944–8950.

[13] L. Ruiz, F. Gama, and A. Ribeiro, "Graph Neural Networks: Architectures, Stability, and Transferability," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 660–682, 2021.

[14] F. Gama, Q. Li, E. Tolstaya, A. Prorok, and A. Ribeiro, "Synthesizing Decentralized Controllers With Graph Neural Networks and Imitation Learning," *IEEE Trans. Signal Process.*, vol. 70, pp. 1932–1946, 2022.

[15] R. Kortvelesy and A. Prorok, "ModGNN: Expert Policy Approximation in Multi-Agent Systems with a Modular Graph Neural Network Architecture," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 9161–9167.

[16] T.-K. Hu *et al.*, "Scalable Perception-Action-Communication Loops With Convolutional and Graph Neural Networks," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 8, pp. 12–24, 2022.

[17] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph Neural Networks for Decentralized Multi-Robot Path Planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 11 785–11 792.

[18] X. Ji, H. Li, Z. Pan, X. Gao, and C. Tu, "Decentralized, Unlabeled Multi-Agent Navigation in Obstacle-Rich Environments using Graph Neural Networks," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 8936–8943.

[19] L. Zhou *et al.*, "Graph Neural Networks for Decentralized Multi-Robot Target Tracking," in *Proc. IEEE Int. Symp. Safety, Security, and Rescue Robotics*, 2022, pp. 195–202.

[20] E. S. Lee, L. Zhou, A. Ribeiro, and V. Kumar, "Graph neural networks for decentralized multi-agent perimeter defense," *Frontiers in Control Engineering*, vol. 4, p. 1104745, 2023.

[21] M. Tzes, N. Bousias, E. Chatzipantazis, and G. J. Pappas, "Graph neural networks for multi-robot active information acquisition," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 3497–3503.

[22] J. Blumenkamp, S. Morad, J. Gielis, Q. Li, and A. Prorok, "A Framework for Real-World Multi-Robot Systems Running Decentralized GNN-Based Policies," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 8772–8778.

[23] S. Agarwal, A. Ribeiro, and V. Kumar, "A Scalable Multi-Robot Framework for Decentralized and Asynchronous Perception-Action-Communication Loops," *arXiv preprint arXiv:2309.10164*, 2025.

[24] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," *Proc. Int. Conf. Neural Inf. Process. Syst.*, vol. 29, 2016.

[25] A. Serra-Gómez, B. Brito, H. Zhu, J. J. Chung, and J. Alonso-Mora, "With Whom to Communicate: Learning Efficient Communication for Multi-Robot Collision Avoidance," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 11 770–11 776.

[26] S. Meng and Z. Kan, "Deep Reinforcement Learning-Based Effective Coverage Control With Connectivity Constraints," *IEEE Control Syst. Lett.*, vol. 6, pp. 283–288, 2022.

[27] B. Wang, "Coverage Problems in Sensor Networks: A Survey," *ACM Comput. Surv.*, vol. 43, no. 4, oct 2011.

[28] A. Suzuki and Z. Drezner, "The p-center location problem in an area," *Location Science*, vol. 4, no. 1, pp. 69–82, 1996.

[29] H. Edelsbrunner and R. Seidel, "Voronoi Diagrams and Arrangements," *Discrete & Computational Geometry*, no. 1, pp. 25–44, 1986.

[30] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[31] F. Pratissoli, B. Capelli, and L. Sabattini, "On Coverage Control for Limited Range Multi-Robot Systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 9957–9963.

[32] A. Kwok and S. Martinez, "A distributed deterministic annealing algorithm for limited-range sensor coverage," *IEEE Trans. Control Syst. Technol.*, vol. 19, no. 4, pp. 792–804, 2010.

[33] H. F. Parapari, F. Abdollahi, and M. B. Menhaj, "Distributed coverage control for mobile robots with limited-range sector sensors," in *Proc. IEEE Int. Conf. on Advanced Intelligent Mechatronics (AIM)*, 2016, pp. 1079–1084.

[34] J. Zhong, H. Cheng, L. He, and F. Ouyang, "Decentralized Full Coverage of Unknown Areas by Multiple Robots With Limited Visibility Sensing," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 338–345, 2019.

[35] M. Schwager, J.-J. Slotine, and D. Rus, "Consensus learning for distributed coverage control," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2008, pp. 1042–1048.

[36] L. Wei, A. McDonald, and V. Srivastava, "Multi-Robot Gaussian Process Estimation and Coverage: Deterministic Sequencing Algorithm and Regret Analysis," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 9080–9085.

[37] M. Santos, U. Madhushani, A. Benevento, and N. E. Leonard, "Multi-robot Learning and Coverage of Unknown Spatial Fields," in *Proc. Int. Symp. Multi-Robot and Multi-Agent Sys.*, 2021, pp. 137–145.

[38] M. Schwager, F. Bullo, D. Skelly, and D. Rus, "A ladybug exploration strategy for distributed adaptive coverage control," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2008, pp. 2346–2353.

[39] M. Mirzaei, F. Sharifi, B. W. Gordon, C. A. Rabbath, and Y. Zhang, "Cooperative multi-vehicle search and coverage problem in uncertain environments," in *Proc. IEEE Conf. Decision and Control*, 2011, pp. 4140–4145.

[40] S. G. Lee, Y. Diaz-Mercado, and M. Egerstedt, "Multirobot Control Using Time-Varying Density Functions," *IEEE Trans. Robot.*, vol. 31, no. 2, pp. 489–493, 2015.

[41] M. Santos, S. Mayya, G. Notomista, and M. Egerstedt, "Decentralized Minimum-Energy Coverage Control for Time-Varying Density Functions," in *Proc. Int. Symp. Multi-Robot and Multi-Agent Sys.*, 2019, pp. 155–161.

[42] A. Vaswani *et al.*, "Attention is All You Need," in *Proc. Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 6000–6010.

[43] M. S. Couceiro, D. Portugal, J. F. Ferreira, and R. P. Rocha, "SEMFIRE: Towards a new generation of forestry maintenance multi-robot systems," in *Proc. IEEE/SICE Int. Symp. on Syst. Integration (SII)*, 2019, pp. 270–276.

[44] W. Luo, C. Nam, G. Kantor, and K. Sycara, "Distributed environmental modeling and adaptive sampling for multi-robot sensor coverage," in *Proc. Int. Conf. Auton. Agents and MultiAgent Systems (AAMAS)*, 2019, pp. 1488–1496.

[45] W. Luo and K. Sycara, "Adaptive sampling and online learning in multi-robot sensor coverage with mixture of gaussian processes," in *Proc. IEEE Int. Conf. Robot. Automat.* IEEE, 2018, pp. 6359–6364.

[46] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Berlin: Springer-Verlag, 2008.

[47] L. C. A. Pimenta, V. Kumar, R. C. Mesquita, and G. A. S. Pereira, "Sensing and coverage for a network of heterogeneous robots," in *Proc. IEEE Conf. Decision and Control*, 2008, pp. 3947–3952.

[48] H. S. Witsenhausen, "A Counterexample in Stochastic Optimum Control," *SIAM Journal on Control*, vol. 6, no. 1, pp. 131–147, 1968.

[49] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proc. Int. Conf. Machine Learning*. JMLR.org, 2015, pp. 448–456.

[50] M. Karavelas, "2D Voronoi Diagram Adaptor," in *CGAL User and Reference Manual*, 5th ed. CGAL Editorial Board, 2023. [Online]. Available: https://doc.cgal.org/5.6/Manual/packages.html#PkgVoronoiDiagram2

[51] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proc. Int. Conf. Learn. Representations*, San Diega, CA, USA, 2015.

[52] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. Int. Conf. Neural Inf. Process. Syst.* Curran Associates, Inc., 2019, pp. 8024–8035.

[53] M. Fey and J. E. Lenssen. (2019, May) Fast Graph Representation Learning with PyTorch Geometric.