LEMURS: Learning Distributed Multi-Robot Interactions

Eduardo Sebastián Thai Duong Nikolay Atanasov Eduardo Montijano Carlos Sagüés

Abstract—This paper presents LEMURS, an algorithm for learning scalable multi-robot control policies from cooperative task demonstrations. We propose a port-Hamiltonian description of the multi-robot system to exploit universal physical constraints in interconnected systems and achieve closed-loop stability. We represent a multi-robot control policy using an architecture that combines self-attention mechanisms and neural ordinary differential equations. The former handles time-varying communication in the robot team, while the latter respects the continuous-time robot dynamics. Our representation is distributed by construction, enabling the learned control policies to be deployed in robot teams of different sizes. We demonstrate that LEMURS can learn interactions and cooperative behaviors from demonstrations of multi-agent navigation and flocking tasks.

I. INTRODUCTION

Multi-robot systems promise improved efficiency and reliability compared to a single robot in many applications, including exploration and mapping [1], [2], agriculture and herding [3]–[6], and search and rescue [7]. However, designing multi-robot control policies that achieve cooperative behaviors may be challenging. First, domain expertise may be required to specify the objective and constraints for a desired task in mathematical terms. Second, scaling the control policy to large teams may be computationally infeasible due to the increase of the joint state and control spaces. The first challenge motivates the use of machine learning techniques to learn reward functions or control policies from demonstration [8]–[23]. The second challenge motivates imposing a sparse structure in the control policy that respects the communication topology of the robot team and allows the complexity to scale with the number of neighbors [10]–[15], [24]–[26]. In this work, we develop LEMURS (LEarning distributed MUlti-Robot interactionS), a learning approach for distributed control synthesis from cooperative task demonstrations that generalizes to different tasks, scales favorably with the number of robots, and handles time-varying robot dynamics.

Recent works focus on learning control policies for optimal control or reinforcement learning problems [9], [18], [19]. Given a cost function, a recurrent neural network [22], or graph convolutions and multi-layer perceptrons [23] have been used to learn centralized control policies. Without a cost function, inverse reinforcement learning [27] may be used to learn centralized [20], [21] or distributed [16] policies from task demonstrations. While black-box neural networks are widely used for learning control policies, they do not encode energy conservation and kinematic constraints satisfied by physical robot systems, and failing to infer them from data may result in unstable behaviors. A key contribution of our work is to represent the robot team as a port-Hamiltonian system [28] and learn a distributed control policy from demonstration by modeling robot interactions as energy exchanges. The use of Hamiltonian mechanics has been explored for centralized control policies or fixed-time known topologies [15], [29], in which scalability is achieved in the absence of communication [17]. Meanwhile, our work achieves scalability with a time-varying topology by modeling robot interactions using self-attention techniques [30].

Learning and execution of control policies for multi-robot systems should scale favorably with an increasing numbers of robots. Learning a joint value or policy function is challenging due to the exponential growth of the state and action space [24]. Successful methods for multi-agent reinforcement learning factorize value functions according to the k-hop neighborhoods [24], [26] or using attention mechanisms [31]. Graph neural networks have been utilized as a scalable and communication-aware policy representation in coverage, exploration, and flocking problems [10]–[14]. Recently, Li et al. [32] combine graph-neural networks with self-attention to solve decentralized multi-robot path planning problems. Many of these techniques assume discrete robot dynamics, fixed or known communication topology, or prior knowledge about the task. In contrast, our approach learns from demonstrated robot trajectories with an unknown task objective and handles time-varying communication and team sizes. In order to handle continuous-time dynamics, we use neural ordinary differential equation (ODE) networks [33]. Closely related, Jiahao et al. [8] develop a neural ODE network that learns distributed controllers but enforces collision avoidance using an explicit potential field and assumes a fixed maximum number of neighbors. By using a port-Hamiltonian formulation and self-attention mechanism, we handle time-varying neighbors, do not constrain the size of the neighborhoods, and learn constraints such as collision avoidance from data.

In summary, we develop LEMURS, a novel algorithm for learning scalable multi-robot control policies from demonstration. Our first contribution is the use of port-Hamiltonian...
dynamics to restrict the family of learned policies to those that are stable and distributed. Our second contribution is a novel learning architecture that integrates concepts of self-attention and neural ODEs to handle continuous-time dynamics, time-varying communication, and large robot teams.

II. PROBLEM STATEMENT

Consider a team of robots, indexed by $\mathcal{V} = \{1, \ldots, n\}$. Assume that the dynamics of each robot $i \in \mathcal{V}$ are known:

$$\dot{x}_i(t) = f_i(x_i(t), u_i(t)), \quad (1)$$

where $x_i(t) \in \mathbb{R}^{n_x}$ and $u_i(t) \in \mathbb{R}^{n_u}$ denote the state and control input of the robot at time $t \geq 0$. The robots interact in a distributed manner, described by a time-varying undirected graph $\mathcal{G}(t) = (\mathcal{V}, \mathcal{E}(t))$, where $\mathcal{E}(t) \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. An edge $(i, j) \in \mathcal{E}(t)$ exists when robots $i$ and $j$ interact at time $t$. Robot $i$ can always interact with itself, i.e., $(i, i) \in \mathcal{E}(t)$ for all $i$. Let $A(t) \in \{0, 1\}^{n \times n}$ be the weighted adjacency matrix associated to $\mathcal{G}(t)$, such that $[A(t)]_{ij} \neq 0$ if and only if $(i, j) \in \mathcal{E}(t)$, and 0 otherwise. The set of $k$-hop neighbors of robot $i$ at $t$ is $\mathcal{N}_k^i(t) = \{j \in \mathcal{V} \mid [A^k(t)]_{ij} \neq 0\}$. Each robot executes an unknown control policy:

$$u_i(t) = \pi_\theta \left(x_{\mathcal{N}_k^i(t)} \right), \quad (2)$$

where $x_{\mathcal{N}_k^i(t)} = \{x_j(t) \mid j \in \mathcal{N}_k^i(t)\}$ and $\theta$ is the control policy parameters. Our objective is to use task demonstrations to learn $\theta$, such that the multi-robot team, following the learned control policy, replicates the task.

Example 1. Consider a flocking task [34] in which a robot team must achieve a formation with aligned velocities, while avoiding collisions. The robots follow double integrator dynamics $p_i(t) = v_i(t)$ and $v_i(t) = u_i(t)$, where $p_i(t) \in \mathbb{R}^m$, $v_i(t) \in \mathbb{R}^m$, and $u_i(t) \in \mathbb{R}^m$ are the position, velocity, and input of robot $i$. A distributed control policy that achieves flocking was developed by Olfati-Saber [34]:

$$u_i(t) = -c_1 p_i(t) - c_2 v_i(t) + \sum_{j \in \mathcal{N}_k^i(t)} \phi(||\Delta p_{ij}(t)||_\sigma) n_{ij}(t) + \sum_{j \in \mathcal{N}_k^i(t)} \rho(||\Delta v_{ij}(t)||_\sigma) (v_j(t) - v_i(t)) \quad (3)$$

where $||\cdot||_\sigma$ is the $\sigma$-norm of a vector and $\Delta p_{ij}(t) = p_j(t) - p_i(t)$. The first and second terms are a proportional controller with gains $c_1, c_2 > 0$ that prevents the formation of sub-flocks, where we have assumed $p_i(t) = v_i(t) = 0$ for each robot flock configuration. The third term avoids robot collisions and induces the desired lattice formation, where $\phi(\cdot)$ is a potential function whose minima are located at the desired inter-agent distances, and $n_{ij}(t)$ is a vector that points in the repulsion/coalition direction. The last term achieves velocity consensus using a distance scaling function $\rho(\cdot)$ that models the robot communication. Further details can be found in [34]. This paper aims to learn policies like (3) from demonstrations.

We assume that state trajectories from successful task executions are available as training data. Let $x(t) = [x_1(t), \ldots, x_n(t)]^T$ and $u(t) = [u_1(t), \ldots, u_n(t)]^T$ denote the joint state and control of the robot team. Given an initial state $x(0) = x_0$, let $x_{0:K} := [x(0), \ldots, x(rT), \ldots, x(KT)]$ and $x_{0:K} := [x(0), \ldots, x(rT), \ldots, x(KT)]$ be demonstrated and learned policy’s trajectories, respectively, with $K$ denoting the number of discrete samples $r$ along the trajectories with sampling interval $T$. Let $\mathcal{D} := \{x_{0:K}\}_{l=0}^L$ denote a dataset of $L > 0$ demonstrated trajectories. Let $\mathcal{D} := \{x_{0:K}\}_{l=0}^L$ be the generated trajectories under policy $\pi_\theta$. We aim to learn a control policy that minimizes the distance between the demonstrated and generated trajectories:

$$\mathcal{L}(\mathcal{D}, \hat{\mathcal{D}}) = \frac{1}{KL} \sum_{l=0}^L \sum_{r=0}^K ||x^r_l(t) - \hat{x}^r_l(t)||_2^2. \quad (4)$$

Formally, the problem we consider is:

$$\min_{\theta} \mathcal{L}(\mathcal{D}, \mathcal{D}) \quad (5a)$$

s.t. $\dot{x}_i^r(t) = f_i(x_i^r(t), u_i^r(t)), \ x_i^r(0) = x_i^r(0), \ \forall i, l, r. \quad (5b)$

$$u_i^r(t) = \pi_\theta(x_{\mathcal{N}_k^i(t)}^r), \ \forall i, l, t. \quad (5c)$$

As specified in the formulation above, the learned control policy should also handle time-varying communication and should be adaptable to changes in the total number of robots or the number of neighbors for each robot.

III. LEMURS

In this section, we present a port-Hamiltonian formulation of the multi-robot dynamics and an energy-based distributed control design that can shape the interactions and Hamiltonian of the closed-loop system (Sec. III-A). Given task demonstrations, we employ self-attention and neural ordinary differential equations to learn the interactions and energy parameters of the control policy that minimize the distance between the demonstrated and generated trajectories (Sec. III-B). To simplify the notation, we omit the time dependence of the states $x$ and controls $u$ in the remainder of the paper.

A. Port-Hamiltonian Formulation of Multi-Robot Dynamics

Port-Hamiltonian mechanics are a general yet interpretable modeling approach for learning and control. On the one hand, many physical networked systems can be described as a port-Hamiltonian system [15] using the same formulation and with a modular and distributed interpretation. Meanwhile, the port-Hamiltonian description allows to derive general energy-based controllers with closed-loop stability guarantees. Since robots are physical systems that satisfy Hamiltonian mechanics, we model each robot’s dynamics in (1) as a port-Hamiltonian system [28]:

$$\dot{x}_i = \left(J_i^{(s)}(x_i) - R_i^{(s)}(x_i)\right) \frac{\partial H_i^{(s)}(x_i)}{\partial x_i} + F_i^{(s)}(x_i)u_i, \quad (6)$$

where the skew-symmetric interconnection matrix $J_i^{(s)}(x_i)$ represents energy exchange within a robot, the positive-semidefinite dissipation matrix $R_i^{(s)}(x_i)$ represents energy exchange between agents, and the vector $F_i^{(s)}(x_i)$ is the control input.
dissipation, the Hamiltonian $H_s^{(i)}(x_i)$ represents the total energy, and the matrix $F_s^{(i)}(x_i)$ is the input gain. Then, the multi-robot system with joint state $x$ also follows port-Hamiltonian dynamics:

$$\dot{x} = (J_s(x) - R_s(x)) \frac{\partial H_s(x)}{\partial x} + F_s(x)u,$$

where $H_s(x) = \sum_{i=1}^n H_s^{(i)}(x_i)$ and

$$J_s(x) = \text{diag}\left( J_s^{(1)}(x_1), \ldots, J_s^{(n)}(x_n) \right),$$

$$R_s(x) = \text{diag}\left( R_s^{(1)}(x_1), \ldots, R_s^{(n)}(x_n) \right),$$

$$F_s(x) = \text{diag}\left( F_s^{(1)}(x_1), \ldots, F_s^{(n)}(x_n) \right).$$

Without control, the trajectories of the open-loop system in (7) would not match the demonstrations in $\mathcal{D}$. The dynamics need to be controlled by the policy in (2) in order to generate desired trajectories. We employ an interconnection and damping assignment passivity-based control (IDA-PBC) approach [28], which injects additional energy to the system through the control input $u$ to achieve some closed-loop dynamics that replicate the demonstrated task:

$$\dot{x} = (J_\theta(x) - R_\theta(x)) \frac{\partial H_\theta(x)}{\partial x},$$

with Hamiltonian $H_\theta(x)$, skew-symmetric interconnection $J_\theta(x)$, and positive semidefinite dissipation $R_\theta(x)$. By matching the terms in (7) and (9), one obtains the policy:

$$u = F_s^\dagger(x) \left( (J_\theta(x) - R_\theta(x)) \frac{\partial H_\theta(x)}{\partial x} - (J_s(x) - R_s(x)) \frac{\partial H_s(x)}{\partial x} \right).$$

where $F_s^\dagger(x) = (F_s^\top(x)F_s(x))^{-1}F_s^\top(x)$ is the pseudo-inverse of $F_s(x)$. If the robots are fully-actuated, i.e., $F_s(x)$ is full-rank, the matching condition on the pseudo-inverse is always satisfied, achieving the desired closed-loop dynamics. For underactuated systems, satisfaction of the matching condition may not always be possible [35]. Being able to achieve zero error $L(\mathcal{D}, \mathcal{D})$ is, hence, related to whether the demonstrated trajectories $\mathcal{D}$ are realizable by the class of control policies in (10). Even if the trajectories in $\mathcal{D}$ are not realizable, the policy parameters $\theta$ may still be optimized to achieve a behavior as similar as possible.

Let $[J_\theta(x)]_{ij}$ and $[R_\theta(x)]_{ij}$ denote the $n_x \times n_x$ blocks with index $(i,j)$, representing the energy exchange between robot $i$ and $j$ and the energy dissipation of robot $i$ caused by robot $j$, respectively. Since the input gain $F_s(x)$ in (8) is block-diagonal, the individual control policy of robot $i$ is:

$$u_i = F_s^{(i)}(x_i) \left( \sum_{j \in V} ([J_\theta(x)]_{ij} - [R_\theta(x)]_{ij}) \frac{\partial H_\theta(x)}{\partial x_j} - \left( J_s^{(i)}(x_i) - R_s^{(i)}(x_i) \right) \frac{\partial H_s^{(i)}(x)}{\partial x_i} \right).$$

The individual control policies in (11) do not necessarily respect the hops in the communication network as desired in (2) because this depends on the structure of $J_\theta(x)$, $R_\theta(x)$, and $H_\theta(x)$. In Sec. III-B, we impose conditions on these terms to ensure that they respect the communication topology and are skew-symmetric, and positive semidefinite, respectively, as required for a valid port-Hamiltonian system.

**Example 2.** By substituting the control policy (3) in the double integrator dynamics of the robots, the closed-loop dynamics for the flocking problem in Example 1 are:

$$\dot{p} = v, \quad \dot{v} = -\frac{\partial U_\theta(p)}{\partial p} - D_\theta(p)v,$$

with $\frac{\partial U_\theta(p)}{\partial p} = c_1p_i + \sum_{j \in N_k^i} \phi(||p_j - p_i||_\sigma)n_{ij}$, (12)

$$[D_\theta(p)]_{ij} = (c_2[I_n]_{ij} + \rho(||p_j - p_i||_\sigma/\tau)) I_m.$$

In port-Hamiltonian terms, $H_s(x) = \frac{1}{2}v^\top v$, $R_s = 0$, $F_s(x) = [0, I_m]^\top$, $J_s(x) = J_\theta(x) = \begin{pmatrix} 0 & I_n \\ -I_n & 0 \end{pmatrix}$, $R_\theta(x) = \begin{pmatrix} 0 & 0 \\ 0 & D_\theta(p) \end{pmatrix}$, and $H_\theta(x) = U_\theta(p) + \frac{1}{2}v^\top v$.

**B. Learning Distributed Multi-Robot Interactions**

The analytical design of scalable cooperative control policies like the flocking controller of Example 1 is challenging when the complexity of the task increases. Instead, we seek to learn control policies that scale with the number of robots, handle time-varying communications and guarantee the port-Hamiltonian constraints. To do so, we first derive conditions on $J_\theta(x)$, $R_\theta(x)$ and $H_\theta(x)$. Then, we develop a novel architecture based on self-attention and neural ordinary differential equations to ensure that the learned control policies guarantee these conditions.

We first impose $J_\theta(x)$ and $R_\theta(x)$ to be block-sparse,

$$[J_\theta(x)]_{ij} = [R_\theta(x)]_{ij} = 0, \quad \forall j \notin N_k^i. \quad (13)$$

This is to satisfy the topology constraints of the multi-robot team. Moreover, we require that the desired Hamiltonian factorizes over $k$-hop neighborhoods:

$$H_\theta(x) = \sum_{i=0}^n H_\theta^{(i)}(x_{\mathcal{N}_k^i}). \quad (14)$$

The factorization in (14) ensures that each robot $i$ can calculate $\partial H_\theta(x)/\partial x_i = \sum_{j \in N_k^i} \partial H_\theta^{(j)}(x_{\mathcal{N}_k^j})/\partial x_i$ by gathering $\partial H_\theta^{(j)}(x_{\mathcal{N}_k^j})/\partial x_i$ from its $k$-hop neighbors $j$.

Then, the control policy $\pi_\theta$ of robot $i$ becomes:

$$u_i = F_s^{(i)}(x_i) \left( \sum_{j \in N_k^i} ([J_\theta(x)]_{ij} - [R_\theta(x)]_{ij}) \frac{\partial H_\theta(x)}{\partial x_j} - \left( J_s^{(i)}(x_i) - R_s^{(i)}(x_i) \right) \frac{\partial H_s^{(i)}(x)}{\partial x_i} \right).$$

Imposing the requirements in (13)-(14) is a first step towards making the control policy in (15) distributed. Note that the terms $[J_\theta(x)]_{ij}$ and $[R_\theta(x)]_{ij}$ might still depend on the joint state $x$ even if $j \in N_k^i$. We discuss how to remove this dependence next and achieve a similar factorization as (14).
Fig. 1: Architecture of LEMURS: robot $i$ receives information from its neighbors. Then, the self-attention module obtains the self-attention equation in (17) and calculates $H(\theta)$, we enforce the skew-symmetry of $J(\theta)$ by:

$$H(\theta) = (\text{vec}(X_i^0))^\top M(\theta) (\text{vec}(X_i^0)) + U(\theta) (X_i^0), \quad (22)$$

where the first term $(\text{vec}(X_i^0))^\top M(\theta) (\text{vec}(X_i^0))$ is a kinetic-like energy function with $M(\theta) (X_i^0) = \text{diag}(1, Z_i^0)$, and the second term $U(\theta) (X_i^0)$ is a potential energy function with $U(\theta) (X_i^0) = 1^\top Z_i^0 1$. The encodings $Z_i^0$ and $Z_i^1$ are calculated using the same steps (17)-(19), with parameters $A_{QR, w}, A_{K, w}, A_{V, w}$ and $A_{QR, w}, A_{K, w}, A_{V, w}$ and $A_{QR, w}$, respectively. With $H(\theta) = 0$ we obtain $\frac{\partial H(\theta)}{\partial x_i} \forall j \in N_k^i$ and compute $\frac{\partial H(\theta)}{\partial x_i} = \sum_{j \in N_k^i} \frac{\partial H(\theta)}{\partial x_j}$.

2) Learning distributed control policies using neural ODE networks: Let $\mathbf{SA}(X_0^0, \theta)$ be the operations (17)-(22) with

$$\theta = \{(A_{QR, k}, A_{K, k}, A_{V, k}, A_{QR, k}, w, A_{K, k}, w, A_{V, k}, w, w=1) : k = (\mathbf{R}, \mathbf{J}, \mathbf{M}, \mathbf{U})\}.\nonumber$$

To address Problem (5), we use a neural ODE network [33] whose structure respects the continuous-time dynamics in (6). To calculate the loss $\mathcal{L}(D, D)$ in (4), for each trajectory $l$ of robot $i$, $\{x_i^l(r^l)\}_r$ in the data, we solve an ODE:

$$\dot{x}_i^l = f_i(x_i^l, \theta), \quad x_i^l(0) = \hat{x}_i^l(0), \quad (23)$$

using an ODE solver to obtain a predicted state $x_i^l(r^l)$ for $i \in \mathcal{V}, l = 1, \ldots, L$:

$$x_i^l = \text{ODESolver}(x_i^l(0), f_i, r^l; \theta). \quad (24)$$

The parameters $\theta$ are updated using gradient descent by back-propagating the loss through the neural ODE solver using adjoint states $y_i = \frac{d}{dr} \frac{\partial \mathcal{L}}{\partial \theta}$. We form an augmented state $z_i = (x_i, y_i, \frac{\partial \mathcal{L}}{\partial \theta})$ that satisfies $\dot{z}_i = f_{z} = (f_i, y_i, \frac{\partial \mathcal{L}}{\partial \theta})$. The gradients $\frac{\partial \mathcal{L}}{\partial \theta}$ are obtained by solving a reverse-time ODE starting from $z_i(r^l) = z_i(r^l)$:

$$(x_i(0), y_i(0), \partial \mathcal{L}/\partial \theta) = \text{ODESolver}(z_i(r^l), f_{z}, r^l). \quad (25)$$

We refer the reader to [33] for more details.

3) Deploying LEMURS: To deploy the control policy (15), we design a message $m_i(t)$, encoding information that robot $i$ needs from robot $j$ at time $t$ to calculate $[\mathbf{g}_{ij}], [\mathbf{R}_{ij}],$ and $H_{ij}(\theta)$.

Each robot $i$ receives a message $m_i = [m_i^{(1)}, m_i^{(2)}, m_i^{(3)}]$ in 3 communication rounds: 1) robot $i$ receives $m_i^{(1)} = x_j \forall j \in N_i$ and calculates $Z_{ij}^{(1)}, Z_{ij}^{(1)}, H_{ij}^{(1)},$ and $\partial H_{ij}^{(1)}/\partial x_j$; 2) robot $i$ receives $m_i^{(2)} = \partial H_{ij}^{(1)}/\partial x_i, Z_{ij}^{(2)}, Z_{ij}^{(2)} \forall j \in N_i$, and calculates $\partial H_{ij}^{(1)}/\partial x_i, [\mathbf{g}_{ij}], [\mathbf{R}_{ij}]$; and 3) each robot
i receives \( m_{ij}^{(3)} = \frac{\partial H_{\theta}}{\partial x_j} \forall j \in N_k^i \) and calculates the control input \( u_i \). We assume negligible delays between communication rounds. If the delay is large, Wang et al. [26] suggest to learn a function that predicts quantities such as \( \frac{\partial H_{\theta}(x)}{\partial x_j}, Z_{ji}^l, Z_{ji}^r \), leading to one communication round. We leave this for future work. If the Hamiltonian changes slowly over sampling interval \( T \), at time \( t = rT \), robot \( i \) can use its previous neighbor states \( x_j((r-1)T) \) to approximate \( m_{ij}^{(3)}(rT) \) and \( m_{ij}^{(5)}(rT) \).

**Example 3.** In the flocking of Examples 1-2, LEMURS can be directly applied to learn \( D_{\theta}(p) \) and \( U_{\theta}(p) \). Another option is to learn \( J_{\theta}(x), R_{\theta}(x) \) and \( H_{\theta}(x) \), obtaining extra degrees of freedom for the control policy.

**IV. RESULTS**

In this section we evaluate LEMURS in three multi-robot tasks with simulated point robots, illustrated in Fig. 2:

1) **Fixed swapping** [15]: Robots are initialized in two columns and navigate to the diagonally opposite position in the other column while avoiding collisions (Fig. 2a). The communication graph is a fixed ring such that robot \( i \) communicates with robots \( (i \pm 1) \mod n \). We use the same parameters as [15]. We generate demonstrations from the following expert controller:

\[
  u_i = -c_1 p_i - c_2 v_i - \sum_{j \in N_k^i} \frac{p_i - p_j}{\sqrt{1 + \sigma ||p_i - p_j||^2}},
\]

with \( c_1 = 0.8, c_2 = 1.0, \sigma = 0.1 \).

2) **Time-varying swapping**: We consider the fixed swapping task but with a time-varying communication graph (Fig. 2c), where \( A(t) \) is such that \( A(t)_{ij} = \text{sigmoid}(\lambda ||p_i(t) - p_j(t)|| - l/2) \) if \( ||p_i(t) - p_j(t)|| < l \) and 0 otherwise, with \( l = 2.4m \) and \( \lambda = 2.0 \). We use the controller in (26) with time-varying neighbors to generate demonstrations.

3) **Flocking**: We consider the flocking task described in Examples 1-3 with parameters from [34] (Fig. 2e). We use the controller (3) to generate demonstrations.

The training and evaluation datasets for each task have \( L = 400 \) trajectories of \( K = 250 \) samples with sampling interval \( T = 0.04s \). The number of demonstrating robots is \( n = 4 \), and the trajectories are split in sub-trajectories of 5 samples for training. We train for 10000 epochs with learning rate 0.001, and new batches of 200 samples every 100 epochs. The ODEsolver is the Euler numerical method [36]. We consider \( k = 1 \) as the number of hops.

The learned control policies are stable and scalable for all the tasks, converging to the desired goal with a larger number of robots as seen in Figs. 2. We plot trajectories from the expert control policy (left) and learned control policies for 12 robots (right), three times larger than the team size in training. Similar results with up to 64 robots can be found on our website\(^1\). For all three tasks, LEMURS achieves similar performance compared to analytical policies, which were used to generate training trajectories. LEMURS successfully captures behaviors that are not encoded a priori in the architecture of the networks nor in the cost function, such as the collision avoidance or the flock formation in flocking. Collision avoidance among the point robots is verified in all tasks by checking the distance between each pair of robots. In the swapping problems, as the training dataset is formed by sub-trajectories of 5 samples, which resemble a straight line in general, LEMURS infers that the motion to the goals should be a straight line as well. On the other hand, the minimum distance among robots is 0.005m, avoiding collisions even in the center of the stage. In flocking task, since we train LEMURS for flocking with only 4 robots in Fig. 2e, LEMURS infers that it is desired to have 4 groups of robots with equal distances between the groups (Fig. 2f), prioritizing formation to safety. In this sense, evaluation with 4 robots yields to a minimum distance among robots of 0.05m, while with 12 robots the minimum distance among robots is 0.001m. To improve generalization, we suggest increasing the number of robots during training, but we leave this for future work.

---

\(^1\)https://eduardosebastianrodriguez.github.io/LEMURS/

---

**Fig. 2:** Demonstration of expert and learned control policies: (left) expert control policies (ECP) for training with 4 robots, (right) learned control policies (LCP) with 12 robots for 3 tasks.

We compare LEMURS with three other learning methods: 1) Multi-Layer Perceptron (MLP), inspired by [15]; 2) Graph Neural Network (GNN) from [10], [13]; and 3) Self-Attention based Graph Neural Network (GNNSA) [32], which uses graph neural networks preceded by a self-attention layer to model communication channels. These
learning models substitute the self-attention layers in our Hamiltonian-based neural ODE networks. We keep the port-Hamiltonian neural ODE architecture for a fair comparison with the other discrete-time and/or black-box policies, leaving the complete adaptation of the other papers to our setting for future work. The size of the layers/likelihoods in the MLP, GNN and GNNSA depends on the number of robots, so scalability is not directly achievable unlike in our approach. LEMURS has 2208 parameters while MLP, GNN and GNNSA have 1 layer/filter with 4448, 4448 and 4672 parameters, respectively. The details are in the Appendix I.

is slightly better than that of GNNSA, and two orders of magnitudes better than that of MLP and GNN.

We also validate scalability. The policies are simulated over a time horizon $KT = 10s$. We train LEMURS with datasets of $n = \{4, 8, 16\}$, for 5 runs using randomized seeds, and test the learned control policies with $n = \{4, 8, 16, 32, 64\}$. The mean and standard deviation of the test loss (Eq. (4)) is normalized by $n$ and plotted in Fig. 3 (right). LEMURS obtains similar test loss with respect to the number of training robots. In the case of fixed swapping, increasing the number of robots in training improves the controller performance since the larger number of robots is, the more data is available to learn about a fixed communication topology. For the time-varying swapping task, a small number of robots in training performs slightly better, potentially because the time-varying topology is more complex with more robots. Meanwhile, for flocking task, increasing the number of training robots slightly improves the performance, even though the topology is also time-varying. This is because the robots form a flocking formation in the training trajectories, leading to a fixed topology in a large portion of the dataset, similar to fixed swapping.

V. CONCLUSIONS

This work presented LEMURS, an algorithm that learns robot interactions from trajectory demonstrations using self-attention and Hamiltonian-based neural ODE networks. LEMURS advances the state of the art by learning control policies that generalize to increasing numbers of robots and time-varying communications. Our evaluation shows that LEMURS learns behaviors such as collision avoidance and flocking formation from state-only trajectories of few robots, and successfully replicates the tasks in larger robot teams.

APPENDIX I

NETWORK AND EXPERIMENT PARAMETERS

The SA architecture is parameterized as follows:

- $[R_{ij}]$: $W = 3$, $h_w = [4, 8, 8]$, $r_w = [8, 8, 8]$, $d_w = [8, 8, 16]$: functions $\beta = \text{sigmoid}$, $\gamma = \alpha = \text{swish}$ [37].
- $[J_{ij}]$: $W = 3$, $h_w = [4, 8, 8]$, $r_w = [8, 8, 8]$, $d_w = [8, 8, 16]$: functions $\beta = \text{sigmoid}$, $\gamma = \alpha = \text{swish}$ [37]; and $[J_{ij}] = 0 \forall i \neq j$.
- $H^\theta$: $W = 3$ layers, $h_w = [6, 8, 8]$, $r_w = [8, 8, 8]$, $d_w = [8, 8, 25]$: functions $\beta = \text{sigmoid}$, $\gamma = \alpha = \text{swish}$ [37].

The network input $X^0_i$ is an offset version of (16) as follows:

- $[R_{ij}]$ and $[J_{ij}]$: $X^0_i = \left[\Delta x_{ii}, \{\Delta x_{ij}\}_{j \in \mathcal{N}^+_i \setminus \{i\}}\right]$.
- $H^\theta$:
  
  $X^0_i = \left[\Delta x_{ii}, 0, 0, \{\Delta x_{ij}, \|\Delta x_{ij}\|_2^2, \|\Delta x_{ij}\|_2\}_{j \in \mathcal{N}^+_i \setminus \{i\}}\right],$

  where $\Delta x_{ii} = x_i - x_i(KT)$ and $\Delta x_{ij} = x_i - x_j$.

The other networks are as follows. For the MLP and GNN, $W = 1$ and $(4 \times n) \times (4 \times 4 \times n)$ parameters from 1 unbiased layer/filter; for $H$, $W = 1$ and $(6 \times n) \times (5 \times 5 \times n)$ parameters from 1 unbiased layer/filter. The GNNSA has $3 \times (4 \times 8) + (8 \times 16)$ additional parameters from three self-attention matrices and one self-attention vector.
REFERENCES

[1] N. Atanasov, J. Le Ny, K. Daniilidis, and G. J. Pappas, “Decentralized active information acquisition: Theory and application to multi-robot SLAM,” in IEEE International Conference on Robotics and Automation, 2015, pp. 4775–4782.

[2] Y. Tian, Y. Chang, F. H. Arias, C. Nieto-Granda, J. P. How, and L. Carlone, “Kima-ulti: Robust, distributed, dense metric-semantic SLAM for multi-robot systems,” IEEE Transactions on Robotics, 2022.

[3] X. Kan, T. C. Thayer, S. Carpin, and K. Karydis, “Task planning on stochastic aisle graphs for precision agriculture,” IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 3287–3294, 2021.

[4] A. Pierson and M. Schwager, “Bio-inspired non-cooperative multi-robot herding,” in IEEE International Conference on Robotics and Automation, 2015, pp. 1843–1849.

[5] E. Sebastián and E. Montijano, “Multi-robot implicit control of herds,” in IEEE International Conference on Robotics and Automation, 2021, pp. 1601–1607.

[6] E. Sebastián, E. Montijano, and C. Sagüés, “Adaptive multirobot implicit control of heterogeneous herds,” IEEE Transactions on Robotics, 2022.

[7] L. Heintzmann, A. Hashimoto, N. Abaid, and R. K. Williams, “Anticipatory planning and dynamic lost person models for human-robot search and rescue,” in IEEE International Conference on Robotics and Automation, 2021, pp. 8252–8258.

[8] T. Z. Jiahao, L. Pan, and M. A. Hsieh, “Learning to swarm with knowledge-based neural ordinary differential equations,” in IEEE International Conference on Robotics and Automation, 2022, pp. 6912–6918.

[9] D. Bloembergen, K. Tuyls, D. Hennes, and M. Kaisers, “Evolutionary dynamics of multi-agent learning: A survey,” Journal of Artificial Intelligence Research, vol. 53, pp. 659–697, 2015.

[10] A. Khan, E. Tolstaya, A. Ribeiro, and V. Kumar, “Graph policy gradients for large scale robot control,” in Conference on Robot Learning, 2020, pp. 823–834.

[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 Conference on Robot Learning, 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 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2021, pp. 8944–8950.

[13] F. Yang and N. Matni, “Communication topology co-design in graph recurrent neural network based distributed control,” in IEEE Conference on Decision and Control, 2021, pp. 3619–3626.

[14] F. Gama, Q. Li, E. Tolstaya, A. Prorok, and A. Ribeiro, “Synthesizing decentralized controllers with graph neural networks and imitation learning,” IEEE Transactions on Signal Processing, vol. 70, pp. 1932–1946, 2022.

[15] L. Purieri, C. L. Galimberti, M. Zakwan, and G. Ferrari-Trecate, “Distributed neural network control with dependability guarantees: a compositional port-hamiltonian approach,” in Learning for Dynamics and Control Conference, 2022, pp. 571–583.

[16] R. Han, S. Chen, and Q. Hao, “Cooperative multi-robot navigation in dynamic environment with deep reinforcement learning,” in IEEE International Conference on Robotics and Automation, 2020, pp. 448–454.

[17] G. Shi, W. Hönig, Y. Yue, and S.-J. Chung, “Neural-swarm: Decentralized close-proximity multitotor control using learned interactions,” in IEEE International Conference on Robotics and Automation, 2020, pp. 3241–3247.

[18] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, “Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning,” in IEEE International Conference on Robotics and Automation, 2018, pp. 6252–6259.

[19] S. H. Semmann, H. Liu, M. Everett, A. De Ruiter, and J. P. How, “Multi-agent motion planning for dense and dynamic environments via deep reinforcement learning,” IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 3221–3226, 2020.

[20] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn, “Robonet: Large-scale multi-robot learning,” in Conference on Robot Learning, 2020, pp. 885–897.

[21] K. Bogert and P. Doshi, “Multi-robot inverse reinforcement learning under occlusion with estimation of state transitions,” Artificial Intelligence, vol. 263, pp. 46–73, 2018.

[22] H. Zhu, F. M. Charamunt, B. Brito, and J. Alonso-Mora, “Learning interaction-aware trajectory predictions for decentralized multi-robot motion planning in dynamic environments,” IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 2256–2263, 2021.

[23] S. Zhou, M. J. Phielipp, J. A. Sefair, S. I. Walker, and H. B. Amor, “Clone swarms: Learning to predict and control multi-robot systems by imitation,” in IEEE/RSJ International Conference on Intelligent Robots and Systems, 2019, pp. 4092–4099.

[24] Q. Li, W. Lin, Z. Liu, and A. Prorok, “Message-aware graph attention networks,” in International Conference on Learning Representations, 2020.

[25] A. Y. Ng, S. Russell, et al., “Algorithms for inverse reinforcement learning,” in International Conference on Machine Learning, vol. 1, 2000, p. 2.

[26] B. Wang, J. Xie, and N. Atanasov, “DARLIN: Distributed multi-agent reinforcement learning with one-hop neighbors,” in IEEE/RSJ International Conference on Intelligent Robots and Systems, 2022.

[27] C. L. Galimberti, L. Purieri, L. Xu, and G. Ferrari-Trecate, “Hamiltonian deep neural networks guaranteeing non-vanishing gradients by design,” arXiv preprint arXiv:2105.13205, 2021.

[28] R. Han, S. Chen, and Q. Hao, “Cooperative multi-robot navigation in dynamic environment with deep reinforcement learning,” in IEEE International Conference on Robotics and Automation, 2020, pp. 448–454.

[29] P. Li, W. Lin, Z. Liu, and A. Prorok, “Message-aware graph attention networks for large-scale multi-robot path planning,” IEEE Robotics and Automation Letters, vol. 6, no. 3, pp. 5533–5540, 2021.

[30] D. V. Kwak, J. J. Park, S. M. Kim, J. H. Park, and H. K. Lee, “Neural ordinary differential equations,” in Advances in Neural Information Processing Systems, vol. 31, 2018.

[31] R. Olfati-Saber, “Flocking for multi-agent dynamic systems: Algorithms and theory,” IEEE Transactions on Automatic Control, vol. 51, no. 3, pp. 401–420, 2006.

[32] G. Blankenstein, R. Ortega, and A. J. Van Der Schaft, “The matching conditions of controlled lagrangians and ida-passivity based control,” International Journal of Control, vol. 75, no. 9, pp. 645–665, 2002.

[33] J. C. Butcher, Numerical methods for ordinary differential equations. John Wiley & Sons, 2016.

[34] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” arXiv preprint arXiv:1710.05941, 2017.