Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks

Ekaterina Tolstaya
Department of Electrical and Systems Eng.
University of Pennsylvania, USA
`eig@seas.upenn.edu`

Fernando Gama
Department of Electrical and Systems Eng.
University of Pennsylvania, USA
`fgama@seas.upenn.edu`

James Paulos
Department of Mechanical Eng.
University of Pennsylvania, USA
`jpaulos@seas.upenn.edu`

George Pappas
Department of Electrical and Systems Eng.
University of Pennsylvania, USA
`pappasg@seas.upenn.edu`

Vijay Kumar
Department of Mechanical Eng.
University of Pennsylvania, USA
`kumar@seas.upenn.edu`

Alejandro Ribeiro
Department of Electrical and Systems Eng.
University of Pennsylvania, USA
`aribeiro@seas.upenn.edu`

Abstract: We consider the problem of finding distributed controllers for large networks of mobile robots with interacting dynamics and sparsely available communications. Our approach is to learn local controllers that require only local information and communications at test time by imitating the policy of centralized controllers using global information at training time. By extending aggregation graph neural networks to time varying signals and time varying network support, we learn a single common local controller which exploits information from distant teammates using only local communication interchanges. We apply this approach to the problem of flocking to demonstrate performance on communication graphs that change as the robots move. We examine how a decreasing communication radius and faster velocities increase the value of multi-hop information.

Keywords: Swarms, Imitation Learning, Graph Neural Networks

1 Introduction

Large scale swarms are composed of multiple agents that collaborate to accomplish a task. In the future, these systems could be deployed to, for example, provide on-demand wireless networks [1], perform rapid environmental mapping [2, 3], search after natural disasters [4, 5], or enable sensor coverage in cluttered and communications denied environments [6]. For as long as scale is moderate, the group can be controlled as a whole from a central agent. However, as we reach for larger number of agents, decentralized control becomes a necessity. Individual agents must decide on control actions that are conducive to accomplishing a collective task from their local observations and communication with nearby peers. It has long been known that finding optimal controllers in these distributed settings is challenging [7]. This motivates the use of heuristics in general and, in particular and as we advocate in this paper, the use of learned heuristics.

We emphasize that the challenge in decentralized control stems from the local information structure generated by the unavoidable restriction to communicate with only nearby agents. Building on this observation, we propose the use of imitation learning to train policies that respect the local information structure of a distributed system, while attempting to mimic the global policy of a clairvoyant expert (Section 2). The value of imitation learning has been demonstrated in single agent robotics problems such as autonomous driving [8] and quadrotor navigation [9]. When designing multi-agent systems, we must contend with the dimensionality growth of the system as new agents are added.

Both of these problems can be overcome if we use a Graph Neural Networks [10, 11, 12, 13, 14]. In particular, aggregation graph neural networks (aggregation GNNs) [13] are especially suited to teams of agents operating over a physical network because the architecture operates in an entirely local fashion involving only communication between nearby neighbors. In small-size multi-agent problems [15], it is shown that explicitly learning the graph of agent relationships aids in distilling decentralized agent policies from expert policies trained using Actor-Critic methods. In contrast, we leverage known relationships and connectivity between agents in order to extract features with graph convolutions, following the approach of [13]. Exploiting the known network structure allows us to consider teams an order of magnitude larger and highlights the value of using information from multi-hop neighbors.

Related work in multiagent learning for control in imitation [16] and reinforcement [17] settings address varying neighbor relationships in communication but provide no mechanism for multi-hop information flow. Explicit multi-hop message passing between all team members allows offline [18] and incremental [19] training for group inference but incurs a superlinear growth in communications with team size. Scalable online training is achieved in [20] by viewing the team as a distributed neural network, but training and inference must cease once agents move and the network changes.

We examine flocking tasks to highlight the ability of our approach to handle dynamic communication networks. Flocking has natural extensions to transportation and platooning of autonomous vehicles where agents rely on local observations to align their velocities and regulate their spacing [21]. Previous work focuses on developing local controllers which incorporate only observations from immediate neighbors [22, 23, 24]. We show that a global controller inspired by [24] outperforms such local controllers, but global approaches are not practical for real deployments. The novelty of our approach to flocking is to aggregate from multi-hop neighbors; this ability allows us to approach the performance of global solutions while respecting realistic communication constraints. Prior to this work, there has been no principled approach for augmenting the communication between neighbors to pass on information aggregated from multi-hop neighbors.

Notation. For a matrix $\mathbf{A}$ with entries a_{ij} we use $[\mathbf{A}]_{ij} = a_{ij}$ to represent its (i, j)th entry and $[\mathbf{A}]_i = [a_{i1}, \ldots, a_{iN}]$ to represent its ith row. For matrices $\mathbf{X}$ and $\mathbf{Y}$ we use $[\mathbf{X}, \mathbf{Y}]$ to represent its block column concatenation and $[\mathbf{X}; \mathbf{Y}]$ for its block row stacking.

2 Control of Networked Systems

We consider a team of N agents distributed in space and tasked with controlling some large scale dynamical process. We characterize this dynamical process by the collection of state values $\mathbf{x}_i(t) \in \mathbb{R}^p$ observed at the locations of each agent i at time t, as well as the control actions $\mathbf{u}_i(t) \in \mathbb{R}^q$ that agents take. Grouping local states into the joint system state matrix $\mathbf{x}(t) = [\mathbf{x}_1^\mathsf{T}(t); \ldots; \mathbf{x}_N^\mathsf{T}(t)] \in \mathbb{R}^{N \times p}$, and local actions into the overall system action $\mathbf{u}(t) = [\mathbf{u}_1^\mathsf{T}(t); \ldots; \mathbf{u}_N^\mathsf{T}(t)] \in \mathbb{R}^{N \times q}$, we can write the evolution of the dynamical process through a differential equation of the form, $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$.

In order to design a controller for this dynamical system we operate in discrete time by introducing a sampling time T_s and a discrete time index n. We define $\mathbf{x}_n = \mathbf{x}(nT_s)$ as the discrete time state and $\mathbf{u}_n$ as the control action held from time nT_s until time $(n+1)T_s$. Solving the differential equation between times nT_s and $(n+1)T_s$, we end up with the discrete dynamical system

$$\mathbf{x}_{n+1} = \int_{nT_s}^{(n+1)T_s} f(\mathbf{x}(t), \mathbf{u}_n) \ dt + \mathbf{x}_n, \quad \text{with } \mathbf{x}(nT_s) = \mathbf{x}_n. \tag{1}$$

At each point in (discrete) time, we consider a cost function $c(\mathbf{x}_n, \mathbf{u}_n)$. The objective of the control system is to choose actions $\mathbf{u}_n$ that reduce the accumulated cost $\sum_{n=0}^{\infty} c(\mathbf{x}_n, \mathbf{u}_n)$. When the collection of state observations $\mathbf{x}_n = [\mathbf{x}_{1n}^\mathsf{T}; \ldots; \mathbf{x}_{Nn}^\mathsf{T}]$ is available at a central location it is possible for us to consider centralized policies π that choose control actions $\mathbf{u}_n = \pi(\mathbf{x}_n)$ that depend on global information. In such case the optimal policy is the one that minimizes the expected long term cost. If the dynamics in $f(\mathbf{x}(t), \mathbf{u}(t))$ and the costs $c(\mathbf{x}_n, \pi(\mathbf{x}_n))$ are known, as we assume here, there are several techniques to find the optimal policy π^* [25, 26]. In this paper we are interested in decentralized controllers that operate without access to global information and interpret the optimal controller as a benchmark that decentralized controllers are trying to imitate.

The agent network is described by the presence of an edge $(i, j) \in \mathcal{E}_n$ which indicates that j may send data to i at time n. When this happens we say that j is a neighbor of i and define the neighborhood

of i at time n as the collection of all its neighbors, $\mathcal{N}_{in} = \{j : (i,j) \in \mathcal{E}_n\}$. We can also define multi-hop neighborhoods of a node. We first define the 0-hop neighborhood of i to be the node itself, $\mathcal{N}_{in}^0 = \{i\}$. The 1-hop neighborhood of i is defined as simply the neighborhood of i, $\mathcal{N}_{in}^1 = \mathcal{N}_{in}$. We can now define the k-hop neighborhood of i as the set of nodes that can reach node i in exactly k hops. The k-hop neighbors of i are the nodes that are $(k-1)$-hop neighbors at time $n-1$ of the neighbors of i at time n. This recursive neighborhood definition characterizes the information that is available to node i at time n. This information includes the local state $\mathbf{x}_{in}$ that can be directly observed by node i at time n as well as the value of the state $\mathbf{x}_{j(n-1)}$ at time $(n-1)$ for all nodes j that are 1-hop neighbors of i at time n since this information can be communicated to node i. Node i can also learn the state $\mathbf{x}_{j(n-2)}$ of 2-hop neighbors at time $n-2$ since that information can be relayed from neighbors. In general, Eqn. 2 defines the information history $\mathcal{H}_{in}$ of node i at time n as the collection of state observations out to a maximal history depth K.

$$\mathcal{N}_{in}^k = \left\{ j' \in \mathcal{N}_{j(n-1)}^{k-1} : j \in \mathcal{N}_{in} \right\}, \quad \mathcal{H}_{in} = \bigcup_{k=0}^{K-1} \left\{ \mathbf{x}_{j(n-k)} : j \in \mathcal{N}_{in}^k \right\} \tag{2}$$

The decentralized control problem consists on finding a policy that minimizes the long term cost $\sum_{n=0}^{\infty} c(\mathbf{x}_n, \mathbf{u}_n)$ restricted to the information structure in (2). This leads to problems in which finding optimal controllers is famously difficult to solve [7] except in some particularly simple scenarios [27]. This complexity motivates the introduction of a method that learns to mimic the global controller. Formally, we introduce a parameterized policy $\pi(\mathcal{H}_{in}, \mathbf{H})$ that maps local information histories $\mathcal{H}_{in}$ to local actions $\mathbf{u}_{in} = \pi(\mathcal{H}_{in}, \mathbf{H})$ as well as a loss function $\mathcal{L}(\pi, \pi^*)$ to measure the difference between the optimal centralized policy π^* and a system where all agents (locally) execute the (local) policy $\pi(\mathcal{H}_{in}, \mathbf{H})$. Our goal is to find the tensor of parameter $\mathbf{H}$ that solves the optimization problem

$$\mathbf{H}^* = \operatorname*{argmin}_{\mathbf{H}} \mathbb{E}^{\pi^*} \left[\mathcal{L} \left(\pi(\mathcal{H}_{in}, \mathbf{H}), \pi^*(\mathbf{x}_n) \right) \right], \tag{3}$$

where we use the notation $\mathbb{E}^{\pi^*}$ to emphasize that the distribution of observed states $\mathbf{x}_n$ over which we compare the policies $\pi(\mathcal{H}_{in}, \mathbf{H})$ and $\pi^*(\mathbf{x}_n)$ is that of a system that follows the optimal policy π^*. The formulation in (3) is one in which we want to learn a policy $\pi(\mathcal{H}_{in}, \mathbf{H})$ that mimics π^* to the extent that this is possible with the information that is available to each individual node. The success of this effort depends on the appropriate choice of the parameterization that determines the family of policies $\pi(\mathcal{H}_{in}, \mathbf{H})$ that can be represented by different choices of parameters, $\mathbf{H}$. In this work, we advocate the use of an aggregation graph neural network (Section 3) and demonstrate its applications to the problem of flocking with collision avoidance (Section 4).

3 Delayed Aggregation Graph Neural Networks

Aggregation graph neural networks (aggregation GNNs) are information processing architectures that operate on network data in a completely local and decentralized way, harnessing all the useful information by repeated exchanges with their neighbors [13]. This makes them suitable choices for parameterizing the decentralized policy $\pi(\mathcal{H}_{in}, \mathbf{H})$ in (3). However, aggregation GNNs operate on fixed graph signals defined over a fixed graph support, so in what follows, we extend aggregation GNNs to operate on time-varying graph processes defined over a time-varying graph support.

The graph support $\mathcal{G}_n$ is conveniently described by a *graph shift operator* $\mathbf{S}_n \in \mathbb{R}^{N \times N}$ which is a matrix whose element $[\mathbf{S}_n]_{ij}$ can be nonzero only if $(j,i) \in \mathcal{E}_n$ or if $i = j$, respecting the sparsity of the graph [28, 29, 30]. It follows that $\mathbf{S}_n$ defines linear, local operations since multiplications with $\mathbf{S}_n$ can be computed only with neighboring exchanges. More precisely, recall that $[\mathbf{x}_{(n-1)}]_j = \mathbf{x}_{j(n-1)}^{\mathsf{T}}$ denotes the state observed by node j at time $n-1$ and further distribute the product $\mathbf{S}_n \mathbf{x}_{n-1}$ so that $[\mathbf{S}_n \mathbf{x}_{n-1}]_i$ is associated with node i. Since $[\mathbf{S}_n]_{ij} \neq 0$ only if $(j,i) \in \mathcal{E}_n$ or if $i = j$ we can write

$$\left[\mathbf{S}_n \mathbf{x}_{n-1} \right]_i = \sum_{j=i, j \in \mathcal{N}_{in}} \left[\mathbf{S}_n \right]_{ij} \left[\mathbf{x}_{n-1} \right]_j. \tag{4}$$

Thus, node i can carry its part of the multiplication operation by receiving information from neighboring nodes. Examples of valid shift operators include weighted and unweighted adjacency matrices as well as weighted, unweighted, or normalized Laplacians. Aggregation GNNs leverage the locality of (4) to build a sequence of recursive k-hop neighborhood [Eq. (2)] aggregations to which

a neural network can be applied. More precisely, consider a sequence of signals $\mathbf{y}_{kn} \in \mathbb{R}^{N \times p}$ that we define through the recursion

$$\mathbf{y}_{kn} = \mathbf{S}_n \mathbf{y}_{(k-1)(n-1)}, \tag{5}$$

with the initialization $\mathbf{y}_{0n} = \mathbf{x}_n$. If we fix the time n and consider increasing values of k, the recursion in (5) produces a sequence of signals where the first element is $\mathbf{y}_{0n} = \mathbf{x}_n$, the second element is $\mathbf{y}_{1n} = \mathbf{S}_n \mathbf{y}_{0(n-1)} = \mathbf{S}_n \mathbf{x}_{n-1}$, and, in general, the kth element is $\mathbf{y}_{kn} = (\mathbf{S}_n \mathbf{S}_{n-1} \dots \mathbf{S}_{n-k+1}) \mathbf{x}_{n-k}$. Thus, (5) is modeling the diffusion of the state $\mathbf{x}_{n-k}$ through the sequence of time varying networks $\mathbf{S}_{n-k+1}$ through $\mathbf{S}_n$. This diffusion can be alternatively interpreted as an aggregation; at node i we can define an aggregation sequence of K elements as

$$\mathbf{z}_{in} = \left[\left[\mathbf{y}_{0n} \right]_i ; \left[\mathbf{y}_{1n} \right]_i ; \dots ; \left[\mathbf{y}_{(K-1)n} \right]_i \right]. \tag{6}$$

The first element of this sequence is $[\mathbf{y}_{0n}]_i = [\mathbf{x}_n]_i = \mathbf{x}_{in}^\top$ which represents the local state of node i. The second element of this sequence is $[\mathbf{y}_{1n}]_i = [\mathbf{S}_n \mathbf{x}_{n-1}]_i$ which aggregates the states $\mathbf{x}_{j(n-1)}$ of 1-hop neighboring nodes $j \in \mathcal{N}_{in}^1$ observed at time $n-1$ with a weighted average. In fact, this element is precisely the outcome of the local average shown in (4). If we now focus on the third element we see that $[\mathbf{y}_{2n}]_i = [\mathbf{S}_n \mathbf{S}_{n-1} \mathbf{x}_{n-2}]_i$ is an average of the states $\mathbf{x}_{j(n-2)}$ of 2-hop neighbors $j \in \mathcal{N}_{in}^2$ at time $n-2$. In general, the $k+1$st element of $\mathbf{z}_{in}$ is $[\mathbf{y}_{kn}]_i = [\mathbf{S}_n \mathbf{S}_{n-1} \dots \mathbf{S}_{n-k+1} \mathbf{x}_{n-k}]_i$ which is an average of the states $\mathbf{x}_{j(n-k)}$ of k-hop neighbors $j \in \mathcal{N}_{in}^k$ observed at time $n-k$. From this explanation we conclude that the sequence $\mathbf{z}_{in}$ is constructed with state information that is part of the local history $\mathcal{H}_{in}$ defined in (2) and therefore a valid basis for a decentralized controller. We highlight in equations (4) and (5) that agents forward the aggregation of their neighbors' information, rather than a list of neighbors' states, further along to multi-hop neighbors.

An important property of the aggregation sequence $\mathbf{z}_{in}$ is that it exhibits a regular temporal structure as it is made up of nested aggregation neighborhoods. This regular structure allows for the application of a regular convolutional neural network (CNN) [13] of depth L, where for each layer $\ell = 1, \dots, L$, we have

$$\mathbf{z}_{in}^{(\ell)} = \sigma^{(\ell)}(\mathbf{H}^{(\ell)} \mathbf{z}_{in}^{(\ell-1)}) \tag{7}$$

with $\sigma^{(\ell)}$ a pointwise nonlinearity and $\mathbf{H}^{(\ell)}$ a bank of small-support filters containing the *learnable* parameters. For each node i, we set $\mathbf{z}_{in}^{(0)} = \mathbf{z}_{in}$ and collect the output $\mathbf{z}_{in}^{(L)} = \mathbf{u}_{in}$ to be the decentralized control action at node i, at time n. We note that the filters $\mathbf{H}^{(\ell)}$ are shared across all nodes. We refer the reader to Section 7.1 in the supplementary materials for further details on aggregation GNNs.

The aggregation GNN architecture, described in equations (5)-(7), constitutes a local parameterization of the policy $\pi(\mathcal{H}_{in}, \mathbf{H})$ that exploits the network structure and involves communication exchanges only with neighboring nodes. To *learn* the parameters for $\mathbf{H}$, we use a training set $\mathcal{T}$ consisting of sample trajectories $(\mathbf{x}_n, \pi^*(\mathbf{x}_n))$ obtained from the global controller $\mathbf{u}_n^* = \pi^*(\mathbf{x}_n)$. We thus minimize the loss function over this training set, Eq. (3), where $\mathbf{u}_n$ collects the output $\mathbf{u}_{in} = \mathbf{z}_{in}^{(L)}$ of (7) at each node i:

$$\mathbf{H}^* = \underset{\mathbf{H}}{\arg\min} \sum_{(\mathbf{x}_n, \pi^*(\mathbf{x}_n)) \in \mathcal{T}} \mathcal{L}\left(\mathbf{u}_n, \mathbf{u}_n^* \right) \tag{8}$$

We note that the policy learned from (8) can be extended to any network since the filters $\mathbf{H}^{(\ell)}$ can be applied independently of $\mathbf{S}_n$, facilitating transfer learning. This transfer is enabled by sharing the filter weights $\mathbf{H}$ among nodes at training time. The learned aggregation GNN models are, therefore, network and node independent. Graph covariance is an advantage for the applications examined in this work, but may not be a suitable assumption in all problems [31].

4 Methods: Learning to Flock

We evaluate the proposed methodology by learning a local controller for flocking with collision avoidance [24]. Consider then a team of N agents in which $\mathbf{r}_i$ denotes the position of agent i and $\mathbf{v}_i = \dot{\mathbf{r}}_i$ denotes its velocity. We assume that accelerations are fully controllable and therefore attempt to design control inputs $\mathbf{u}_i$ so that the change in velocity of agent i is $\dot{\mathbf{v}}_i = \mathbf{u}_i$. Further denote as $\mathbf{r}_{ij} := \mathbf{r}_j - \mathbf{r}_i$ the relative position of agent j with respect to agent i and introduce a constant $\rho > 1$ to define the collision avoidance potential

$$U(\mathbf{r}_i, \mathbf{r}_j) = \|\mathbf{r}_{ij}\|^2 - \log \|\mathbf{r}_{ij}\|^2 \text{ if } \|\mathbf{r}_{ij}\| < \rho; \quad 1/\rho^2 - \log(\rho^2) \text{ otherwise.} \tag{9}$$

4

Since in addition to avoid collisions we want all agents to coordinate on their velocities we propose the controller

$$\mathbf{u}_i^* = -\sum_{j=1}^N (\mathbf{v}_i - \mathbf{v}_j) - \sum_{j=1}^N \nabla_{\mathbf{r}_i} U(\mathbf{r}_i, \mathbf{r}_j). \tag{10}$$

The combination of the collision avoidance potential with the velocity agreement term $\mathbf{v}_i - \mathbf{v}_j$ pushes the agents to coordinate their velocities while avoiding collisions. Observe that the potential $U(\mathbf{r}_i, \mathbf{r}_j)$ diverges at $\|r_{ij}\| = 0$, has a minimum when the distance between agents is $\|r_{ij}\| = 1$ and is indifferent when $\|r_{ij}\| > \rho$. It therefore pushes agents to either be at distance $\|r_{ij}\| = 1$ of each other or at distance $\|r_{ij}\| > \rho$ of each other.

The controller in (10) requires access to all agent velocities and all agent positions. In the parlance of Section 2, it is a clairvoyant centralized controller. In practice, agents can have access to local information only as they can only sense and communicate with agents within distance $\|\mathbf{r}_{ij}\| < R$ of each other. A controller that respects the locality of sensing and communication is

$$\mathbf{u}_i^{\dagger} = -\sum_{j \in \mathcal{N}_i} (\mathbf{v}_i - \mathbf{v}_j) - \sum_{j \in \mathcal{N}_i} \nabla_{\mathbf{r}_i} U(\mathbf{r}_i, \mathbf{r}_j). \tag{11}$$

The controller in (11) differs from the centralized controller in (10) in that the sums are over agents $j \in \mathcal{N}_i$ defined as those whose distance to agent i is $\|\mathbf{r}_{ij}\| < R$ for some communication radius R. For both the centralized and decentralized controllers, we have chosen $R = \rho$. It follows that the controller in (11) is decentralized as it can be implemented by accessing local information only. The controllers in (10) and (11) have the same stationary points but (11) may – indeed, it most often does – take longer to coordinate agent velocities. In the next section we use an aggregation GNN to learn a local controller that mimics (10). We will show that this learned controller outperforms (11) and achieves a performance that is similar to (10).

The global and local controllers are both non-linear in the states of the agents. The classical aggregation GNN approach does not allow for non-linear operations prior to aggregation, so we cannot use the position and velocity vectors alone to imitate the global controller. Rather than directly using the state $[\mathbf{r}_i, \mathbf{v}_i]$ of each node i for aggregation, we design the relevant features needed to replicate the non-linear controller using only a linear aggregation operation, where $[\mathbf{x}_n]_i \in \mathbb{R}^6$:

$$[\mathbf{x}_n]_i = \left[\sum_{j \in \mathcal{N}_i} (\mathbf{v}_{i,n} - \mathbf{v}_{j,n}), \quad \sum_{j \in \mathcal{N}_i} \frac{\mathbf{r}_{ij,n}}{\|\mathbf{r}_{ij,n}\|^4}, \quad \sum_{j \in \mathcal{N}_i} \frac{\mathbf{r}_{ij,n}}{\|\mathbf{r}_{ij,n}\|^2} \right] \tag{12}$$

This observation vector is then used during aggregation as described in (5)-(6). The local controller also requires the computation of (12), so we are not giving the GNN an unfair advantage by providing the instantaneous measurements of neighbors' states. We quantify the cost of a trajectory by the variance in velocities, where T is the number of time steps in the trajectory. The variance in velocities measures how far the system is from consensus in velocities [32]:

$$C = \frac{1}{N} \sum_{n=1}^T \sum_{j=1}^N \left\| \mathbf{v}_{j,n} - \frac{1}{N} \left[\sum_{i=1}^N \mathbf{v}_{i,n} \right] \right\|^2. \tag{13}$$

For our baseline scenario we consider a flock of $N = 100$ agents with a communication radius of $R = \rho = 1.0$ m and a discretization time period of $T_s = 0.01$ s. The flock locations were initialized uniformly on the disc with radius $\sqrt{N}$ to normalize the density of agents for changing flock sizes. Initial agent velocities are controlled by a parameter $v_{init} = 3.0$ m/s: agent velocities are sampled uniformly from the interval $[-v_{init}, +v_{init}]$ and then a bias for the whole flock is added, also sampled from $[-v_{init}, +v_{init}]$. To eliminate unsolvable cases, configurations are resampled if any agent fails to have at least two neighbors or if agents begin closer than 0.1 m. Finally, acceleration commands are saturated to the range $[-100, 100]$ m/s^2 to improve the numerical stability of training.

To obtain an estimate of the action to take, each agent operates an aggregation GNN architecture, as described in Section III, with input features given by (12). The aggregated vectors $\mathbf{z}_{in}$ in Eq. (6) are built from $K-1$ neighbor exchanges, and then fed into a fully connected neural network as per Eq. (7), with two hidden layers of 32 neurons each and a Tanh activation function. The network was implemented in PyTorch and trained over a MSE cost function, using the Adam optimizer with learning rate $5 \cdot 10^{-5}$ and forgetting factors $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Each model was trained

(a) Average difference in velocities

(b) Average minimum distance to a neighbor

(c) Flock positions using the GNN

(d) Flock positions using the local controller
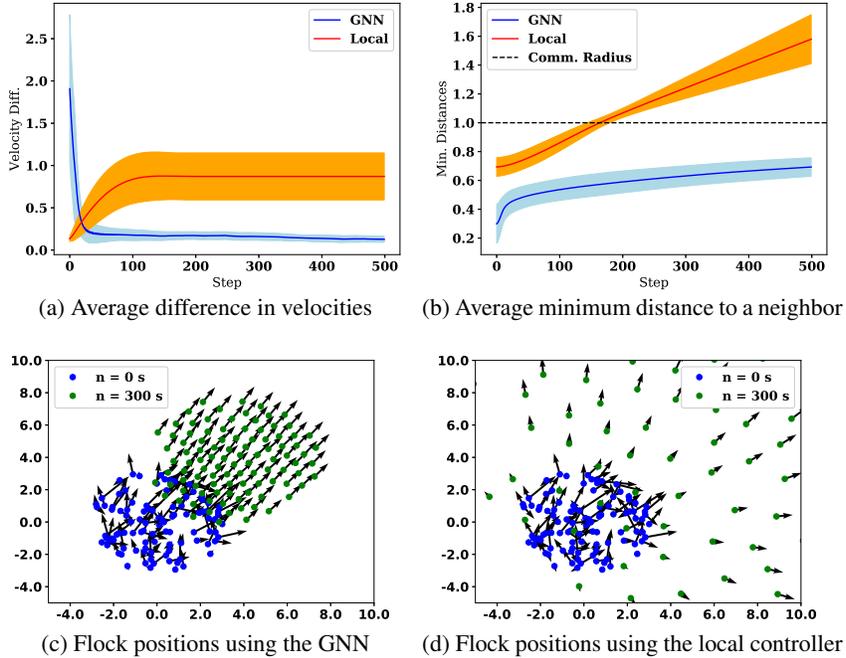
Figure 1. The GNN ($K = 3$) maintains a cohesive flock, while the local controller allows the flock to scatter.

using 400 trajectories, each of length 200 steps total. For testing, 20 trajectories of length 200 were observed by using the learned controller only.

In practice, following the optimal policy to collect training data results in a distribution of states that is not representative of those seen at test time. To resolve this we use the Dataset Aggregation (DAgger) algorithm and follow the learner's policy instead of the expert's with probability $1 - \beta$ when collecting training trajectories [33]. The probability β of choosing the expert action while training is decayed by a factor of 0.993 after each trajectory to a minimum of 0.5.

5   Results

We report results comparing (11) and (10) for point masses with fully controllable accelerations in Section 5.1. This simple setting allows for an exploration of the effect of different system parameters such as initial velocity or communication radius, to determine experimentally the scenarios on which the aggregation GNN offers good performance. In Section 5.2 we study the case of transfer learning, where we train the model in one network but test it in another (for example, with different number of agents), and also by exporting the trained architecture to other physical models beyond the point-mass model, as shown in the AirSim simulator (Sec. 7.2).

5.1   Learning to flock with point masses

First, we compare the performance of the GNN controller with $K = 3$ to the local controller $\mathbf{u}_i^{\dagger}$ (11). Figure 1a depicts the magnitude of velocity differences between agents over the course of a trajectory in terms of the population mean and standard devision. The GNN converges much more rapidly and, unlike the local controller, approaches a perfect velocity consensus. Part of the reason for this is explained in Figure 1b, which plots agents' minimum distance to any neighbor over time. The GNN control approaches a uniform flock spacing, but the local controller fails to stop agents from dispersing quickly enough. Soon the local controller's network has become completely disconnected as agent distances exceed their communication range of $R = 1$. One flock trajectory is depicted for the GNN controller in Fig. 1c and the local controller in Fig. 1d. Each diagram shows the initial agent positions and velocities at time $n = 0$ and then at $n = 300$, qualitatively illustrating the stable flocking of the GNN and failure of the local controller.
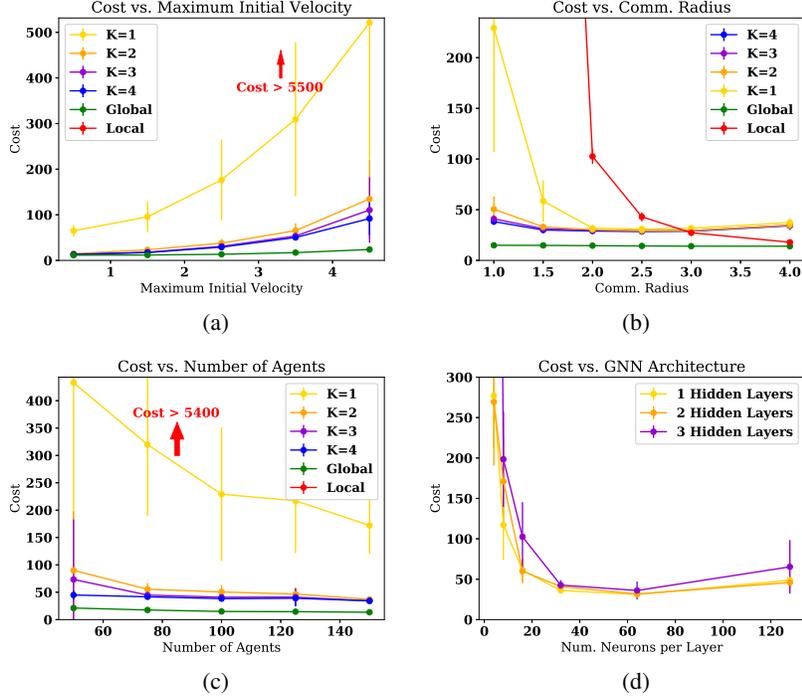
Figure 2. The flock's maximum initial velocities, communication radius, and the number of agents are key parameters affecting the control cost. The GNN architecture uses 2 hidden layers with 32 neurons each.

Next, we study the performance of the GNN controller (for different values of K) compared to both the local controller $\mathbf{u}_i^\dagger$ and the global controller $\mathbf{u}_i^*$ [eq. (10)] under different flocking scenarios; namely, different initial velocities (Fig. 2a), communication radius (Fig. 2b), and number of agents (Fig. 2c). Then, we also consider different aggregation GNN architecture hyperparameters (Fig. 2d. We recall that the performance is measured by the cost metric defined in (13). In general, we observe that the GNN cost is bounded by the global controller as a best-case baseline, and the the local controller as the worst-case baseline, as expected, showing, in many cases, a marked improvement over the local controller $\mathbf{u}_i^\dagger$.

More specifically, in Fig. 2a, we fix the communication radius $R = 1.0$ m for $N = 100$ agents. The $K = 1$ controller, which uses the same data as the local controller, performs an order of magnitude worse than $K = 2$ to $K = 4$, whose performance is comparable. At the highest initial velocity of $v_{init} = 4.5$ m/s, the $K = 4$ GNN performs slightly better than the rest. Fig. 2b shows that the flocking problem becomes more challenging as the communication radius decreases, for fixed maximum initial velocities, $v_{init} = 3.0$ m/s, and $N = 100$ agents. The difference between the GNNs is most dramatic at $R = 1.0$ m, where the $K = 3$ and 4 controllers perform much better than all but the global controller. It is interesting to note that the cost of the local controller dips below the GNN controllers for large communication radii such as $R = 4.0$ m. This may be due to the lack of truncation of the GNN features in (12). In Fig. 2c, we observe that the flocking cost per agent, computed by (13), decreases in value and variance as the flock size increases, with fixed $R = 1.0$ m and $v_{max} = 3.0$ m/s. The GNN approach generalizes easily to 150 agents, with no penalties on larger flocks. In Fig. 2d, we observe that the GNN architecture resulting in the lowest control cost uses 1-2 hidden layers and 32-64 neurons, for fixed $v_{init} = 3.0$, $N = 100$, $R = 1.0$ m. For smaller architectures, we observe under-fitting.

5.2 Transfer to Leader Following

Next, we investigate a new application of flocking in the presence of leader agents, with which the rest of the flock must align velocities. This application is extremely relevant for human-robot interaction for control of large swarms, and has not been previously explored by [24]. In the previous section, the flock positions and velocities are initialized at random. This induces a symmetry in the

7

(a) Transfer to flocking with leaders.

(b) Transfer to grid formation.

(c) Flocking with two leader agents

(d) Grid of agents with radial velocities
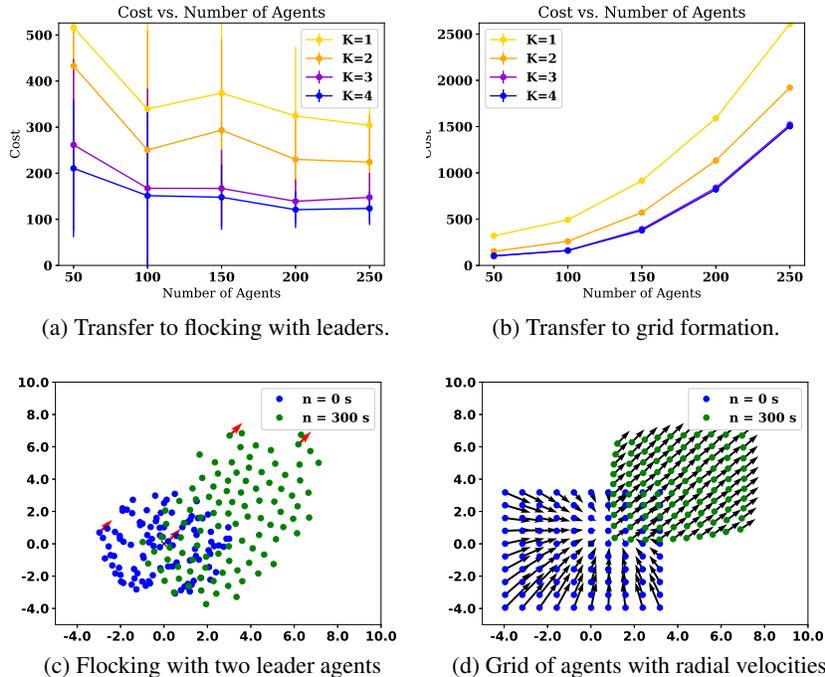
Figure 3. The trained GNN is transferred to new challenging scenarios with large numbers of agents.

configuration - the distribution of velocities in an agent's 1-hop neighborhood is the same as in its 2-hop or 3-hop neighborhood. Now, our goal is to investigate applications in which this symmetry is not present, to emphasize the capabilities of the aggregation GNN for $K = 3$ and $K = 4$. All models in this experiment were trained for $v_{init} = 3.0$ m/s, $N = 100$, and $R = 1.0$ m.

We first examine the transfer of the trained controllers to a system with two leader agents that have equal velocities that remain constant throughout an episode. That is, we train the architecture on examples of the symmetric network, but we test it on the leader system. Results are shown in Fig. 3c. We can observe that the $K = 3$ and $K = 4$ aggregation GNNs provide the best performance which is in agreement with the intuition that 2-hop and 3-hop aggregations allow the agents to respond more quickly to a leader agent who may be multiple hops away. In the second scenario, agents are initialized in a grid, with velocities radially inwards toward the centroid of the flock. The initial velocities of agents are initialized to be proportional to their distance from the center of the flock. Results are shown in Fig. 3b. As the number of agents increases, we see the cost increases exponentially because the initial velocities increase, which is consistent with Fig. 2a. This scenario is particularly prone to the scattering behavior of Fig. 1d due to the high probability of near-collisions, but the $K = 3$ and $K = 4$ controllers successfully align the agents' velocities and promote regular spacing among agents.

6 Conclusion

We have demonstrated the utility of aggregation graph neural networks as a tool for automatically learning distributed controllers for large teams of agents with coupled state dynamics and sparse communication links. We envision the use of an Aggregation GNN-based controller in large-scale drone teams deployed into communication-limited environments to accomplish coverage, surveillance or mapping tasks. In these settings, it is critical for agents to incorporate information from distant teammates in spite of local communication constraints; we show aggregations GNNs can be extended to accomplish this even with the time-varying agent states and time-varying communication networks typical of mobile robots. In experiments, learning decentralized controllers for flocking and additional applications confirms the value of multi-hop information to performance and robustness to number of agents and communication radius. In future work, enforcing state or input constraints could help avoid these failure modes.

References

[1] V. Sharma, M. Bennis, and R. Kumar. UAV-assisted heterogeneous networks for capacity enhancement. *IEEE Communications Letters*, 20(6):1207–1210, 2016.

[2] S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In *Robotics and Automation, 2000. Proceedings. ICRA 2000. IEEE International Conference on*, volume 1, pages 321–328. IEEE, 2000.

[3] S. Thrun and Y. Liu. Multi-robot slam with sparse extended information filers. In *Robotics Research. The Eleventh International Symposium*, pages 254–266. Springer, 2005.

[4] J. L. Baxter, E. Burke, J. M. Garibaldi, and M. Norman. Multi-robot search and rescue: A potential field based approach. In *Autonomous robots and agents*, pages 9–16. Springer, 2007.

[5] J. S. Jennings, G. Whelan, and W. F. Evans. Cooperative search and rescue with a team of mobile robots. In *Advanced Robotics, 1997. ICAR'97. Proceedings., 8th International Conference on*, pages 193–200. IEEE, 1997.

[6] H. Zhang and J. C. Hou. Maintaining sensing coverage and connectivity in large sensor networks. *Ad Hoc & Sensor Wireless Networks*, 1(1-2):89–124, 2005.

[7] H. S. Witsenhausen. A counterexample in stochastic optimum control. *SIAM Journal on Control*, 6(1):131–147, 1968.

[8] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[9] A. Giusti, J. Guzzi, D. C. Ciresan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2016.

[10] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)*, Banff, Apr. 2014. URL http://arxiv.org/abs/1213.6203.

[11] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Annu. Conf. Neural Inform. Process. Syst. 2016*, Barcelona, Spain, 5-10 Dec. 2016. NIPS Foundation.

[12] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th Int. Conf. Learning Representations*, Toulon, France, 24-26 Apr. 2017. Assoc. Comput. Linguistics.

[13] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro. Convolutional neural network architectures for signals supported on graphs. *IEEE Trans. Signal Process.*, 67(4):1034–1049, Feb. 2019.

[14] L. Ruiz, F. Gama, A. G. Marques, and A. Ribeiro. Median activation functions for graph neural networks. In *44th IEEE Int. Conf. Acoust., Speech and Signal Process.*, Brighton, UK, 12-17 May 2019. IEEE. URL http://arxiv.org/abs/1810.12165.

[15] A. Tacchetti, H. F. Song, P. A. M. Mediano, V. Zambaldi, J. Kramar, N. C. Rabinowitz, T. Graepel, M. Botvinick, and P. W. Battaglia. Relational forward models for multi-agent learning. In *International Conference on Learning Representations*, New Orleans, May 2019.

[16] J. Paulos, S. W. Chen, D. Shishika, and K. Vijay. Decentralization of multiagent policies by learning what to communicate. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, May 2019.

[17] M. Hüttenrauch, A. Šošić, and G. Neumann. Deep Reinforcement Learning for Swarm Systems. *Journal of Machine Learning Research*, 20(54):1–31, 2019.

[18] G. A. D. Caro, A. Giusti, J. Nagi, and L. M. Gambardella. A simple and efficient approach for cooperative incremental learning in robot swarms. In *2013 16th International Conference on Advanced Robotics (ICAR)*, pages 1–8, Nov. 2013. doi:10.1109/ICAR.2013.6766596.

[19] A. Giusti, J. Nagi, L. Gambardella, and G. A. D. Caro. Cooperative sensing and recognition by a swarm of mobile robots. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 551–558, Oct. 2012. doi:10.1109/IROS.2012.6385982.

[20] M. Otte. An emergent group mind across a swarm of robots: Collective cognition and distributed sensing via a shared wireless neural network. *The International Journal of Robotics Research*, 37(9):1017–1061, Aug. 2018. doi:10.1177/0278364918779704.

[21] H. G. Tanner. Flocking with obstacle avoidance in switching networks of interconnected vehicles. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 3006–3011. Citeseer, 2004.

[22] C. W. Reynolds. Flocks, Herds and Schools: A Distributed Behavioral Model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 25–34, New York, NY, USA, 1987. ACM. doi:10.1145/37401.37406.

[23] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on automatic control*, 48(6):988–1001, 2003.

[24] H. G. Tanner, A. Jadbabaie, and G. J. Pappas. Stable flocking of mobile agents part ii: dynamic topology. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 2, pages 2016–2021. IEEE, 2003.

[25] K. Zhou, J. C. Doyle, K. Glover, et al. *Robust and optimal control*, volume 40. Prentice hall New Jersey, 1996.

[26] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[27] C. Eksin, P. Molavi, A. Ribeiro, and A. Jadbabaie. Bayesian quadratic network game filters. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 4589–4593. IEEE, 2013.

[28] A. Sandryhaila and J. M. F. Moura. Big data analysis with signal processing on graphs. *IEEE Signal Process. Mag.*, 31(5):80–90, Sep. 2014.

[29] S. Chen, R. Varma, A. Sandryhaila, and J. Kovačević. Discrete signal processing on graphs: Sampling theory. *IEEE Trans. Signal Process.*, 63(24):6510–6523, Dec. 2015.

[30] A. G. Marques, S. Segarra, G. Leus, and A. Ribeiro. Sampling of graph signals with successive local aggregations. *IEEE Trans. Signal Process.*, 64(7):1832–1843, Apr. 2016.

[31] F. Gama, J. Bruna, and A. Ribeiro. Stability properties of graph neural networks. *arXiv:1905.04497v2 [cs.LG]*, 4 Sep. 2019. URL http://arxiv.org/abs/1905.04497.

[32] L. Xiao, S. Boyd, and S.-J. Kim. Distributed average consensus with least-mean-square deviation. *Journal of parallel and distributed computing*, 67(1):33–46, 2007.

[33] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

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

[35] D. Mellinger. *Trajectory Generation and Control for Quadrotors*. Ph.D., University of Pennsylvania, Philadelphia, 2012.

7 Appendix

7.1 Aggregation GNN Evaluation

(a) $\mathbf{y}_{0n}$　　　　(b) $\mathbf{y}_{1n}$　　　　(c) $\mathbf{y}_{2n}$　　　　(d) $\mathbf{y}_{3n}$

(e) Processing at the i-th node
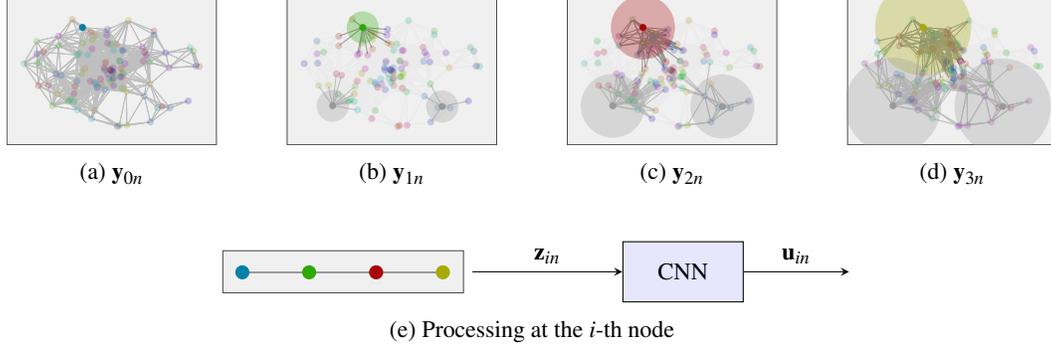
Figure 4. Aggregation graph neural networks perform successive local exchanges between each node and its neighbors. For each k-hop neighborhood (illustrated by the increasing disks), record $\mathbf{y}_{kn}$ (5) to build signal $\mathbf{z}_n$ which exhibits a regular structure (6). Figures (a) The value of the state (b) One-hop neighborhood (c) Two-hop neighborhood (d) Three-hop neighborhood. (e) Once the regular time-structure signal $\mathbf{z}_n$ is obtained, we take each row $\mathbf{z}_{in}$, representing the information collected at node i, and we process it through a CNN to obtain the value of the decentralized controller $\mathbf{u}_{in}$ at node i at time n (7).

In this appendix, we present a detailed algorithm to compute the output of Aggregation GNNs, see Fig. 4. Essentially, each node in the network exchanges information with its neighbors repeatedly, building sequence $\mathbf{z}_{in}$ (6). Since this sequence is a regular sequence (i.e. nearby entries in $\mathbf{z}_{in}$ correspond to the information aggregated from neighboring nodes), a traditional CNN can be directly applied, and an output computed. This CNN is applied locally at each node, with parameters shared across all nodes.

More specifically, Algorithm 1 summarizes the inference methodology for the aggregation GNN at a single node of the network. At time n, the agent receive aggregation sequences from its neighbors $z_{j(n-1)}$. Then, the agent pools this information from neighbors to form the current aggregation vector z_{in} which is input to the learned controller $\pi(\mathbf{z}_{in}, \mathbf{H})$ to compute the new action $\mathbf{u}_{in}$. Finally, the agent transmits its aggregation vector $\mathbf{z}_{in}$ to its current neighbors $\mathcal{N}_{in}$.

Algorithm 1 Aggregation Graph Neural Network at Agent i.

1: **for** n=0,1,..., **do**
2:　　Receive aggregation sequences from $j \in \mathcal{N}_{in}$ [eq. (6)]

$$\mathbf{z}_{j(n-1)} = \left[\left[\mathbf{y}_{0(n-1)} \right]_j ; \left[\mathbf{y}_{1(n-1)} \right]_j ; \ldots ; \left[\mathbf{y}_{(K-1)(n-1)} \right]_j \right]$$

3:　　Update aggregation sequence components [eq. (5) and (4)]

$$\left[\mathbf{y}_{kn} \right]_i = \left[\mathbf{S}_n \mathbf{y}_{k(n-1)} \right]_i = \sum_{j=1, j \in \mathcal{N}_{in}} \left[\mathbf{S}_n \right]_{ij} \left[\mathbf{y}_{k(n-1)} \right]_j$$

4:　　Observe system state $\mathbf{x}_{in}$
5:　　Update local aggregation sequence [cf. (6)]

$$\mathbf{z}_{in} = \left[\mathbf{x}_{in}^{\mathsf{T}} ; \left[\mathbf{y}_{1n} \right]_i ; \ldots ; \left[\mathbf{y}_{(K-1)n} \right]_i \right]$$

6:　　Compute next local action using the learned controller

$$\mathbf{u}_{in} = \pi\left(\mathbf{z}_{in}, \mathbf{H} \right)$$

7:　　Transmit local aggregation sequence $\mathbf{z}_{in}$ to neighbors $j \in \mathcal{N}_{in}$
8: **end for**

7.2 High order dynamics

The ideal point masses provide a convenient benchmark for testing the effects of system parameters, but ultimately, we are most interested in flocking for robotic systems. Our goal in this section is compare several approaches to flocking in large teams of up to 50 quadrotors in simulation. Testing in the AirSim simulator allows us to test our controllers in the presence of higher order dynamics, slower control rates, and latency in observations [34].

The first challenge is that the AirSim simulator runs in real time, and produces an order of magnitude fewer training data points as compared to the point-mass simulation. To speed up training, we attempt to train on a simulated point mass system that was tuned to the parameters of AirSim. Since, we were not able to achieve an exact control frequency for the simulation, we instead observed that commands were issued at a variable time interval, with a mean of 0.12 seconds, and a standard deviation of 0.018 seconds. Therefore, we adapted the point mass simulation to match the AirSim simulation by sampling $T_s \sim \mathcal{N}(0.12, 3 \times 10^{-4})$ and applying the linear model in (1). The inputs and outputs of the controllers were scaled by a factor of $l = 6$ before evaluation of the GNN so that the optimal spacing dictated by the potential function (9) does not result in collisions. The scaling of the control actions and features was performed by: $\mathbf{x}_{n+1} = f(\mathbf{x}_n, l \cdot \pi(\mathbf{x}_n/l))$, where f is the dynamics of the system, and $l = 6$ is the scaling factor.

The second challenge was converting from the desired acceleration values produced by the controller to the desired roll and pitch commands. We follow the approach of [35], which linearizes the dynamics of the robot about the hover point. In our case, the roll, ϕ, and pitch, θ, are proportional to the desired acceleration in the x and y dimensions. For a fixed yaw orientation, we used the approximation: $\phi = \mathbf{u}_2/g$ and $\theta = -\mathbf{u}_1/g$. The signs are inconsistent with those of [35] due to the AirSim convention that the $+Z$ dimension is down. This approximation is only valid for small accelerations, so the acceleration inputs were clipped to a range of $[-3.0, 3.0]$ m/s^2. Performance of the controllers was quantified using the variance of velocities in (13). We initialize the flock uniformly spaced in a grid formation with a distance of about 4.8 m between agents, and with x/y velocities sampled uniformly on the range $[-3.0, 3.0]$ m/s. The communication radius in AirSim was fixed at 9 m.

Fig. 5 compares the performance of controllers trained in simulation against those trained on the stochastic point mass model. The results of all eight GNN models are framed by the local and global controller as the upper and lower baselines. The GNN controller with $K = 4$ trained in AirSim outperformed all other learned controllers. For $K = 1$ and $K = 2$, the controllers trained in simulation and AirSim had similar performance, but as additional aggregation operations are added, the benefit of training in AirSim becomes more obvious. We believe that the $K = 4$ model trained on point masses has overfit to the ideal dynamics, so the performance degrades. Both in simulation and in point-mass experiments, we observed a failure mode, in which a small group of agents is moving too quickly and escapes from the rest of the group. This small sub-flock typically exhibits flocking behavior among the several agents, but has no ability to re-join the flock, because it is permanently outside of the communication range of the rest of the agents. This drawback results from the lack of hard constraints on the connectivity in the system.
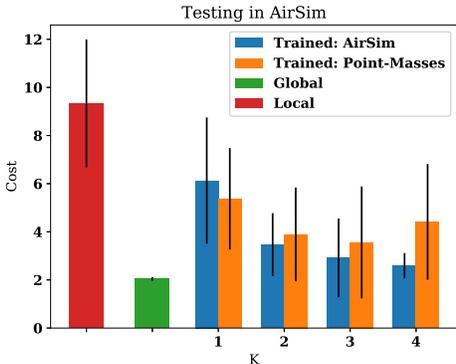
Figure 5. Four models were trained in AirSim and four on the stochastic point masses, and then all models were tested in AirSim.