Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks

Ekaterina Tolstaya¹, Fernando Gama¹, James Paulos², George Pappas¹, Vijay Kumar¹, Alejandro Ribeiro¹

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 which require only local information and local 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 a decentralized linear quadratic regulator problem and observe how faster communication rates and smaller network degree increase the value of multi-hop information. Separate experiments learning a decentralized flocking controller demonstrate performance on communication graphs that change as the robots move.

I. INTRODUCTION

The modern world depends on the fixed infrastructure provided by large scale cyber-physical systems. Smart-grid utilities, road traffic control systems, and industrial processes are all engineering challenges which involve complex plant dynamics, sparse placements of sensors and actuators, and communication constraints which require local controllers to perform with locally available information.

In the future, we envision hundreds of mobile robots working cooperatively to perform services such as providing on-demand wireless networks [21], environmental mapping [27, 26], search after natural disasters [11, 13], or sensor coverage in cluttered and communications denied environments [22]. Mobile robots force distributed solutions. The task may require shared information, but computation, sensing, and actuation are local to each robot. Mobility also poses unique challenges not present with fixed infrastructure. The communication graph between agents is no longer fixed. In addition, we wish robots teams to obtain resiliency through redundancy and interchangeability, and this motivates deploying a single shared local policy for each agent instead of a suite of specialized local controllers.

Our goal is to learn local agent policies which can collectively control a large scale dynamical process using information gathered through a sparse, potentially time-varying communication network. Our approach is to use imitation learning to approximate a global optimal policy using networked local controllers which at test time require only local information and connectivity. To improve the quality of these local policies, information from non-adjacent neighbors is distilled using Aggregation Graph Neural Networks.

Related work in decentralized optimal control is concerned with the design of optimal controllers given foreknowledge of both the plant model and of information-sharing constraints imposed on the controller by an existing communication network. In the case of a linear time invariant (LTI) system with the special property that these network constraints happen to be quadratically invariant with respect to the plant, the optimal control problem has a convex formulation [17]. Unfortunately, the resulting controller implementations are not scalable for many systems of interest. In particular, quadratic invariance requires all locally collected information to be exchanged in the common case where all system states may directly or indirectly influence each other. This observation has led to further work emphasizing the synthesis of localized controllers by imposing structure on the closed loop system response as part of the design process [28, 29]. However, these methods still assume fixed communication graphs and yield controllers specialized to each node and so are not directly applicable to teams of interchangeable mobile robots.

We address these difficulties by learning local controllers. However, the behavior cloning techniques which have been successfully applied to robotics applications such as autonomous driving [16] and quadrotor navigation [11] can not be directly applied to multi-robot teams. From the point of view of each agent, its local control system is partially observable, with other agents’ unobservable states affecting future values. Therefore, each agent must accumulate information from neighbors to gain a better understanding of the state of the system, and to choose better control actions.

Graph neural networks (GNNs) offer a solution to this problem of aggregating network information by exploiting the graph structure [3, 5, 14, 9, 18]. In particular, the aggregation GNN developed in [9, 10] offers an architecture that operates in an entirely local fashion, involving communication only with nearby neighbors, making it especially suited for teams of agents interacting over a physical network. In the context of multi-agent systems, [23] uses imitation learning to learn decentralized policies from expert policies originally trained using Actor-Critic methods. In that approach, a graph of relationships between agents is explicitly learned by a neural network. In contrast, we leverage the known relationships and connectivity between agents in order to use graph convolutions to extract features, following the approach of [9]. Exploiting the known network structure allows us to consider teams of agents an order of

This work is supported by NSF DGE-1321851, ARL DCIST CRA W911NF-17-2-0181, Intel Science and Technology Center for Wireless Autonomous Systems (ISTC-WAS), NSF CCF 1717120, and ARO W911NF1710438.

¹Department of Electrical and Systems, ²Department of Mechanical Engineering, University of Pennsylvania, Philadelphia, PA 19104, USA
eig@seas.upenn.edu
magnitude larger, and highlights the value of using information from multi-hop neighbors.

We begin by describing the optimal control problem for a dynamical process, and then pose the additional information constraints which define the more difficult decentralized control problem for a networked system. In Section 3 we introduce an extension of Aggregation Graph Neural Networks to time varying signals supported on time varying networks and their application to information exchange among teams of agents. This framework is applied to the specific problem of distributed LQR control in Section 4, and experiments illustrate the value of aggregating multi-hop information as communication rate or degree of the communication network are varied. Successful transfer of models learned on one network to a different network motivates the study of flocking dynamics in Sect 5, where now the communication network is explicitly time varying as the agents move.

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

## II. CONTROL OF NETWORKED SYSTEMS

We consider a team of $N$ agents distributed in space and use $r_i(t)$ to denote the possibly time varying position of agent $i$ at time $t$. A dynamical process is evolving in the space where the team is deployed. We characterize this dynamical process by the collection of state values $x_i(t) \in \mathbb{R}^p$ observed at the locations of each agent $i$, as well as the control actions $u_i(t) \in \mathbb{R}^q$ that agents take, and a noise term $w_i(t) \in \mathbb{R}^p$ to model uncertainties and model mismatch. Grouping local states into the joint system state matrix $x(t) = [x_1^T(t) ; \ldots ; x_N^T(t)] \in \mathbb{R}^{N \times p}$, local actions into the overall system action $u(t) = [u_1^T(t) ; \ldots ; u_N^T(t)] \in \mathbb{R}^{N \times q}$ and noise terms into the vector $w(t) = [w_1^T(t) ; \ldots ; w_N^T(t)] \in \mathbb{R}^{N \times p}$, we can write the evolution of the dynamical process through a differential equation of the form

$$\dot{x}(t) = f(x(t), u(t), w(t)).$$

(1)

Although not formally required, we are interested in processes in which the function $f$ has spatial locality in the sense that the effect of $x_i(t)$ and $u_i(t)$ on the state $x_j(t)$ of a different agent diminishes with their distance $||r_i(t) - r_j(t)||$ (see Section IV for an example).

In order to design a controller to affect the behavior of the dynamical system in (1), we operate in discrete time. To do so, introduce a sampling time $T_k$ and a discrete time index $n$. Define the discrete time state $x_n = x(nT_k)$ and denote as $u_n$, the action that the system takes at time $nT_k$ and holds until $(n+1)T_k$. Solving (1) between times $nT_k$ and $(n+1)T_k$ we end up with the discrete dynamical system

$$x_{n+1} = f_{nT_k}^{(n+1)T_k} f(x(t), u_n, w(t)) \, dt, \text{ with } x(nT_k) = x_n.$$  

(2)

At each point in (discrete) time, we consider a cost function $c(x_n, u_n)$. The objective of the control system is to choose actions $u_n$ that reduce the accumulated cost $\sum_{n=0}^{\infty} c(x_n, u_n)$. When the collection of state observations $x_n = [x_{n1}^T; \ldots ; x_{nN}^T]$ is available at a central location it is possible for us to consider centralized policies $\pi$ that choose control actions $u_n = \pi(x_n)$ that depend on global information. In such case the optimal policy is the one that minimizes the expected long term cost

$$\pi^* = \arg\min_{\pi} \mathbb{E} \left[ \sum_{n=0}^{\infty} c(x_n, \pi(x_n)) \right]$$

(3)

If the dynamics in $f(x(t), u(t), w(t))$ and the costs $c(x_n, \pi(x_n))$ are known, as we assume here, there are several techniques to find the optimal policy $\pi^*$. In this paper we are interested in decentralized controllers that operate without access to global information and interpret (3) as a benchmark that decentralized controllers are trying to imitate.

### A. Decentralized Control via Imitation of Central Control

The locations of agents at time $t = nT_k$ determine a connectivity graph $\mathcal{G}_n$ with an asymmetric edge set $\mathcal{E}_n$ composed of pairs $(i, j)$ having associated weights $w_{ij}$.

As in the case of the dynamical process in (1) we think of the weights as decreasing with the agents’ distance $\|r_i - r_j\|$ but do not formally require it (see Section IV for an example). We further define the weighted adjacency matrix $W_n$ as a sparse matrix with nonzero entries $[W_n]_{ij} = w_{ij}$ for $(i, j) \in \mathcal{E}_n$. For all $(i, j) \notin \mathcal{E}_n$ we have $[W_n]_{ij} = 0$.

The presence of the edge $(i, j) \in \mathcal{E}_n$ means that it is possible for $j$ to 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

$$N_{in} = \{ j : (i, j) \in \mathcal{E}_n \}.$$  

(4)

It is also of interest to define multihop neighborhoods of a node. To do so begin by convening that the 0-hop neighborhood of $i$ is $N_{i0}^0 = \{i\}$, namely, the node itself. Further rename the neighborhood of $n$ as the 1-hop neighborhood and denote $N_{in}^1 = N_{in}$. We can now define the $k$-hop neighborhood of $i$ as the set of nodes that can reach node $i$ in exactly $k$ hops. Their formal definition can be given by the recursion

$$N_{in}^k = \left\{ j : \exists j \in N_{jn}^{k-1} : j \in N_{in} \right\}.$$  

(5)

As per (5), 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 is made in order to characterize the information that is available to node $i$ at time $n$. This information includes the local state $x_{in}$ that can be directly observed by node $i$ at time $n$ as well as the value of the state $x_{jn}$ 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 $x_{jn-2}$ of 2-hop neighbors at time $n-2$ since that information can be relayed from neighbors.

In general, we can define the information history of node $i$ at time $n$ as the collection of state observations

$$\mathcal{H}_{in} = \bigcup_{k=0}^{K-1} \{ x_{jn-k} : j \in N_{in}^k \}.$$

(6)
where we have chosen a maximal history depth $K$. The decentralized control problem consists of finding a policy that minimizes the long term cost $\sum_{t=0}^{\infty} c(x_n, u_n)$ restricted to the information structure in [6]. This leads to problems in which finding optimal controllers is famously difficult to solve [30] except in some particularly simple scenarios [6]. This complexity motivates the introduction of a method that learns to mimic the centralized controller in (3). Formally, we introduce a parametrized policy $\pi(\mathcal{H}_{in}, H)$ that maps local information histories $\mathcal{H}_{in}$ to local actions $u_n = \pi(\mathcal{H}_{in}, H)$ as well as a loss function $L(\pi, \pi^*)$ to measure the difference between the optimal centralized policy $\pi^*$ and a system where all agents (locally) execute the (local) policy $\pi(\mathcal{H}_{in}, H)$. Our goal is to find the tensor of parameter $H$ that solves the optimization problem

$$H^* = \arg\min_{H} \mathbb{E}[L(\pi(\mathcal{H}_{in}, H), \pi^*(x_n))].$$  \hspace{1cm} (7)

where we use the notation $\mathbb{E}[\cdot]$ to emphasize that the distribution of observed states $x_n$ over which we compare the policies $\pi(\mathcal{H}_{in}, H)$ and $\pi^*(x_n)$ is that of a system that follows the optimal policy $\pi^*$.

The formulation in (7) is one in which we want to learn a policy $\pi(\mathcal{H}_{in}, H)$ that mimics $\pi^*$ to the extent that the information that is available to each individual node. The success of this effort depends on the appropriate choice of the parametrization that determines the family of policies $\pi(\mathcal{H}_{in}, H)$ that can be represented by different choices of parameters, $H$. In this work, we advocate the use of an aggregation graph neural network (Section IV) and demonstrate its applications to the problem of distributed control of a linear system (Section V) and to the problem of flocking with collision avoidance (Section V).

### III. AGGREGATION GRAPH NEURAL NETWORKS

We propose the adaptation of an aggregation graph neural network (GNN) [9, 10] for the learning parametrization of the policy $\pi(\mathcal{H}_{in}, H)$ in (7). We begin by following the graph signal processing literature to introduce a graph shift operator $S$ for the learning parametrization of the aggregation graph neural network (Section III) and demonstrate its applications to the problem of distributed control of a linear system (Section IV) and to the problem of flocking with collision avoidance (Section V).

The choice of the parametrization that determines the family of possible with the information that is available to each individual node $i$, at time $n-1$ and further distribute the product $S_{n}x_{n-1}$ so that $[S_{n}x_{n-1}]_{i}$ is associated with node $i$. Since $[S_{n}]_{ij} \neq 0$ only if $(i,j) \in \mathcal{E}_n$ or if $i = j$ [22, 19, 4, 58]. This property means that the shift operator abides to the sparsity pattern of the graph $G_n$ and that we can therefore implement multiplications by $S_n$ using information exchanges between neighboring nodes. Indeed, recall that $[x_{(n-1)}]_{j} = x_{jn}^T$ denotes the state observed by node $j$ at time $n-1$ and further distribute the product $S_{n}x_{n-1}$ so that $[S_{n}x_{n-1}]_{i}$ is associated with node $i$. Since $[S_{n}]_{ij} \neq 0$ only if $(i,j) \in \mathcal{E}_n$ or if $i = j$ we can write

$$[S_{n}x_{n-1}]_{i} = \sum_{j=1}^{N\in\mathcal{E}_n} [S_{n}]_{ij} [x_{n-1}]_{j}.$$  \hspace{1cm} (8)

Thus, node $i$ can carry its part of the multiplication operation by receiving information from neighboring nodes. We point out that the adjacency matrix $W_n$ has a sparsity pattern that makes it a valid shift operator. Other eligible shift operators are unweighted and normalized adjacency matrices as well as weighted, unweighted, or normalized Laplacians. In the experiments in Sections V we set $S_n = W_n$, but our methods are applicable to any matrix for which the local computation in (8) is feasible.

Aggregation GNNs leverage the locality of [9] to build a sequence of recursive $k$-hop neighborhood [cf. [3]] aggregations to which a neural network can be applied; see Fig. 1. More precisely, consider a sequence of signals $y_{in} \in \mathbb{R}^{N\times T}$ that we define through the recursion

$$y_{in} = S_{n}y_{(k-1)(n-1)},$$  \hspace{1cm} (9)

with the initialization $y_{0n} = x_{n}$. If we fix the time $n$ and consider increasing values of $k$, the recursion in (9) produces a sequence of signals where the first element is $y_{0n} = x_{n}$, the second element is $y_{1n} = S_{n}y_{(0)(n-1)} = S_{n}x_{n-1}$, and, in general, the $k$th element is $y_{kn} = (S_{n} \ast S_{n-1} \ast \ldots \ast S_{n-k+1})x_{n-k}$. Thus, (9) is modeling the diffusion of the state $x_{n-k}$ through the sequence of time varying networks $S_{n-k+1}$ through $S_{n}$. This diffusion can be equally interpreted as an aggregation. Indeed, if we restrict attention to node $i$ and limit the diffusion to $K$ elements we can define the aggregation sequence at node $i$ as

$$z_{in} = \left[ y_{0n}^T; y_{1n}^T; \ldots; y_{(K-1)n}^T \right]^T.$$  \hspace{1cm} (10)

The first element of this sequence is $[y_{0n}]_{i} = [x_{n}]_{i} = x_{in}^T$ which represents the local state of node $i$. The second element of this sequence is $[y_{1n}]_{i} = [S_{n}x_{n-1}]_{i}$ which aggregates the states $x_{n}$ of 1-hop neighboring nodes $j \in \mathcal{N}_n$ observed at time $n-1$ with a weighted average. In fact, this element is precisely the outcome of the local average shown in (8). If we now focus on the third element we see that $[y_{2n}]_{i} = [S_{n}S_{n-1} \cdots S_{n-K+1}]_{i}$ is an average of the states $x_{jn}$ of 2-hop neighbors $j \in \mathcal{N}_{n-1}$ at time $n-2$. In general, the $k+1$st element of $z_{in}$ is $[z_{kn}]_{i} = [S_{n-k} \cdots S_{n}]_{i}$ which is an average of the states $x_{jn}$ of $k$-hop neighbors $j \in \mathcal{N}_{n-k}$ observed at time $n-k$. From this explanation we conclude that the sequence $z_{in}$ is constructed with state information that is part of the local history $\mathcal{H}_{in}$ defined in (6) and therefore a valid basis for a decentralized controller. We highlight in equations (8) and (9) that agents forward the aggregation of their neighbors’ information, rather a lists of neighbors’ states, further along to multi-hop neighbors; see Algorithm 1.

An important property of the aggregation sequence $z_{in}$ 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) [9] of depth $L$, where for each layer $\ell = 1, \ldots, L$, we have

$$z_{in}^{(\ell)} = \sigma^{(\ell)}(H_{\ell}^{(\ell)}z_{in}^{(\ell-1)}),$$  \hspace{1cm} (11)

with $\sigma^{(\ell)}$ a pointwise nonlinearity and $H_{\ell}^{(\ell)}$ a bank of small-support filters containing the learnable parameters. For each node $i$, we set $z_{0n}^{(\ell)} = z_{in}$ and collect the output $z_{in}^{(L)} = u_{in}$ to be the decentralized control action at node $i$, at time $n$. We note that the filters $H_{\ell}^{(\ell)}$ are shared across all nodes.

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 $z_{jn(n-1)}$. Then, the agent pools this information from neighbors to form
the current aggregation vector $z_{in}$ which is input to the learned controller $\pi(z_{in}, \mathbf{H})$ to compute the new action $u_{in}$. Finally, the agent transmits its aggregation vector $z_{in}$ to its current neighbors $\mathcal{N}_{in}$.

The aggregation GNN architecture, described in equations (9)-(11), constitutes a local parametrization 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 $(x_n, \pi^*(x_n))$ obtained from the centralized controller $u_n^* = \pi^*(x_n)$, cf. (3). We thus minimize the loss function over this training set, cf. (7).

$$\mathbf{H}^* = \arg\min_{\mathbf{H}} \sum_{(x_n, \pi^*(x_n)) \in \mathcal{T}} \mathcal{L}(u_n, u_n^*)$$

where $u_n$ collects the output $u_{in} = z_{in}^{(L)}$ of (11) at each node $i$.

**Remark 1:** We note that the policy learned from (12) can be extended to any network since the filters $\mathbf{H}^{(l)}$ 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.

**IV. DISTRIBUTED LINEAR QUADRATIC REGULATOR**

To illustrate these ideas, consider a linear time invariant system with white Gaussian noise in which the state and action at node $i$ are scalars. In this case we have matrices $\mathbf{A} \in \mathbb{R}^{N \times N}$ and $\mathbf{B} \in \mathbb{R}^{N \times N}$ so that the generic model in (1) reduces to

$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t) + \mathbf{w}(t),$$

where the noise $\mathbf{w}(t)$ is a Wiener process with covariance $\Sigma$. This system when sampled with a rate $T_s$ produces a linear system in discrete time characterized by the linear difference equation

$$x_{n+1} = \mathbf{A}_T x_n + \mathbf{B}_T u_n + \mathbf{w}_n,$$

where the matrices are given by $\mathbf{A}_T = e^{\mathbf{A} T_s}$ and $\mathbf{B}_T = \int_0^{T_s} e^{\mathbf{A} s} \mathbf{B} \, ds$ and the noise is normal white with covariance $\Sigma_T = \int_0^{T_s} e^{\mathbf{A} s} \Sigma e^{\mathbf{A}^T s} \, ds$ [3]. We further choose the cost function to be of the quadratic form

$$c(x_n, u_n) = x_n^T \mathbf{Q} x_n + u_n^T \mathbf{R} u_n.$$  

For the linear system in (14) and the quadratic cost in (15), it is well known that the optimal policy is the linear map

$$\pi^*(x_n) = \mathbf{K}_T x_n,$$

in which $\mathbf{K}_T$ is the solution of the algebraic Riccati equation [33]. The controller in (15) cannot be implemented in a distributed manner. For once, we have made no assumption on the sparsity patterns of any of the matrices involved. But even if we assume that the matrices $\mathbf{A}_T$, $\mathbf{B}_T$, and $\Sigma_T$ match the sparsity of the graph, the gain $\mathbf{K}_T$ may be a full matrix as it involves matrix inverses. Therefore, we adopt the strategy

**Remark 1:** We note that the policy learned from (12) can be extended to any network since the filters $\mathbf{H}^{(l)}$ 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.

**IV. DISTRIBUTED LINEAR QUADRATIC REGULATOR**

To illustrate these ideas, consider a linear time invariant system with white Gaussian noise in which the state and action at node $i$ are scalars. In this case we have matrices $\mathbf{A} \in \mathbb{R}^{N \times N}$ and $\mathbf{B} \in \mathbb{R}^{N \times N}$ so that the generic model in (1) reduces to

$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t) + \mathbf{w}(t),$$

where the noise $\mathbf{w}(t)$ is a Wiener process with covariance $\Sigma$. This system when sampled with a rate $T_s$ produces a linear system in discrete time characterized by the linear difference equation

$$x_{n+1} = \mathbf{A}_T x_n + \mathbf{B}_T u_n + \mathbf{w}_n,$$

where the matrices are given by $\mathbf{A}_T = e^{\mathbf{A} T_s}$ and $\mathbf{B}_T = \int_0^{T_s} e^{\mathbf{A} s} \mathbf{B} \, ds$ and the noise is normal white with covariance $\Sigma_T = \int_0^{T_s} e^{\mathbf{A} s} \Sigma e^{\mathbf{A}^T s} \, ds$ [3]. We further choose the cost function to be of the quadratic form

$$c(x_n, u_n) = x_n^T \mathbf{Q} x_n + u_n^T \mathbf{R} u_n.$$  

For the linear system in (14) and the quadratic cost in (15), it is well known that the optimal policy is the linear map

$$\pi^*(x_n) = \mathbf{K}_T x_n,$$

in which $\mathbf{K}_T$ is the solution of the algebraic Riccati equation [33]. The controller in (15) cannot be implemented in a distributed manner. For once, we have made no assumption on the sparsity patterns of any of the matrices involved. But even if we assume that the matrices $\mathbf{A}_T$, $\mathbf{B}_T$, and $\Sigma_T$ match the sparsity of the graph, the gain $\mathbf{K}_T$ may be a full matrix as it involves matrix inverses. Therefore, we adopt the strategy

**Algorithm 1** Aggregation Graph Neural Network at Agent $i$.

1: for $n=0,1,\ldots$, do
2: Receive aggregation sequences from $j \in \mathcal{N}_{in}$ [cf. (10)]
   $$z_{j(n-1)} = \begin{bmatrix} y_{0(n-1)}^j \mid y_{1(n-1)}^j \mid \cdots \mid y_{(K-1)(n-1)}^j \end{bmatrix}$$
3: Update aggregation sequence components [cf. (9) and (8)]
   $$y_n = \sum_{j \in \mathcal{N}_{in}} \mathbf{S}_n y_{(n-1)}^j$$
4: Observe system state $x_n$
5: Update local aggregation sequence [cf. (10)]
   $$z_n = \begin{bmatrix} y_n \mid \cdots \mid y_{(K-1)n} \end{bmatrix}$$
6: Compute next local action using the learned controller
   $$u_n = \pi(z_{in}, \mathbf{H})$$
7: Transmit local aggregation sequence $z_n$ to neighbors $j \in \mathcal{N}_{in}$
8: end for
of learning a decentralized controller by imitating the optimal policy $\pi^*(x_n) = K_T x_n$ [cf. (5)]. Our choice of parametrization is the aggregation Graph Neural Network described in Section III.

A. LQR Distributed over a Geometric Network

We now consider the special case of distributed LQR control for a linear system whose dynamic structure and communication network are both linked to an underlying random geometry graph induced by the positions of agents [15]. We are given the locations of the $J$ agents distributed over the unit square, where $r_i \in I \times I$ is the location of agent $i$. The relationship between the states of all agents is encoded using a linear system (14) with an $A$ matrix determined by the distance between agents, where the elements of the $A$ are:

$$a_{ij} = \begin{cases} 0 & i = j \\ e^{-\alpha\|r_i - r_j\|^2} & i \neq j \end{cases} \quad (17)$$

The constant $\alpha$ regulates the dependence between agents. To reduce interference, we regulate the transmission power to keep the number of neighbors constant. For a given $D$, the cardinality of the set of in-neighbors of agent $i$ is $\|N_i\| = D$. To construct the set of neighbors of each agent $i$, we allow connections to the nearest $D$ agents. Each agent can receive information from $D$ nearest in-neighbors, and knows their weights in the linear system. The time-invariant communication network, denoted $W$, has weighted connections to each node from its $D$-nearest neighbors: $w_{ij} = a_{ij}$ if node $i$ is a $D$-nearest neighbor of node $j$, and 0 otherwise.

The fixed network degree allows us to illuminate the effects of aggregation in systems with limited connectivity. Also, it is necessary to divide the matrix $S$ by its largest eigenvalue to enable the numerical stability of aggregation operations.

B. Neural Network Architecture

The agent architecture follows the construction of the Aggregation GNN in Section III. The aggregated vectors of length $K$ were input to a fully connected neural network with one hidden layer of $2K$ neurons and a ReLu activation function. Following the notation of (11), this architecture corresponds to using $L = 2$ and a full matrix for $H^{(l)}$. The network was trained using the smooth L1 training loss function with L2 regularizer of $10^{-8}$, and the training process implemented using the Numpy and PyTorch libraries.

C. Experiment Parameters

Next, we provide a summary of the experimental parameters used for the LQR experiments. We used systems with 100 agents and the following system matrices: $B = 10I$, $Q = I$, $R = 100I$. The state of the system $x_0$ was initialized by drawing a sample from a uniform distribution on $[-1, 1]$. The covariance of the white noise $w$ was $\Sigma = 0.01I$. The default sampling time was chosen to be $T_s = 0.01$. The parameter regulating the strength of connections between nodes was set to be $\alpha = 10.0$.

To train the Aggregation GNN, we used 500 system trajectories of length 120. The reported LQR cost was averaged over 200 trajectories of length 100 at steady state for each of 10 random graphs for both the optimal solution and the neural network. For the discretization and degree experiments, we report the ratio of the LQR cost obtained by the GNN divided by the cost obtained by the optimal controller. For the transfer experiment, we provide the median trajectory cost, due to the presence of a small number of divergent trajectories.

D. Sampling Time and Aggregation Filter Length

It was shown by [17] that the system dynamics need to be sufficiently slow relative to the communication and control clocks to enable the use of a distributed controller. The effect is two-fold: changing the discretization of the linear system slows down the dynamics of the system and makes the control system less sensitive to errors in the control due to decreased eigenvalues of $A$. For discretizations below, but not including, 0.05, the network can be controlled using the Aggregation GNN. Networks with smaller discretizations, $T_s = 0.005$ and $T_s = 0.001$, benefit slightly from a longer aggregation length of $K = 4$, while the systems for $T_s = 0.01$ do not. We hypothesize that for larger discretizations, the delay in the information aggregated from distant multi-hop neighbors renders this information useless to the controller.

E. Transfer to New Systems

A transfer learning experiment was performed to examine the performance of a GNN trained on one system, but tested on another. For the aggregation, we change the graph to that of the current test system, but use the old neural network weights from the model trained on the original system. This enables the generalization of our learned controller to new graphs.

Five linear systems were generated, and for each degree, the corresponding networks were computed. Then, a total of 30 GNN models were trained, five for each degree (columns). The baseline results for testing a model on the same system it was trained on are provided in the "Self" row. Then, 5 new systems were generated for testing of the 30 models. For each row, the a new network was computed for each of the 5 systems. The median cost value across the 25 combinations of systems, for 200 trajectories each, are reported in each cell of the table. The parameters $K = 4, T_s = 0.01$ were fixed for these experiments. Most of the GNN controllers generalized well to

| $K$ | $0.05$ | $0.01$ | $0.005$ | $0.001$ |
|-----|-------|-------|--------|--------|
| 1   | -     | -     | -      | -      |
| 2   | 0.88  | 1.43  | 1.32   |        |
| 3   | 0.98  | 1.03  | 1.00   |        |
| 4   | 0.97  | 1.01  | 1.01   |        |
| 5   | 0.98  | 1.01  | 1.01   |        |
| 6   | 0.98  | 1.01  | 1.02   |        |
the new systems, except the controller trained for a network degree of 4 (Fig. 3).

These results suggest that training one model for all nodes of a large linear system already allows the network behave correctly for nodes in previously-unseen systems. The weight sharing approach improves generalization, but differences in connectivity can change the controller performance. We observed a large variation in the LQR cost, with some experiments showing better performance on the previously-unseen systems.

F. Network Degree and Aggregation Filter Length

We aim to understand the relationship between the connectivity of the network and the impact of aggregation (Fig. 3). In the problem formulation, we fix the degree of the communication network to emphasize the effects of multi-hop aggregation. In networks with fewer connections, increasing the length of the aggregation filter is essential for ensuring good control performance. We show that aggregation is less impactful for highly connected networks. For the degree of 4, each node can obtain information from only the 4 nearest neighbors. In this case, an aggregation filter length of 4 is necessary to obtain acceptable control performance. This allows each node to indirectly obtain information from 3-hop neighbors. On the other hand, for the network a degree of 14, aggregation past the 1-hop neighborhood (K=2) is unnecessary because the information from the fourteen 1-hop neighbors is sufficient for control.

V. FLOCKING

We examine the flocking to highlight the ability of our approach to handle dynamic communication networks. The broader applications of flocking include transportation and platooning of autonomous vehicles [24], in which agents must align their velocities and regulate their spacing for safety and efficiency. Our goal is to approximate a global controller for flocking using an Aggregation GNN that has access only to local information.

Ideally, we want a controller that works as well as the global controller, but respects the communication constraints in our large-scale distributed system. We demonstrate that a global controller outperforms local controller proposed by [25]. The novelty of our approach to flocking is the ability to aggregate and use information from multi-hop neighbors. Previous local approaches to flocking have used only single-hop neighbors’ information [25, 12]. Prior to this work, there has been no principled approaches for augmenting the communication between neighbors to pass on information aggregated from multi-hop neighbors.

Consensus algorithms do consider multi-hop communication, but, as compared to [12], we are interested in controllers that incorporate collision avoidance and go beyond simple consensus. We follow the approach of [25], which incorporates a potential function to regulate the spacing of agents. Though we note that this approach for collision avoidance does not guarantee a lack of collisions, or the preservation of the connectivity within the flock. We observe that the flock often splits up into smaller groups, and that agents in the sub-flocks remain connected to each other, but become disconnected from the majority of the flock.

The Aggregation GNN controller is trained using a point-mass physics model to imitate a global controller derived from the local controller of [25]. Our experiments test the impact of longer aggregation filter lengths and we demonstrate that aggregating from 2-hop neighbors does help with control, but we believe that longer filter lengths are not useful due to the continuing changes in the connectivity of the network and agent states. We also demonstrate the capability of our trained controllers to generalize from a point-mass control system to a simulated flock of quadrotors in the AirSim simulator [20].

A. Flocking Dynamics

Following the approach of [25], we describe the acceleration-controlled dynamics of each agent \(i\) to be:

\[
\ddot{r}_i = v_i, \quad \dot{v}_i = u_i \tag{18}
\]

where \(r_i = (x_i, y_i)^T\) and \(\dot{r}_i = (\dot{x}_i, \dot{y}_i)^T\), the two-dimensional position and velocity of each agent, \(r_i, \dot{r}_i \in \mathbb{R}^2\). The input to the system is \(u_i = (u_{x_i}, u_{y_i})^T\), the acceleration of the agent in each of the two dimensions, \(u_i \in \mathbb{R}^2\). We discretize this system for each agent \(i\) at time step \(k\) and add noise to the velocity component, \(w_{i,k} \in \mathbb{R}^2, w_{i,k} \sim \mathcal{N}(0, I)\):

\[
\begin{bmatrix}
\dot{r}_{i,n+1} \\
\dot{v}_{i,n+1}
\end{bmatrix} =
\begin{bmatrix}
1 & T \cdot I \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
r_{i,n} \\
v_{i,n}
\end{bmatrix} +
\begin{bmatrix}
0 & 0.17, I
\end{bmatrix}
\begin{bmatrix}
u_{i,n} \\
w_{i,n}
\end{bmatrix} \tag{19}
\]
In the next section, we are assuming that each agent can measure its’ neighbors positions and velocities instantaneously, so the implicit time index is $n$ for all quantities of the global and local controllers. We drop the time index $n$ for clarity in the section below.

B. Global and Local Control for Flocking

We denote the relative position between agents, $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$. An agent $i$ is in agent $j$’s set of neighbors $\mathcal{N}_i$ if the distance between the agents is less than a threshold, $\|\mathbf{r}_{ij}\| < R$. Only agents within range can observe each others’ states and participate in the aggregation of information. In the case of the global controller, the velocity and position of all agents is measured simultaneously. Therefore, the flock’s network $\mathcal{S}$ is an unweighted adjacency matrix with no self loops, where the edge weight $s_{ij} = 1$ if $\|\mathbf{r}_{ij}\| < R$ and 0 otherwise.

We follow the approach of [25] to derive a feedback controller for flocking based on potential functions that regulate the distances between agents. One potential function that satisfies these conditions depends on the distance between the two agents, $\|\mathbf{r}_{ij}\|$:  

$$
U_{ij} = \begin{cases} 
\frac{1}{\|\mathbf{r}_{ij}\|^2} + \log \|\mathbf{r}_{ij}\|^2 & \|\mathbf{r}_{ij}\| < R \\
VR & \|\mathbf{r}_{ij}\| \geq R
\end{cases}
$$

We restrict this function to be continuous at $V_R = \frac{1}{R^2} + \log(R^2)$ and compute $V_R$ accordingly. Next, we define the potential function $U_i$ as the aggregation of the potentials of the neighbors of node $i$:  

$$
U_i = (N - \|\mathcal{N}_i\|)VR + \sum_{j \in \mathcal{N}_i} U_{ij}(\|\mathbf{r}_{ij}\|)
$$

We use this potential function to define a global controller that relies on communication among all agents:  

$$
\mathbf{u}_i^\top = -\sum_{j=1}^J (\mathbf{v}_j - \mathbf{v}_i) - \sum_{j=1}^J \nabla \mathbf{r}_j U_{ij}
$$

Our goal will be to learn to approximate the global controller using only local information.

We can compare the global controller to a local controller that uses only information from neighbors $\mathcal{N}_i$ of node $i$:  

$$
\mathbf{u}_i^\top = -\sum_{j \in \mathcal{N}_i} (\mathbf{v}_j - \mathbf{v}_i) - \sum_{j \in \mathcal{N}_i} \nabla \mathbf{r}_j U_{ij}
$$

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 potential function-based controllers. Rather than directly using the state $[\mathbf{r}_i, \mathbf{v}_i]$ of each node $i$ during aggregation, we design the relevant features needed to replicate the non-linear controller using only a linear aggregation operation, where $[\mathbf{x}_n]_i \in \mathbb{R}^6$:  

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

This observation vector is then used during aggregation as described in (9)-(10). The local controller also requires the computation of (24), so we are not giving the GNN an unfair advantage by providing the instantaneous measurements of neighbors’ states.

We quantify the costs of the observed trajectories by the variance in velocities, where $\bar{\mathbf{v}}$ is the mean velocity among all agents:  

$$
c(\mathbf{r}_i, \mathbf{v}_i) = \frac{1}{N} \sum_{j=1}^J \|\mathbf{v}_j - \bar{\mathbf{v}}\|^2
$$

The variance in velocities measures how far the system is from consensus in velocities [31].

C. Methods and Results

We explore flocking as an application of the GNN controller to demonstrate the effect of aggregation in a rapidly changing network. We also explore the transfer of policies trained on point-masses to testing in the AirSim simulator.

1) Experiment Parameters: For the point-mass system, a flock of 30 agents was used, with initial positions sampled uniformly within a radius of 6.0 of the origin and initial velocities sampled uniformly on the range $[-3,3]$. The default sampling time and aggregation filter length were chosen to be $T_s = 0.01$ and $K = 3$. Agents were able to communicate within a radius of $R = 1.0$. The GNN architecture was modified to accommodate for the vector of six features per agent. The input layer had 6K neurons and the hidden layer had 12K neurons. For training, 600 trajectories of length 200 were used. For testing, 50 trajectories were used.

2) Aggregation for Flocking: We demonstrate the effect of changing the aggregation filter length on the performance of the Aggregation GNN controller for flocking (Fig. 5) using the global and local controllers as benchmarks. The reported cost values were computed by taking the median over 50 test trajectories for each of 10 separately trained models. The error bound is one half of the interquartile range (IQR). The median was used because random initializations of the flock produce a large number of outliers. We observe that all GNN controllers outperform the local controller, but fall behind the global controller. We hypothesize that the GNN for $K = 1$, which has access to the exact same information as the local controller, outperforms the local controller due to the GNN’s model capacity, including the ability to use weights other than $\pm 1$, and the non-linear activation. These capabilities may allow
the controller to memorize the steady state configuration and reach it more quickly with the same information.

3) Testing in AirSim: Next, we evaluate the performance of GNN controllers trained on the point-mass flocking problem in testing on 50 drones in the AirSim simulator. The GNN controller that was trained on point-masses with ideal dynamics was tested in the presence of latency of the dynamics and observations of a real drone. The absolute locations and velocities of the drones were scaled by a factor of $C = 6$ before evaluation of the GNN so that the optimal spacing dictated by the potential function does not result in collisions. The API provided by AirSim allows us to send velocity commands to the drones in the flock, but the trained GNN model produces acceleration outputs. If the current velocity of drone $i$ is given by $v_{i,n}$, we compute the next velocity command as follows:

$$v_{i,n+1} = v_{i,n} + t_{i} C \hat{\pi}(\frac{1}{C} z_{i,n}, H)$$  \hspace{1cm} (26)

The altitude of the drones was fixed to be 40 meters, so that we could apply the learned two-dimensional flocking controller. The mean time interval of the control loop was measured to be 133ms, corresponding to an average control and communication frequency of 7.5 Hz.

We test our algorithm on a task in which two flocks of 25 quadrotorls are moving towards each other with equal and opposite velocities of 1.8 m/s. The desired flocking behavior is pictured in Figure 6 showing a regular spacing of 0.5 m between drones, with aligned velocities. The results of the global and local controllers are provided as benchmarks. The mean cost is reported over 20 trajectories of 300 steps, with a 1 standard deviation error bound. Aggregation for $K = 2$ provides the best results out of the non-centralized controllers (Fig. 7). We believe that the $K = 3$ controller may be struggling with the increased latency between subsequent observations in the simulation as compared to the point-mass system, and therefore performs worse than the controller for $K = 2$.

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 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.

VI. 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 networked linear systems and learning flocking behaviors confirms the value of multi-hop information to performance and robustness to varying control rates and degree of the communication graph.

In the future, we plan to train similar agent architectures in a reinforcement learning setting in order to improve the quality of the learned controllers as well as address tasks where no optimal global policy is available. While these learned controllers can operate on dynamically changing communication graphs, the flocking experiments illustrate that a loss of connectivity is not always recoverable. In future work, enforcing state or input constraints could help avoid these failure modes. Finally, we would like to apply this same architecture to large heterogeneous teams of robots.
REFERENCES

[1] Joseph L Baxter, EK Burke, Jonathan M Garibaldi, and Mark Norman. Multi-robot search and rescue: A potential field based approach. In Autonomous robots and agents, pages 9–16. Springer, 2007.

[2] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. Automatica, 38(1):3–20, 2002.

[3] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and deep locally connected networks on graphs. arXiv:1312.6203v3 [cs.LG], 21 May 2014. URL http://arxiv.org/abs/1312.6203

[4] 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.

[5] 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.

[6] Ceyhun Eksin, Pooya Molavi, Alejandro Ribeiro, and Ali Jadbabaie. Bayesian quadratic network game filters. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, pages 4589–4593. IEEE, 2013.

[7] G.F. Franklin, J.D. Powell, and M.L. Workman. Digital Control of Dynamic Systems, Second Edition. Addison-Wesley, 1980.

[8] 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.

[9] 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.

[10] F. Gama, A. G. Marques, A. Ribeiro, and G. Leus. Aggregation graph neural networks. In 44th IEEE Int. Conf. Acoust., Speech and Signal Process., Brighton, UK, 12-17 May 2019. IEEE.

[11] Alessandro Giusti, Jérôme Guzzi, Dan C Ciresan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni 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.

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

[13] James S Jennings, Greg Whelan, and William 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.

[14] 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.

[15] Mathew Penrose et al. Random geometric graphs. Number 5. Oxford university press, 2003.

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

[17] M. Rotkowitz and S. Lall. A characterization of convex problems in decentralized control. IEEE Transactions on Automatic Control, 50(12):1984–1996, Dec. 2005. doi: 10.1109/TAC.2005.860365.

[18] 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

[19] 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.

[20] Shital Shah, Debadeepa Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Field and Service Robotics, 2017. URL https://arxiv.org/abs/1705.05065

[21] Vishal Sharma, Mehdi Bennis, and Rajesh Kumar. UAV-assisted heterogeneous networks for capacity enhancement. IEEE Communications Letters, 20(6):1207–1210, 2016.

[22] D. I Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Processing Mag., 30(3):83–98, May 2013.

[23] Andrea Tacchetti, H Francis Song, Pedro AM Mediano, Vinicius Zambaldi, Neil C Rabinowitz, Thore Graepel, Matthew Botvinick, and Peter W Battaglia. Relational forward models for multi-agent learning. arXiv preprint arXiv:1809.11044, 2018.

[24] Herbert G Tanner. Flocking with obstacle avoidance in 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] Sebastian Thrun and Yufeng Liu. Multi-robot slam with sparse extended information filers. In Robotics Research. The Eleventh International Symposium, pages 254–266. Springer, 2005.

[26] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on, volume 1, pages 321–328. IEEE, 2000.

[27] YuhShyang Wang, Nikolai Matni, and John C Doyle. Separable and localized system level synthesis for large-

[28] Mathew Penrose et al. Random geometric graphs. Number 5. Oxford university press, 2003.

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

[30] M. Rotkowitz and S. Lall. A characterization of convex problems in decentralized control. IEEE Transactions on Automatic Control, 50(12):1984–1996, Dec. 2005. doi: 10.1109/TAC.2005.860365.

[31] 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

[32] 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.

[33] Shital Shah, Debadeepa Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Field and Service Robotics, 2017. URL https://arxiv.org/abs/1705.05065

[34] Vishal Sharma, Mehdi Bennis, and Rajesh Kumar. UAV-assisted heterogeneous networks for capacity enhancement. IEEE Communications Letters, 20(6):1207–1210, 2016.

[35] D. I Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Processing Mag., 30(3):83–98, May 2013.

[36] Andrea Tacchetti, H Francis Song, Pedro AM Mediano, Vinicius Zambaldi, Neil C Rabinowitz, Thore Graepel, Matthew Botvinick, and Peter W Battaglia. Relational forward models for multi-agent learning. arXiv preprint arXiv:1809.11044, 2018.

[37] Herbert 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.

[38] Herbert G Tanner, Ali Jadbabaie, and George 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.

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

[40] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on, volume 1, pages 321–328. IEEE, 2000.
[29] YuhShyang Wang, Nikolai Matni, and John C Doyle. A system level approach to controller synthesis. *IEEE Transactions on Automatic Control*, 2019.

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

[31] Lin Xiao, Stephen Boyd, and Seung-Jean Kim. Distributed average consensus with least-mean-square deviation. *Journal of parallel and distributed computing*, 67(1):33–46, 2007.

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

[33] Kemin Zhou, John Comstock Doyle, Keith Glover, et al. *Robust and optimal control*, volume 40. Prentice hall New Jersey, 1996.