On the Stability of Gated Graph Neural Networks

Antonio Marino, Claudio Pacchierotti, Paolo Robuffo Giordano

On Stability of Gated Graph Neural Networks

Antonio Marino[1], Claudio Pacchierotti[2], Paolo Robuffo Giordano[2]

***Abstract*— In this paper, we aim to find the conditions for input-state stability (ISS) and incremental input-state stability (δISS) of Gated Graph Neural Networks (GGNNs). We show that this recurrent version of Graph Neural Networks (GNNs) can be expressed as a dynamical distributed system and, as a consequence, can be analysed using model-based techniques to assess its stability and robustness properties. Then, the stability criteria found can be exploited as constraints during the training process to enforce the internal stability of the neural network. Two distributed control examples, flocking and multi-robot motion control, show that using these conditions increases the performance and robustness of the gated GNNs.**

***Index Terms*— distributed control, graph neural network, stability analysis**

I. INTRODUCTION

MULTI-agent systems have been successfully studied in the past few years [1]. With respect to single-agent approaches, coordinated multi-agent systems are expected to collaboratively solve tasks and offer more flexibility, all features that make these systems suited to solve problems in a variety of disciplines including computer science, electrical engineering, and robotics [2]. The collaborative control of multiple agents must take into account the needs of the group. For multi-robot applications, individual robot motion should be generated by using not only local sensing data, but also knowledge about the group state, usually retrieved through communication with a limited number of (neighboring) team members [3]. Hence, communication is one of the key elements to realize distributed solution in multi-agent systems.

In the last decade, the control community has widely adopted neural networks in data-driven control applications, taking advantage of their superior approximation capabilities [4]. In distributed control, neural networks are particularly convenient since they can approximate distributed policies without the need of cumbersome optimizations and designs. In the literature, we can find multiple examples of data driven approaches especially based on reinforcement learning [5], [6] fed by input data such as images [7]. However, these approaches work on local sensing data without communicating with the other agents. The communication is used in distributed machine learning where the learning process is partitioned on several machines that contribute to the group knowledge [8]. On the contrary, in data-driven distributed control, a recent trend involves using Graph Neural Networks (GNNs) to encode distributed control and planing solutions.

GNNs perform prediction and analysis on graphs, a convenient topological representation in different kinds of problems like text classification, protein interface predictions, and social network decisions [9]. For these latter, GNNs are more effective than the classical neural network architectures [10]. At the same time, they gave a new perspective in realization of distributed control.

Gama et al. [11] use GNNs to define a distributed LQR controller, casting the linear-quadratic problem as a self-supervised learning to find the best GNN-based distributed control, thanks to the native GNN distributed nature. The same authors [12] develop a GNN-based flocking control deployed on large team scale. Further examples can be found in space coverage [13], multi robots path [14] and motion planning [15] also in obstacles rich environment GNN [16]. GNNs are used also in approaches for enhancing multi agents perception [17] or performing a distributed active information acquisition [18] translating the multi-robot information gathering problem to a graph representation and formulate GNN-based decision maker.

In the recent literature, a very relevant discussion is about making the data-driven based method robust and stable [19]. In this context, many works applied contraction analysis to demonstrate recurrent neural network stability [20], or directly closed-loop stability in continuous learning [21] and adaptive control [22]. Other works have attempted to formulate new neural network models for achieving closed-loop stability, such as [23] where the controller is obtained from an Hamiltonian Deep Neural Networks and the stability is guaranteed by the compositional properties of port-Hamiltonian systems. Recently, the authors in [24], [25] demonstrated the ISS and incremental ISS (δISS) [26] for LSTMs and GRUs, two of the most popular recurrent neural network models.

Inspired by these last results, the goal of this work is to characterize the δISS properties of the recurrent version of GNN, i.e. Gated Graph Neural Networks (GGNN) [27]. These models use gated mechanisms to deploy distributed recurrent models able to reason on temporal- and spatial-based relationships among the agents. The δISS property is a stronger property than plain ISS and leads to asymptotically convergence of two state trajectories when their respective input sequences are close, regardless of the initial conditions for the states. Therefore, we directly focus on the incremental stability results avoiding the complexity of other frameworks, e.g., contraction analysis.

To the best of our knowledge, this is the first time ISS and δISS are proven for GGNN, whereas previous works have focused on limited stability results like stability to graph perturbations [27], [28]. Instead, this letter considers

[1] is with Univ Rennes, CNRS, Inria, IRISA – Rennes, France. E-mail: antonio.marino@irisa.fr

[2] are with CNRS, Univ Rennes, Inria, IRISA – Rennes, France. E-mail: {claudio.pacchierotti,prg}@irisa.fr

This work was supported by the ANR-20-CHIA-0017 project "MULTI-SHARED"

the system internal stability to the input features in a more general dynamical system analysis. The conditions derived in this work are in the form of nonlinear inequalities on GGNN weights: these can be exploited to certify the stability of a trained neural model, or can be enforced as constraints during the training process to guarantee the stability of the GGNN.

The remaining sections of this paper are organized as follow. Section II reviews the preliminaries about graph neural networks and presents their recurrent versions, RGNNs and GGNNs. Section III and IV present the main stability results for the one-layer and deep GGNN. In Sect. V, we also provide a perspective of these results when considering communication delays. We then show two examples of the conditions found for flocking control, in Sect. VI, and multi robot motion control, in Sect. VII. Concluding remarks are finally presented in Sect. VIII.

II. PRELIMINARIES

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph where $\mathcal{V} = \{v_1, \dots, v_N\}$ is the vertex set (representing the N agents in the group) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set. Each edge $e_k = (i, j) \in \mathcal{E}$ is associated with a weight $w_{ij} \geq 0$ such that $w_{ij} > 0$ if the robots i and j can interact and $w_{ij} = 0$ otherwise. As usual, we denote with $\mathcal{N}_i = \{j \in \mathcal{V} | \ w_{ij} > 0\}$ the set of neighbors of robot i. We also let $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ be the adjacency matrix with entries given by the weights w_{ij}. Defining the degree matrix $\boldsymbol{D} = \text{diag}(d_i)$ with $d_i = \sum_{j \in \mathcal{N}_i} w_{ij}$, the Laplacian matrix of the graph is $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$.

The Laplacian matrix has a natural sparsity pattern that allows to implement distributed behaviors. Let $\boldsymbol{x} \in \mathbb{R}^N$ be a vector whose i-component x_i is assigned to robot i. This graph signal does not contain any information about the graph, but it can be processed using the graph Laplacian to incorporate graph topological information

$$\boldsymbol{\ell}_i \boldsymbol{x} = \sum_{j \in \mathcal{N}_i} w_{ji}(x_i - x_j), \tag{1}$$

where $\boldsymbol{\ell}_i$ is the i-th row of $\boldsymbol{L}$. This process is also known as *aggregation* in the graph signal processing literature. The signal manipulation can be operated by means of any support matrix $\boldsymbol{S}$, e.g., Laplacian, adjacency matrix, weighted Laplacian, and so forth, which respects the sparsity pattern of the graph. Such matrix is often called *graph shift operator*, because it implements a linear combination of signal values in neighboring nodes that each node v_i can locally access. The choice of the support matrix depends on the application and the type of the information to exchange. Later, in Sects. VI–VII, we will use the Laplacian as support matrix, commonly used in distributed control. However, the proposed techniques do not assume the use of a specific support matrix.

Performing k repeated applications of $\boldsymbol{S}$ on the same signal represents the aggregation of the k-hop neighborhood information. In analogy with traditional signal processing, this property can be used to define a linear graph filtering [29] that processes the multi features signal $\boldsymbol{x} \in \mathbb{R}^{N \times G}$ with G features:

$$H_{\boldsymbol{S}}(\boldsymbol{x}) = \sum_{k=0}^{K} \boldsymbol{S}^k \boldsymbol{x} \boldsymbol{H}_{\boldsymbol{k}}. \tag{2}$$

where the weights $\boldsymbol{H}_{\boldsymbol{k}} \in \mathbb{R}^{G \times F}$ define the output of the filter. Note that $\boldsymbol{S}^k = \boldsymbol{S}(\boldsymbol{S}^{k-1})$, so that it can be computed locally with repeated 1-hop communications between a node and its neighbors. Hence, the computation of $H_{\boldsymbol{S}}$ is distributed on each node.

A. Graph Neural Network

Although $H_{\boldsymbol{S}}$ is simple to evaluate, it can only represent a linear mapping between input and output filters. GNNs increase the expressiveness of the linear graph filters by means of pointwise nonlinearities $\rho : \mathbb{R}^{N \times F_{l-1}} \to \mathbb{R}^{N \times F_{l-1}}$ following a filter bank. Letting $H_{\boldsymbol{S}l}$ be a bank of $F_{l-1} \times F_l$ filters at layer l, the GNN layer is defined as

$$\boldsymbol{x}_l = \rho(H_{\boldsymbol{S}l}(\boldsymbol{x}_{l-1})), \qquad \boldsymbol{x}_{l-1} \in \mathbb{R}^{N \times F_{l-1}}. \tag{3}$$

Starting by $l = 0$ with F_0, the signal tensor $\boldsymbol{x}_{l_n} \in \mathbb{R}^{N \times F_{l_n}}$ is the output of a cascade of l_n GNN layers. This particular GNN is often called convolution graph network because each layer applies a graph signal convolution (2). GNNs inherit some interesting properties from graph filters, such as permutational equivariance [30] and their local and distributed nature, showing superior ability to process graph signals [18], [31], [32]. In particular, a property that is interesting in the majority of the applications is the prediction stability to perturbations of the graph support, i.e., perturbations of the graph supports will cause bounded output variations determined by the size of the support change. Finally, the GNN is stable to the graph perturbation if the filters frequency responses [33] are integral Lipschitz :

$$|h(\lambda_j) - h(\lambda_i)| \leq 2C \frac{|\lambda_j - \lambda_i|}{|\lambda_j + \lambda_i|} \tag{4}$$

where $\lambda_j, \lambda_i \in \mathbb{R}$ are any support matrix eigenvalues, and $C > 0$ a proper integral Lipschitz constant. This condition restricts the graph frequency response variability to the midpoint of λ variations.

B. Recurrent Graph Neural Network

In some cases, a recurrent autoregressive model of GNN is more expressive and powerful [34], for instance in traffic forecasting [35] and, in general, for all the problems that show a clear time dependency. Analogously to RNN, their well-known centralized counterpart, *graph recurrent neural network* (GRNN) are systems which exploit memory to learn patterns in sequences of data which, in the dynamic system case, can be observations of the world or measurements of the outputs of a system to be identified. However, in the case of RNNs, the number of parameters still depends on the number of agents preventing RNNs from scaling to inputs with large dimensions. More importantly, this issue hinders their ability to account for other structures inherent to the data. Instead, GRNNs account for the graph topology by constraining the data processing and shrinking the regression space. Moreover, they allow leveraging the graph structure to find repetitive and symmetrical feature patterns.

In a GRNN, the state is updated based on an input tensor $\boldsymbol{u} \in \mathbb{R}^{N \times G}$ and on the current neural network state $\boldsymbol{x} \in \mathbb{R}^{N \times F}$, being itself a signal tensor. A state-space representation of the GRNN is given by:

$$\boldsymbol{x}^+ = \rho(A_{\boldsymbol{S}}(\boldsymbol{x}) + B_{\boldsymbol{S}}(\boldsymbol{u}) + \boldsymbol{1}_N \otimes \boldsymbol{b})$$

where A_S, B_S are respectively the state and the input graph filters, $\boldsymbol{b} \in \mathbb{R}^F$ is the bias and $\mathbf{1}_N$ is the unitary vector of dimension N. The bias is equal for all the agents in the network. This structure allows the computation of $\boldsymbol{x}$ to be performed in an entirely local fashion, involving only repeated exchanges with the one-hop neighbors of each node. The state $\boldsymbol{x}$ represents the state of a dynamic controller/observer of which the GRNN is an approximation.

C. Gated Graph Neural Network

Traditional RNNs suffer of the vanishing gradient problems rising from long sequences dependencies which require deep models [36], [37]. The same issues arise in the graph counterpart with long temporal sequence. However, with GRNN one also has to deal with a form of vanishing gradient in space where some nodes or paths of the graph might get assigned more importance than others in long range exchanges (temporal and spatial), leading to imbalances in the graph encoding of the information. This problem is connected to the Laplacian over smoothing which accumulates in time on the state temporal sequence.

For example, in a graph with highly connected subgraphs, the aggregation (1) will weight more the components of such subgraphs which in turn will gain ever greater importance on the state, dominating the other components.

Forgetting factors can be applied to mitigate this problem reducing the influence of past or new signal on the state. A Gated Graph Neural Network (GGNN) [27] is a recurrent Graph Neural Network that uses a gating mechanism to control how the past information influences the update of the GNN states. As before, GGNNs admit the following state-space representation [38]

$$\begin{cases} \tilde{\boldsymbol{q}} = \sigma(\tilde{A}_S(\boldsymbol{x}) + \tilde{B}_S(\boldsymbol{u}) + \hat{b}) \\ \hat{\boldsymbol{q}} = \sigma(\hat{A}_S(\boldsymbol{x}) + \hat{B}_S(\boldsymbol{u}) + \tilde{b}) \\ \boldsymbol{x}^+ = \sigma_c(\hat{\boldsymbol{q}} \circ A_S(\boldsymbol{x}) + \tilde{\boldsymbol{q}} \circ B_S(\boldsymbol{u}) + b) \end{cases} \tag{5}$$

with $\sigma = \frac{1}{1+e^{-x}}$ being the logistic function, and σ_c the hyperbolic tangent. $\hat{A}_S, \hat{B}_S$ are graph filters of the forgetting gate, $\tilde{A}_S, \tilde{B}_S$ are graph filters (2) of the input gate, and A_S and B_S are the state graph filters (2). $\hat{\boldsymbol{q}}, \tilde{\boldsymbol{q}} \in Q \subseteq [0,1]^{N \times F}$ are the state and the output gates multiplied via the Hadamard product $\circ$ by the state and the inputs of the network, respectively. $\hat{\boldsymbol{b}}.\tilde{\boldsymbol{b}}, \boldsymbol{b} \in \mathbb{R}^{N \times F}$ are respectively the biases of the gates and the state built as $\mathbf{1}_N \otimes b$ with the same bias for every agents. We note that, in the literature, there exist several Gated GNNs structures [9], [27], [38] sharing gating mechanisms of different kinds (temporal, attention, and so on). However, the system in (5) generalizes well the features of these GNNs and provides a good level of abstraction for analyzing its stability. Indeed, we want to focus on the stability properties of such neural models when a gate mechanism is combined with a distributed computation.

D. Incremental Input State Stability

In this section, we recall the definition of ISS and δISS that will be used throughout the paper. Recalling the definitions of $\mathcal{KL}$, $\mathcal{K}_\infty$ functions [39] and the infinite norm $||\cdot||_\infty$, the following definition of ISS is given

Definition II.1 (ISS). *System (5) is called input-to-state stable if there exist functions $\beta \in \mathcal{KL}$ and $\gamma \in \mathcal{K}_\infty$ such that, for any $t \in \mathbb{Z} \geq 0$, any initial state $\boldsymbol{x}(0) \in \mathcal{X}$ any input sequence $\boldsymbol{u} \in \mathcal{U}$ it holds that:*

$$||\boldsymbol{x}(t)||_\infty \leq \beta(||\boldsymbol{x}(0)||_\infty, t) + \gamma_u(||\boldsymbol{u}||_\infty) + \gamma_b(||\boldsymbol{b}||_\infty) \tag{6}$$

A further desirable property is incremental ISS (δISS) [40]. The δISS property ensures that any pair of state trajectories converge towards each other even if they start from different initial conditions. Moreover, their difference is bounded only by the differences of their inputs (i.e. for example an ideal control corrupted by an additive noise), thus enhancing the system robustness [41].

Definition II.2 (δISS). *System (5) is is called incrementally input-to-state stable [26] if there exist functions $\beta_\delta \in \mathcal{KL}$ and $\gamma_\delta \in \mathcal{K}_\infty$ such that, for any $t \in \mathbb{Z} \geq 0$, any initial states $\boldsymbol{x}(0)_1, \boldsymbol{x}(0)_2 \in \mathcal{X}$ any input sequences $\boldsymbol{u_1}, \boldsymbol{u_2} \in \mathcal{U}$ it holds that:*

$$\begin{aligned} ||\boldsymbol{x}(t)_1 - \boldsymbol{x}(t)_2||_\infty \leq & \beta_\delta(||\boldsymbol{x}(0)_1 - \boldsymbol{x}(0)_2||_\infty, t) \\ & + \gamma_\delta(||\boldsymbol{u}_1 - \boldsymbol{u}_2||_\infty) \end{aligned} \tag{7}$$

Remark 1. *In the neural network context, the δISS property ensures that any difference in the initial conditions will be eventually discarded, and thus the same outputs will correspond to the same observations. Moreover, since the stability is valid for $t > 0$, for a training with a finite time sequence dataset it is guaranteed that all the NN state trajectories converge to a unique solution.*

III. ONE-LAYER GGNN STABILITY

In this section we will discuss the stability properties of a single layer GGNN. For the rest of the paper the following assumption will be made

Assumption 1. *The input $\boldsymbol{u}$ is unity-bounded: $\boldsymbol{u} \in \mathcal{U} \subseteq [-1,1]^{N \times G}$, i.e. $||\boldsymbol{u}||_\infty \leq 1$.*

This is a quite mild assumption since the input signal is usually normalized or it is the result of others network layers with unitary output activation functions.

Before stating the sufficient conditions for the ISS of GGNN, we will first introduce the notation for the following quantities

$$\begin{aligned} & & S_K &\triangleq [I, S, \dots, S^K] \\ A &\triangleq [A_0, \dots, A_K]^T & B &\triangleq [B_0, \dots, B_K]^T \\ \tilde{A} &\triangleq [\tilde{A}_0, \dots, \tilde{A}_K]^T & \hat{A} &\triangleq [\hat{A}_0, \dots, \hat{A}_K]^T \\ \tilde{B} &\triangleq [\tilde{B}_0, \dots, \tilde{B}_K]^T & \hat{B} &\triangleq [\hat{B}_0, \dots, \hat{B}_K]^T \end{aligned} \tag{8}$$

where K is the filters length. The i-th component of the gates is considered as one feature for j-th robot. Then, in light of assumption (1) and knowing that $||\boldsymbol{x}|| \leq 1$, we have

$$\begin{aligned} |\hat{q}_i| &\leq ||\hat{\boldsymbol{q}}||_\infty \max_{u \in \mathcal{U}, x \in \mathcal{X}} ||\sigma(\hat{A}_S(\boldsymbol{x}) + \hat{B}_S(\boldsymbol{u})) + \hat{b}||_\infty \\ &\leq || \max_{u \in U, x \in \mathcal{X}} \sigma(\hat{A}_S(\boldsymbol{x}) + \hat{B}_S(\boldsymbol{u})) + \hat{b}||_\infty \\ &\leq \sigma(|| \max_{u \in U, x \in \mathcal{X}} \hat{A}_S(\boldsymbol{x}) + \hat{B}_S(\boldsymbol{u}) + \hat{b}||_\infty) \\ &\leq \sigma(||\hat{A}_S(\boldsymbol{x}_{max}) + \hat{B}_S(\boldsymbol{u}_{max}) + \hat{b}||_\infty) \end{aligned} \tag{9}$$

where $\boldsymbol{x}_{max}$ and $\boldsymbol{u}_{max}$ are the maximum values of $\boldsymbol{x}$ and $\boldsymbol{u}$. Recalling that a graph filter in equation (2) can be written

as $\hat{H}_S(\boldsymbol{x}) = [I, S, \dots, S^K](I_K \otimes \boldsymbol{x})[\hat{H}_0, \hat{H}_1, \dots, \hat{H}_K]^T$ and using notation (8), equation (9) becomes

$$\begin{aligned} |\hat{q}_i| &\leq \sigma(||S_K||_\infty(||\hat{A}||_\infty||\boldsymbol{x}||_\infty + ||\hat{B}||_\infty||\boldsymbol{u}||_\infty) + ||\hat{\boldsymbol{b}}||_\infty) \\ &= \sigma(||S_K||_\infty(||\hat{A}||_\infty + ||\hat{B}||_\infty) + ||\hat{\boldsymbol{b}}||_\infty) \triangleq \sigma_{\hat{q}}. \end{aligned} \tag{10}$$

We identify the induced ∞-norm as $||\cdot||_\infty$. Similarly for $\tilde{q}_i$

$$|\tilde{q}_i| \leq \sigma(||S_K||_\infty(||\tilde{A}||_\infty + ||\tilde{B}||_\infty) + ||\tilde{\boldsymbol{b}}||_\infty) \triangleq \sigma_{\tilde{q}}. \tag{11}$$

Theorem 1. *A sufficient condition for the ISS of a single-layer GGNN network is that $\mathcal{A} \leq 1$, where*

$$\mathcal{A} \triangleq \sigma_{\hat{q}}||S_K||_\infty||A||_\infty. \tag{12}$$

The proof based on the results for GRU and LSTM [24], [25] is provided in the Appendix I-A.

The δISS stability requires to analyze the evolution of the maximum distance of two states trajectories $\boldsymbol{x_1}, \boldsymbol{x_2}$, starting from two different initial condition $\boldsymbol{x}_1(0), \boldsymbol{x}_2(0)$ and having two different inputs $\boldsymbol{u_1}, \boldsymbol{u_2}$. We also need another assumption

Assumption 2. *Given two support matrixes $||S_1(t)||_\infty, ||S_2(t)||_\infty, \forall t \in \mathbb{Z}^+$ associated with two different graphs, they are bounded by the same $||\bar{S}||_\infty$.*

Theorem 2. *Under the assumption 2, a sufficient condition for the system (5) to be δISS is $\mathcal{A}_\delta \leq 1$; where*

$$\begin{aligned} \mathcal{A}_\delta \triangleq\ & \sigma_{\hat{q}}||\bar{S}_K||_\infty||A||_\infty + \frac{1}{4}||\bar{S}_K||^2_\infty||\hat{A}||_\infty||A||_\infty \\ & + \frac{1}{4}||\bar{S}_K||^2_\infty||\tilde{A}||_\infty||B||_\infty. \end{aligned} \tag{13}$$

The proof to this theorem is reported in Appendix I-B. Assumption 2 is reasonable for multi-agent systems, since graphs always have a finite number of agents with finite number of links between each others. When we deal with scalability and dynamic graphs, the assumption may be restrictive based on the choice of the support matrix. For examples, by using the adjacency matrix the upper bound of its norm would be the maximum number of the links for one robot, i.e. N (the team size). Therefore, training the GGNN using the adjacency matrix and being $||\bar{S}||_\infty = N$, the stability condition would be respected for teams with a maximum number of links for each agent up to N. For group with $N' > N$ we can guarantee stability if the agents have a number of neighbors less than N.

Remark 2. *In practice we can solve this issue by constraining, at runtime, the cardinality of $\#\mathcal{N}_i < N$ for every agent i in the team.*

Remark 3. *For a GGNN that uses normalized support matrices, e.g., normalized Laplacian, the assumption is met without any further restriction on the graph topology. Moreover the use of normalized support reduces the regularizing term in the loss function at training time. In the experiments we will use normalized Laplacian to compare stable and unstable neural network.*

For the reader's convenience, in the following, we will denote the neural network using the stable GGNN as sGGNN.

IV. DEEP GGNN STABILITY

With the word "deep" we refer to the stack of multiple layers of the Neural Networks. It is natural to ask whether the stability properties extend from the single layer to a multi-layer structure, commonly used in practice. We image the interconnection among layers by feeding the future state of one layer to the next one

$$\begin{cases} \tilde{\boldsymbol{q}}^i = \sigma(\tilde{A}_S(\boldsymbol{x}^i + \tilde{B}_S(\boldsymbol{u}^i) + \hat{\boldsymbol{b}}) \\ \hat{\boldsymbol{q}}^i = \sigma(\hat{A}_S(\boldsymbol{x}^i + \hat{B}_S(\boldsymbol{u}^i) + \tilde{\boldsymbol{b}})) \\ \boldsymbol{x}^{i+} = \sigma_c(\hat{q}^i \circ A_S(\boldsymbol{x}^i) + \tilde{q}^i \circ B_S(\boldsymbol{u}^i) + \boldsymbol{b}) \\ \boldsymbol{u}^i = \boldsymbol{x}^{i-1+}, \quad \boldsymbol{u}^1 = \boldsymbol{u} \end{cases} \tag{14a}$$

for all the layer $i \in \{1, \dots, M\}$. The output of the network results from a graph output filter

$$\boldsymbol{y} = Y_S(\boldsymbol{x}^M) + \boldsymbol{b_y}. \tag{14b}$$

The following then holds

Theorem 3. *The GGNN network is ISS if $\mathcal{A}^i \leq 1$ for every layer $i \in \{1, \dots, M\}$, where $\mathcal{A}^i$ are defined like in Theorem 1*

Proof. The deep GGNN (14) can be considered as a cascade of subsystems, so it is ISS if every subsystem is ISS. □

The δISS condition is more complex since there does not exist, to our knowledge, a general study in the literature for a cascade of δISS systems. However, in this case we can state

Theorem 4. *The deep GGNN is δISS stable if $\mathcal{A}^i_\delta \leq 1$ for every layer $i \in \{1, \dots, M\}$, where $\mathcal{A}^i$ are defined like in Theorem 2.*

The proof is left in the Appendix I-C.

V. GGNN STABILITY UNDER COMMUNICATION DELAY

The previous scheme of GGNN can represent any dynamics on a graph. However, it does not take into account the communication steps and the possible delay in the application of the supported matrix S. In each communication step, performed with sampling time T, we considered that the nodes communicate the data used in the graph filters and, at each T, we obtain the output of a GGNN layer computed on the data $[x(t-K), x(t-(K-1)), \dots, x(t)]$. We explicit the delayed system in (5) as:

$$\begin{cases} \tilde{\boldsymbol{q}} = \sigma(\tilde{A}_{S^t}(\boldsymbol{x}(t-K)) + \tilde{B}_{S^t}(\boldsymbol{u}(t-K)) + \tilde{\boldsymbol{b}}) \\ \hat{\boldsymbol{q}} = \sigma(\hat{A}_{S^t}(\boldsymbol{x}(t-K)) + \hat{B}_{S^t}(\boldsymbol{u}(t-K)) + \hat{\boldsymbol{b}}) \\ \boldsymbol{x}(t+1) = \sigma_c(\hat{q} \circ A_{S^t}(\boldsymbol{x}(t-K)) + \tilde{q} \circ B_{S^t}(\boldsymbol{u}(t-K)) + \boldsymbol{b}) \\ \boldsymbol{y} = Y_{S^t}(\boldsymbol{x}(t-K) + \boldsymbol{b_y}) \end{cases} \tag{15}$$

where the delay of K communication steps can be represented by a delay between the input, old of K steps, and the updating rule of the state. Moreover, we must consider graph filters with dynamic support matrices, i.e. matrices that change between each communication step. Thus, the graph filters considered

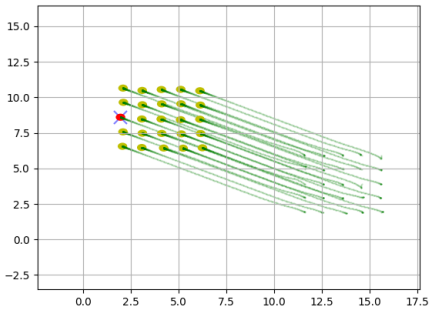


Fig. 1: **Flocking control**: a group of agents (yellow dots) move in order to reach the same velocity and to avoid collision. The leader (red dot) moves in order to reach the target (blue cross) and avoids the collision with the other agents.

so far become [42]

$$H_{S^t}(\boldsymbol{x}(t)) = \begin{bmatrix} I_N \\ \vdots \\ \prod_{\tau=t-(K-1)}^{t} S(\tau) \end{bmatrix}^T diag\left(\begin{bmatrix} \boldsymbol{x}(t) \\ \vdots \\ \boldsymbol{x}(t-K) \end{bmatrix} \right) \begin{bmatrix} H_0 \\ \vdots \\ H_K \end{bmatrix}$$

This filter expression is called unit-delayed filter [43]. Under Assumption 2, the infinite norm of the support matrices in time is upper bounded. Therefore, the formulation (15) allows us to conclude that the system remains δISS under the same conditions of the theorem 2.

Remark 4. *As noted in [43], the trajectories and the underlying graph observed at training time and the one observed at deployment are different, causing an increase of the training error. This is not an issue since, at a reasonable sampling time, the sequence of support matrices will not present drastic changes as their spectral characteristics are similar and the graph filters satisfy (4) with* $C = (||H||_\infty - 1)||\bar{S}||_\infty/2$*, and thus are stable to graph perturbations.*

VI. FLOCKING CONTROL EXAMPLE

In the following, we will show an application of the stable GGNN on a case study involving flocking control (Fig. 1) with a leader. In the problem of flocking, the agents are initialized to follow random velocities and the goal is to have them all fly at the same velocity while avoiding collisions with each other. Moreover, one of the agents takes the leader role conducting the team toward a target, unknown to the other agents. Flocking is a canonical problem in decentralized robotics [44], [45].

Dynamics and Expert Controller

We considered N agents described by the position $\boldsymbol{r}(t) \in \mathbb{R}^{N\times 2}$ and the velocity $\boldsymbol{v}(t) \in \mathbb{R}^{N\times 2}$ with a double integrator dynamics

$$\boldsymbol{r}(t+1) = \boldsymbol{r}(t) + T\boldsymbol{v}(t); \quad \boldsymbol{v}(t+1) = \boldsymbol{v}(t) + T\boldsymbol{u}(t);$$

with the discrete acceleration $\boldsymbol{u}(t) \in \mathbb{R}^{N\times 2}$ taken as system input. Note that the agent dynamics is used for building the dataset and for simulation purpose, but it is not provided to the learning algorithm.

Since the objective is to make all the agents reach the same velocity, the control must be tuned in order to minimize the following cost function

$$J(v(t)) = \frac{1}{N}\sum_{i=1}^{N} ||v_i(t) - \frac{1}{N}\sum_{j=1}^{N} v_j(t)||_2^2 \tag{16}$$

where v_i, v_j are the velocities for the agent i and j, respectively. The cost function measures the distance of the agent velocities from the average velocities of the team. The cost $J(v(t))$ under the control $\boldsymbol{u}(t)$ can be analysed on the time horizon T such that $t \in [0, T]$ to evaluate the convergence rate of the system. Moreover, the leader, randomly picked among the agents, does not follow the objective in (16) but rather it minimizes its distance from the target ($\boldsymbol{d}$). Therefore, the expert controller [42] for the followers is given by

$$\boldsymbol{u}_f(t) = -\boldsymbol{L}(t)\boldsymbol{v}(t) - \nabla_{\boldsymbol{r}} CA(\boldsymbol{r}(t), \boldsymbol{r}_j(t))|_{j=1\dots N} \tag{17a}$$

and for leader it is

$$\boldsymbol{u}_l(t) = -W_p(\boldsymbol{r}_l(t) - \boldsymbol{d}(t)) - \nabla_{\boldsymbol{r}_l} CA(\boldsymbol{r}_l(t), \boldsymbol{r}_j(t))|_{j=1\dots N} \tag{17b}$$

where W_p is a gain, $\boldsymbol{r}_l \in \mathbb{R}^2$ is the leader position, $\nabla_{\boldsymbol{r}} CA(\boldsymbol{r}(t), \boldsymbol{r}_j(t))/\nabla_{\boldsymbol{r}_l} CA(\boldsymbol{r}_l(t), \boldsymbol{r}_j(t))$ are the gradient of the collision avoidance potential with respect to the position of the agents/leader $\boldsymbol{r}/\boldsymbol{r}_l$, evaluated at the position $\boldsymbol{r}(t)/\boldsymbol{r}_l(t)$ and the position of every other agent $\boldsymbol{r}_j(t)$ at time t. The i-element of $\nabla_{\boldsymbol{r}} CA$ for each robot i with respect to robot j is given by [46]

$$\nabla_{\boldsymbol{r}_i} CA(\boldsymbol{r}_{ij}) = \begin{cases} -\frac{\boldsymbol{r}_{ij}}{||\boldsymbol{r}_{ij}||_2^4} - \frac{\boldsymbol{r}_{ij}}{||\boldsymbol{r}_{ij}||_2^2} & if ||\boldsymbol{r}_{ij}||_2^2 \leq R_{CA} \\ \boldsymbol{0} & otherwise \end{cases} \tag{18}$$

with $\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$ and $R_{CA} > 0$ indicating the minimum acceptable distance between agents. This potential function is a non negative, non smooth function that goes to infinity when the distance reduces and grows when the distance exceeds R_{CA}, in order to avoid the team losing the connectivity [46]. $\boldsymbol{u}_f(t), \boldsymbol{u}_l(t)$ are a centralized controller since computing them requires agent i to have instantaneous evaluation of $\boldsymbol{L}(t)\boldsymbol{v}(t)$ and $\boldsymbol{r}_j(t)$ of every other agent j in the team. R_{CA} and W_p are tunable parameters of the controllers.

A. Neural Network Architecture

We assume that the agents form a communication graph when they are in a sphere of radius R between each others and that exchanges occur at the sampling time T, so that the action clock and the communication clock coincide.

The input features vector $\boldsymbol{w}_i \in \mathbb{R}^{10}$ of the robot i for the designed neural network is

$$\boldsymbol{w}_i = \Bigg[\boldsymbol{v}_i, \sum_{j\in\mathcal{NS}_i} \frac{\boldsymbol{r}_{ij}}{||\boldsymbol{r}_{ij}||_2^4}, \sum_{j\in\mathcal{NS}_i} \frac{\boldsymbol{r}_{ij}}{||\boldsymbol{r}_{ij}||_2^2}, \{\boldsymbol{0}_2, \boldsymbol{r}_l - \boldsymbol{d}\}, \{[0,1],[1,0]\} \Bigg] \tag{19}$$

where $\mathcal{NS}_i$ is the set of the sensing agents within a sphere of radius R_{CA} centred in the robot i. Moreover, the vector contains the zero vector $\mathbf{0}_2 \in \mathbb{R}^{1\times 2}$ and the one-hot encoding $[0,1]$, if the agent is a follower while $(\boldsymbol{r}_l - \boldsymbol{d})$ and $[1,0]$, if the agent is a leader. We chose the one-hot encoding instead of the binary one, because it allows differentiating the neural network weights between the leader and the follower. Note that all the information in the vector $\boldsymbol{w}_i$ are locally available at the sampling/control time T.

The core of the neural network for the flocking control is a layer of GGNN with $F = 50$ features in the hidden state and filters length $K = 2$. Note that the choice of K affects the complexity of the stability condition imposed since it will constraint more parameters. The input features are first processed by a cascade of two fully connected layers of 128 nodes before feeding the graph neural network. A readout of two layers with 128 nodes combines the F-features GGNN hidden state to get the bidimensional control $\boldsymbol{u}$ saturated to the maximum admissible control. The input layers and the readout that encapsulate the GGNN shape a more realistic setting to test the stability of the graph neural network that is usually used in combination with other kinds of neural models. Following the Remark 3, we used normalized Laplacian as a support matrix

B. Training

We collected a dataset by recording 120 trajectories, further separated in three subsets of training, validation and test set using the proportion $70\% - 10\% - 20\%$, respectively. Each trajectory is generated by randomly positioning the agents in a square such that their inter-distance is between 0.6 m and 1.0 m and their initial velocities are picked at random from the interval $[-2,2]$ m/s in each direction. The leader is randomly selected among the agents and the target position is randomly located within a square of length 20 m centered at the location of the leader. Regardless of the target location, the trajectories have duration of 2.5 s and input saturation at 5 m/s^2. Moreover, the 120 trajectories are recorded with a random number of agents among $N = [4,6,10,12,15]$. We fixed the communication range to $R = 4$ m and the sensing to $R_{CA} = 1$ m. We trained the models for 120 epochs and executed the DAGGER algorithm [47] every 20 epochs. The algorithm evaluates the expert controller in (17) on the enrolled state trajectories applying the learnt control and adding them in the training set. Note that, thanks to the use of DAGGER, we do not need a large dataset. We solve the imitation learning problem using the ADAM algorithm [48] with learning rate $1e-3$ and forgetting factors 0.9 and 0.999. The loss function used for the imitation learning is the mean squared error between the output of the model and the optimal control action. The stability condition (2) is imposed as a regularization term on the cost function as in [24] with a weights of $\rho^+ = 1$ and $\rho^- = 0.01$.

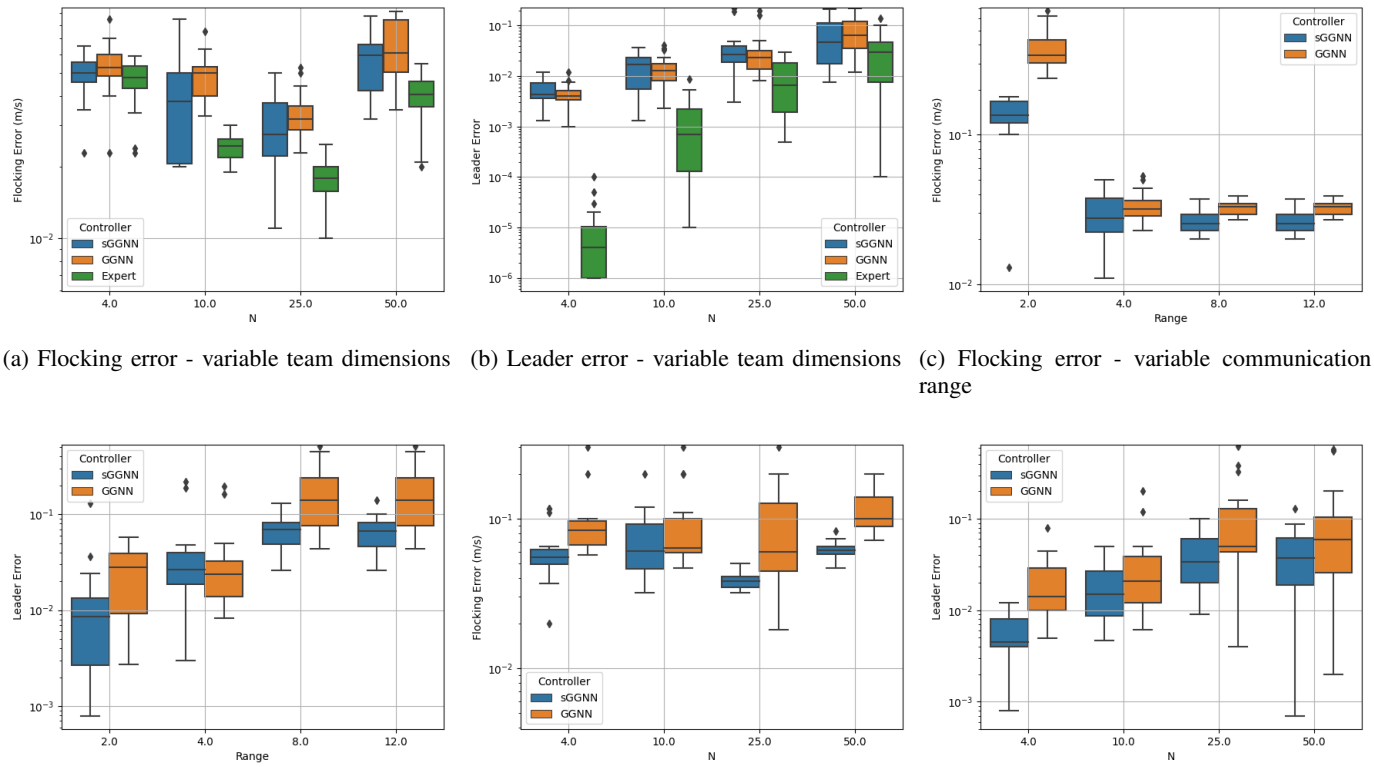
C. Results

In Fig. 2, we show a comparison between the stable GGNN (sGGNN) and non-stable GGNN controller for the flocking controller case. We evaluate the two controllers on 3 sets of experiments with 40 trajectories each. In the experiments, we vary team size, communication range and network delay to test the robustness of the controllers. Figures 2b, 2d and 2f report the leader position error evaluated after a fixed time of 2.5 s with respect to the leader starting location, i.e. e_f/e_s with e_f, e_s respectively being the final and the initial square distance of the leader from the target. Figures 2a, 2c and 2e show the average flocking error (16) in the interval $0 - 2.5$ s in logarithmic scale. Moreover, we consider a failure when the control leads to an agent-agent collision, the leader-target distance diverges, or any agent-agent distance diverges (i.e. the team splits).

In the first experiment, we evaluate the transferring at scale of the two controllers to variations of the team size N among $[4,10,25,50]$, while the communication range is fixed at $R = 4$ m without communication delay. The results, reported in Figs. 2a, 2b, show better performances for the sGGNN in achieving the flocking state as seen by the average error, that is generally closer to the expert controller than the non-stable one. When more agents are involved in the graph, the flocking error is low since, in both distributed and centralized controllers, the inter-agents collision avoidance constrains the motion of each agent by encouraging the velocity agreement. However, in the case of 50 agents, the leader struggles more to lead the entire team toward the target and opposing to the group motion resulting in higher flocking and leader errors for all the three controllers. In general, sGGNN and GGNN lead to similar leader errors, even if, on average, they are roughly 7%-12% smaller for the non-stable neural network at the cost of a higher flocking error.

In the second set of experiment, a general higher level of robustness is observed when the communication range differs from the one used at training time, $R = 4$ m, as can been seen in Figs. 2c, 2d where the range varies between $[2,4,8,10]$ and the team size is set at $N = 25$ without communication delay. Specifically, when the range is $R = 2$ m, the non-stable learnt control causes a flocking error of 0.40 m/s with outliers up to 0.68 m/s, greater than the average of 0.13 m/s of the stable control. Moreover, with GGNN, we experienced more group division and a consequent splitting of the team which explains the increase in the flocking error. However, in this condition, we also report the leader divergence and a consequent worst leader error with respect to the stable control of 27%. While the stable controller never results in failure situation, the non-stable GGNN causes 12% failures with range of $R = 2$ m. As expected, when the communication range increases the flocking errors decrease for both controllers, since they are able to communicate with more agents at the same time. However, in this case, the leader error increases, since the leader is often encapsulated by the other team agents and it is thus forced to follow them to not collide, causing a slower convergence to the target. Note that this behavior also affects the expert controller.

In the last experiment, we consider not instantaneous communication ($T = 0.01$ s) and evaluate again the transference at scale. With large team sizes, the non-stable controller fails 20%-35% of the experimented trajectories, while the stable one succeeded 100%. Figures 2e, 2f show consistency of the results with respect to the non-delay case, confirming



(a) Flocking error - variable team dimensions (b) Leader error - variable team dimensions (c) Flocking error - variable communication range

(d) Leader error - variable communication range (e) Flocking error - network with communication delay and variable team dimensions (f) Leader error - network with communication delay and variable team dimensions

Fig. 2: Flocking Error and Leader Error for stable and non-stable GGNN controllers evaluation varying the team size (with and without instantaneous communication) and the communication range, reported using box plots that display median, minimum,maximum, 25th/75th percentiles, and potential outliers.

that the controller remains stable even with not instantaneous communication.

In Fig. 3, we also reported the performances of the sGGNN and GGNN trained and tested with the Laplacian matrix to confirm what we stated in the Remark 3. As we can see, with a higher communication range than the one used at training time, the differences between the two controllers disappear except for differences caused by training errors.

VII. MULTI ROBOT MOTION CONTROL EXAMPLE

Another representative example is the distributed control of multi-robot team moving in a cluttered space (Fig. 4). This is a problem already studied in different works [7], [49], [50]. In particular, the proposed method is based on [51] that aims at solving the multi-robot path planning using GNN. However, while the method in [51] is based on discrete space and a discrete decision time, where each robot take a discrete directional decision for the next motion, here we consider smooth trajectories and continuous space. With this choice, we aim at controlling the robots with smooth inputs, a case that better shows the robustness and stability properties of the GGNN. The problem objective is to drive a group of robots starting from random positions toward their respective targets in a cluttered space.

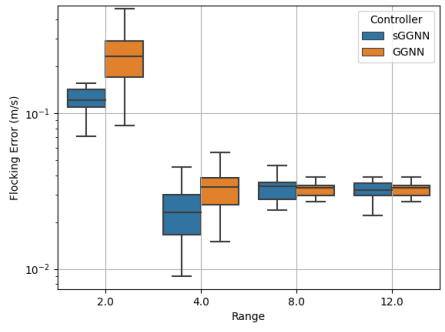
Dynamics and Expert Controller

We consider N agents described by the position $\boldsymbol{r}(t) \in \mathbb{R}^{N\times 2}$ in a single integrator dynamics with the velocity $\boldsymbol{u}(t) \in \mathbb{R}^{N\times 2}$ taken as the system input. As before, the learning algorithm is agnostic to the agent dynamics.

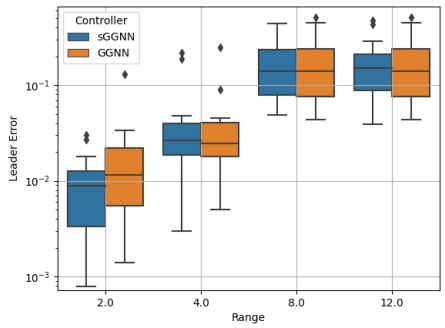
For the expert controller, we used a combination of RRT [52] to find an obstacle-free path and an MPC to control the agents in a continuous time. The MPC minimizes the divergence of the agents from the RRT-generated paths while constraining the agent motion in order to avoid inter-agent and obstacle collisions:

$$
\begin{aligned}
\min_{\boldsymbol{u}(t),\boldsymbol{r}(t)} \quad & \sum_{i=0}^{N} ||\boldsymbol{r}_{id}(t)||_2^2 \\
\text{s.t.} \quad & \dot{\boldsymbol{r}}_i = \boldsymbol{u}_i \; i = 1,\dots,N. \\
& g_o(\boldsymbol{r_{io}}(t)) \geq \boldsymbol{b}_o, \; i = 1,\dots,N. \\
& g_{ij}(\boldsymbol{r_{ij}}(t)) \geq \boldsymbol{b}_{ij}, \; i,j = 1,\dots,N. \\
& ||\boldsymbol{u}_i||_2 \leq 1 \; i = 1,\dots,N.
\end{aligned}
$$

where $\boldsymbol{r_{id}}$ is the distance from the RRT-path for the agent i, g_o, g_{ij} are the quadratic distances between the agent positions and the obstacles in the space ($\boldsymbol{r_{io}}$) and between the agents ($\boldsymbol{r_{ij}}$). As we can see, the MPC is naturally a centralized solution since it uses all the agent dynamics to generate the control inputs.



(a) Flocking error



(b) Leader error

Fig. 3: Flocking Error and Leader Error for stable and non-stable GGNN controllers with variable communication range using Laplacian

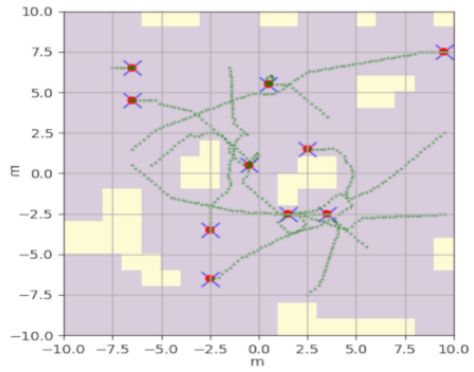


Fig. 4: **Multi Robot Motion Control**: a group of agents (red dots) move to reach their targets (blue cross) avoiding agent-agent and agent-obstacle (in yellow) collisions.

A. Neural Network Architecture

As in the previous example, we assume the agents to form a communication graph, as in the previous example, if they are within a communication radius of R. The input features vector $\boldsymbol{w}_i \in \mathbb{R}^{10}$ of the robot i for the designed neural network is

$$\boldsymbol{w}_i = \Bigg[\boldsymbol{r}_{id}, \sum_{o\in\mathcal{NS}_{io}} \frac{\boldsymbol{r}_{io}}{||\boldsymbol{r}_{io}||_2^4}, \sum_{o\in\mathcal{NS}_{io}} \frac{\boldsymbol{r}_{io}}{||\boldsymbol{r}_{io}||_2^2}, \sum_{j\in\mathcal{NS}_i} \frac{\boldsymbol{r}_{ij}}{||\boldsymbol{r}_{ij}||_2^4}, \sum_{j\in\mathcal{NS}_i} \frac{\boldsymbol{r}_{ij}}{||\boldsymbol{r}_{ij}||_2^2},\Bigg] \tag{20}$$

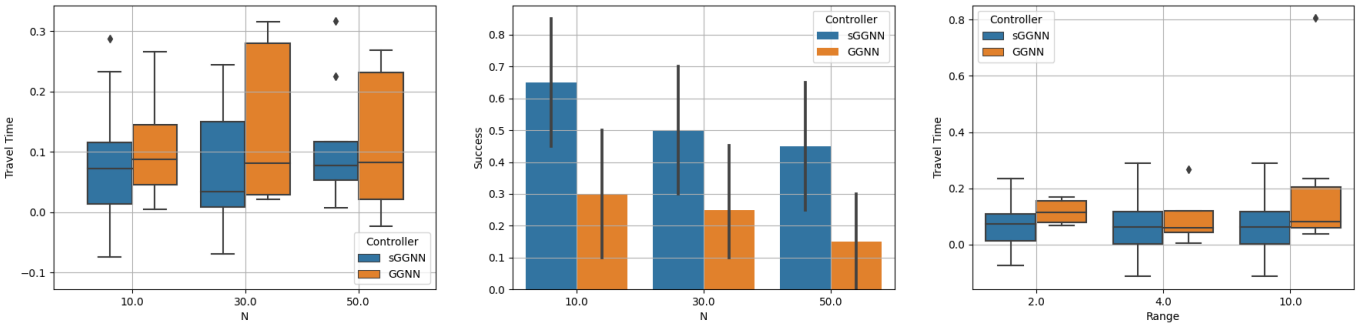
where $\mathcal{NS}_i, \mathcal{NS}_{io}$ are respectively the set of the agents and obstacles in the sensing range R_C of the robot i. At time T, all these information are available to the agent i.
Similarly to the previous case, we encapsulate the GGNN between 2 layers MLP input of 128 nodes and 2 layers readout of 128 nodes that give the agent velocity control. However, this time we adopt a 2 layers GGNN with $F = 30$ features in the hidden state and filter of length $K = 2$ to test the deep-GGNN stability. Here we used the normalized Laplacian as support matrix.

B. Training

We recorded 40 trajectories to build the dataset, further separated in three subsets of training, validation and test set using the proportion $70\% - 10\% - 20\%$, respectively. The trajectories are the results of RRT+MPC controller running with $N = 10$ agents randomly located in a square space of 20m $\times$20m with 15% obstacles density. The agent targets are equally randomly located in the free space. We fixed the communication range to $R = 4m$ and the sensing one to $R_C = 1m$. The training runs for 200 epochs with the DAGGER algorithm executed every 20 epochs. We used the ADAM algorithm with learning rate $1e-3$ and forgetting factors 0.9 and 0.999. The loss function used for the imitation learning is the mean squared error between the output of the model and the expert control action. We enforced the stability as in the previous example with weights of $\rho^+ = 1$ and $\rho^- = 0.01$.

C. Results

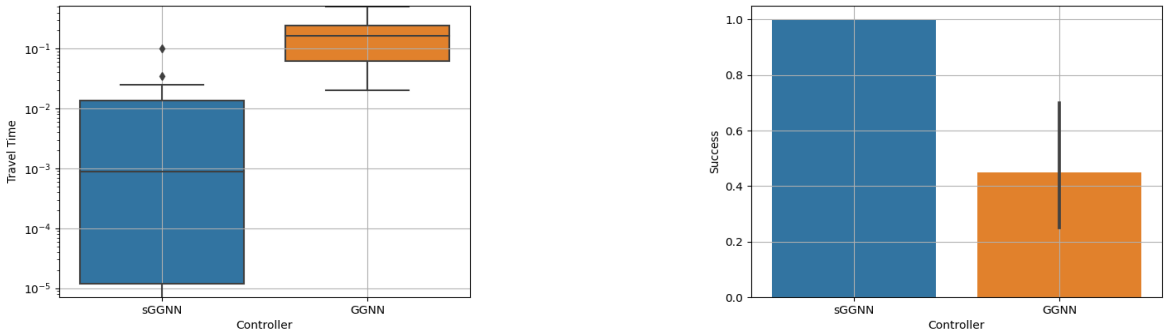
We evaluate the stability condition for the multi-robot motion control and report the results in Fig. 5. The comparison is carried out on three sets of experiments evaluating transfarable at scale, robustness on communication range and obstacle density. We recorded 40 trajectories for each case on a 20m $\times$20m map. For this application, we are primarily interested in the control success rate showed in Figs. 5b, 5d and 5f, where we consider successful the trajectories free of collisions and deadlocks. On successful trajectories, we also computed the travel time increase with respect to the expert controller. We obtained all the trajectories in the non instantaneus communication setting and, when not stated otherwise, with team size, the communication range and the obstacle density respectively of $N = 10, R = 4.0$m and 20%.
As in the flocking example, we first test the scalability of the two controllers to the team size (N) variations among $[10, 30, 50]$. sGGNN has a success rate that attests between 65% and 45% showing better performances compared to the GGNN one that is always below 30% as reported in 5b.
The travel time increase is comparable for the two control solutions as we can see in the Figs. 5a, 5c and 5e, even if the stable one presents more situations of negative travel time increasing. This latter may have values below zeros due to the presence of the RRT in the expert controller that finds an obstacle-free but not the shortest path. Hence GGNNs control can lead to trajectories that are faster since they work with the target location directly. This phenomenon is less evident when



(a) Travel time - variable team dimensions (b) Success rate - variable team dimensions (c) Travel time - variable communication range

(d) Success rate - variable communication range (e) Travel Time - variable obstacle density (f) Success rate - variable obstacle density

Fig. 5: Success rate and flow time for stable and non-stable GGNN controllers evaluation varying the team size, the communication range and the obstacles density for a $20m \times 20m$ map; the flow time increasing is computed as $(Tf - Tf^*)/Tf^*$ with Tf^* expert controller time of arrival.



(a) Flow Time - GGNN + rrt (b) Success rate - GGNN + rrt

Fig. 6: Success rate and flow time for stable and non-stable GGNN controllers following a precomputed rrt path to avoid the obstacles. The controllers run on the map of $20m \times 20m$ with 30% of randomly generated obstacles and 50 agents. The flow time increasing is computed as $(Tf - Tf^*)/Tf^*$ with Tf^* expert controller time of arrival.

the number of robots increases (Fig. 5a) or when the obstacle density increases (Fig. 5e).

The sGGNN does not show particular robustness to variations in the obstacle density since, as reported in Fig. 5f, the successful rate of 80% for 10% of obstacles drops to 45% for 30% of obstacles, the same success rate of the $N = 50$ and 20% of obstacles, even if it generalizes better than the non-stable learnt controller as confirmed by the higher success rate and the lower average flow time with respect to the non-stable controller. This suggests a poor obstacle avoidance behaviour, further demonstrated by the results in Fig. 6b where we isolate failures caused by agent-agent collision by testing a combination of gated GNN and RRT. For an obstacle density of 30% and 50 agents, we feed the RRT path to neural models projecting it on the sensing range by replacing the target location in the input layer (20). In this case, sGGNN+RRT reaches

100% success and an average travel time of 10e−3, confirming that sGGNN realizes a well distributed approximation of the MPC. On the contrary, GGNN+RRT still results in a high failure rate due to agent-agent collisions. One may find similar results varying the communication range as reported in Fig. 5d, where the sGGNN success rate remains unchanged. On the other hand, with GGNN on a lower communication range, we experienced more inter-agent collisions.

VIII. CONCLUSIONS

In this work, we devise sufficient conditions for the ISS and Incremental ISS of gated graph neural networks. When GGNN are used to learn distributed policies, the proposed stability conditions allow to guarantee that the trained networks enjoy the ISS/δISS property, which is particularly useful during the synthesis of distributed controllers. The proposed condition has been tested on the flocking control and multi robot motion control, showing good modeling performances. Results suggest that enforcing stability properties on the learnt controller makes it closer to the expert centralized one and more robust to parametric changes in a deployment scenario, such as communication radius and team size.

APPENDIX I

A. proof to Theorem 1

In the following, we report the proof of theorem (1).

Proof. Without loss of generality, we assume that the initial state belongs to the invariant set $\mathcal{X} = [-1, 1]^{N\times F}$. This is not a restrictive assumption since even if $\boldsymbol{x}(0) \notin \mathcal{X}$, at the next iteration it will be in $\mathcal{X}$ due to the activation function σ_c. In light of the definitions given in (8), it holds that

$$\begin{aligned}||\boldsymbol{x}^+||_\infty &\leq ||\sigma_c(\hat{q}\circ A_S(\boldsymbol{x}) + \tilde{q}\circ B_S(\boldsymbol{u}) + \boldsymbol{b})||_\infty \\ &\leq ||\hat{q}\circ A_S(\boldsymbol{x}) + \tilde{q}\circ B_S(\boldsymbol{u}) + \boldsymbol{b}||_\infty \\ &\leq \sigma_{\hat{q}}||S_K||_\infty||A||_\infty||\boldsymbol{x}||_\infty + \\ &\quad \sigma_{\tilde{q}}||S_K||_\infty||B||_\infty||\boldsymbol{u}||_\infty + ||\boldsymbol{b}||_\infty \\ &\leq \mathcal{A}||\boldsymbol{x}||_\infty + \mathcal{B}||\boldsymbol{u}||_\infty + ||\boldsymbol{b}||_\infty\end{aligned} \tag{21}$$

From theorem (1), by iterating the (21) for t steps we get

$$\begin{aligned}||\boldsymbol{x}(t)||_\infty \leq \mathcal{A}^t||\boldsymbol{x}(0)||_\infty + ||(1-\mathcal{A})^{-1}\mathcal{B}||_\infty||\boldsymbol{u}||_\infty + \\ ||(1-\mathcal{A})^{-1}||_\infty||\boldsymbol{b}||_\infty\end{aligned}$$

which proves the ISS property according to the definition II.1 □

B. proof to Theorem 2

In the following, we report the proof of theorem 2

Proof. Given two states $\boldsymbol{x}_1, \boldsymbol{x}_2$, it holds that

$$\begin{aligned}\boldsymbol{x}_1^+ - \boldsymbol{x}_2^+ =& \sigma_c(\hat{q}_1\circ A_{S1}(\boldsymbol{x_1}) + \tilde{q}_1\circ B_{S1}(\boldsymbol{u_1}) + \boldsymbol{b}) \\ &- \sigma_c(\hat{q}_2\circ A_{S2}(\boldsymbol{x_2}) + \tilde{q}_2\circ B_{S2}(\boldsymbol{u_2}) + \boldsymbol{b}).\end{aligned}$$

Owing to the Lipschitz assumption of σ_c and σ for Lipschitz constants respectively of 1 and $\frac{1}{4}$, the distance between the two state trajectories is bounded by

$$\begin{aligned}&||\boldsymbol{x}_1^+ - \boldsymbol{x}_2^+||_\infty \leq \\ &\quad ||(\hat{q}_1\circ A_{S1}(\boldsymbol{x_1}) + \tilde{q}_1\circ B_{S1}(\boldsymbol{u_1})) - \\ &\quad\quad (\hat{q}_2\circ A_{S2}(\boldsymbol{x_2}) + \tilde{q}_2\circ B_{S2}(\boldsymbol{u_2}))||_\infty \leq \\ &\quad ||(\hat{q}_1\circ A_{S1}(\boldsymbol{x_1}) - \hat{q}_2\circ A_{S2}(\boldsymbol{x_2})) + \\ &\quad\quad (\tilde{q}_1\circ B_{S1}(\boldsymbol{u_1}) - \tilde{q}_2\circ B_{S2}(\boldsymbol{u_2}))||_\infty \leq \\ &\quad ||\hat{q}_1\circ(A_{S1}(\boldsymbol{x_1}) - A_{S2}(\boldsymbol{x_2}))||_\infty + \\ &\quad\quad ||(\hat{q}_1 - \hat{q}_2)\circ A_{S2}(\boldsymbol{x_2})||_\infty + \\ &\quad\quad ||\tilde{q}_1\circ(B_{S1}(\boldsymbol{u_1}) - B_{S2}(\boldsymbol{u_2}))||_\infty + \\ &\quad\quad ||(\tilde{q}_1 - \tilde{q}_2)\circ B_{S2}(\boldsymbol{u_2})||_\infty.\end{aligned} \tag{22}$$

Different input features will correspond to different graph topologies and different support matrices S_1, S_2; thus different graph filters with the same parameters. For this reason, the differences of graph filters in the previous equation require further development. Let us focus on the state dependent graphs of the previous inequality. It holds that

$$\begin{aligned}&||\hat{q}_1\circ(A_{S1}(\boldsymbol{x_1}) - A_{S2}(\boldsymbol{x_2}))||_\infty \leq \\ &\quad \sigma_{\hat{q}}||S_{K1}(I_K\otimes\boldsymbol{x_1})A - S_{K2}(I_K\otimes\boldsymbol{x_2})A||_\infty \leq \\ &\quad \sigma_{\hat{q}}||S_{K1}(I_K\otimes\boldsymbol{x_1} - I_K\otimes\boldsymbol{x_2}) + \\ &\quad\quad (S_{K1} - S_{K2})(I_K\otimes\boldsymbol{x_2})||_\infty||A||_\infty.\end{aligned} \tag{23}$$

Under the assumption of theorem 2, we have $||S_{K1}||_\infty, ||S_{K2}||_\infty \leq ||\bar{S}_K||_\infty$. In light of $||\boldsymbol{x}||_\infty \leq 1$, equation (23) becomes

$$\begin{aligned}&||\hat{q}_1\circ(A_{S1}(\boldsymbol{x_1}) - A_{S2}(\boldsymbol{x_2}))||_\infty \leq \\ &\quad\quad \sigma_{\hat{q}}(||\bar{S}_K||_\infty||A||_\infty||\boldsymbol{x_1} - \boldsymbol{x_2}||_\infty + \\ &\quad\quad (||S_{K1} - S_{K2}||_\infty)||A||_\infty).\end{aligned} \tag{24}$$

Applying the same reasoning to the other terms in the inequality (22), we obtain

$$\begin{aligned}&||\boldsymbol{x}_1^+ - \boldsymbol{x}_2^+||_\infty \leq \\ &\quad (\sigma_{\hat{q}}||\bar{S}_K||_\infty||A||_\infty + \frac{1}{4}||\bar{S}_K||_\infty^2||\hat{A}||_\infty||A||_\infty + \\ &\quad \frac{1}{4}||\bar{S}_K||_\infty^2||\tilde{A}||_\infty||B||_\infty)||\boldsymbol{x_1} - \boldsymbol{x_2}||_\infty + \\ &\quad (\sigma_{\tilde{q}}||\bar{S}_K||_\infty||B||_\infty + \frac{1}{4}||\bar{S}_K||_\infty^2||\hat{B}||_\infty||A||_\infty + \\ &\quad \frac{1}{4}||\bar{S}_K||_\infty^2||\tilde{B}||_\infty||B||_\infty)||\boldsymbol{u_1} - \boldsymbol{u_2}||_\infty + \\ &\quad \mathcal{W}(||S_{K1} - S_{K2}||_\infty) \leq \\ &\quad \mathcal{A}_\delta||\boldsymbol{x_1} - \boldsymbol{x_2}||_\infty + \mathcal{B}_\delta||\boldsymbol{u_1} - \boldsymbol{u_2}||_\infty + \\ &\quad \mathcal{W}(||S_{K1} - S_{K2}||_\infty)\end{aligned} \tag{25}$$

where $\mathcal{W}$ gathers all the coefficient multiplying the difference $||S_{K1} - S_{K2}||_\infty$. We can consider this latter as an additional bounded input, $||S_{K1} - S_{K2}||_\infty \leq ||\bar{S}_K||_\infty - 1$ which, analogously to the input features, is defined by the team state. Hence, as stated in the theorem (2), it holds

$$\begin{aligned}&||\boldsymbol{x_1}(t) - \boldsymbol{x_2}(t)||_\infty \leq \mathcal{A}_\delta^t||\boldsymbol{x_1}(0) - \boldsymbol{x_2}(0)||_\infty + \\ &\quad (1-\mathcal{A}_\delta)^{-1}\delta\mathcal{B}||\boldsymbol{u_1} - \boldsymbol{u_2}||_\infty + \\ &\quad (1-\mathcal{A}_\delta)^{-1}\mathcal{W}||S_{K1} - S_{K2}||_\infty.\end{aligned} \tag{26}$$

The state trajectories have then a maximum distance that is asymptotically bounded by a function monotonically increasing with the maximum distance between the input sequences for

$$\gamma_\delta = \begin{bmatrix}(1-\mathcal{A}_\delta)^{-1}\delta\mathcal{B} & (1-\mathcal{A}_\delta)^{-1}\mathcal{W}\end{bmatrix} \left\| \begin{matrix} \boldsymbol{u_1}-\boldsymbol{u_2} \\ S_{K1}-S_{K2}\end{matrix} \right\|_\infty . \tag{27}$$

Therefore the system is incrementally ISS under the definition II.2.

□

C. proof of Theorem 4

Proof. To analyse the incremental stability it is useful to separate each layer. From the proof of theorem I-B, for the first layer we know that:

$$\begin{aligned}||\boldsymbol{x_1}^{1+}-\boldsymbol{x_2}^{1+}||_\infty \leq \mathcal{A}_\delta^1||\boldsymbol{x_1}^1-\boldsymbol{x_2}^1||_\infty + \mathcal{B}_\delta^1||\boldsymbol{u_1}-\boldsymbol{u_2}||_\infty + \\ \mathcal{W}^1||S_{K1}-S_{K2}||_\infty\end{aligned} \tag{28}$$

As a result, for the second it holds

$$\begin{aligned}&||\boldsymbol{x_1}^{2+}-\boldsymbol{x_2}^{2+}||_\infty \leq \\ &\quad \mathcal{A}_\delta^2||\boldsymbol{x_1}^2-\boldsymbol{x_2}^2||_\infty + \mathcal{B}_\delta^2||\boldsymbol{x_1}^{1+}-\boldsymbol{x_2}^{1+}||_\infty + \\ &\quad \mathcal{W}^2||S_{K1}-S_{K2}||_\infty \leq \\ &\quad \mathcal{A}_\delta^2||\boldsymbol{x_1}^2-\boldsymbol{x_2}^2||_\infty + \mathcal{B}_\delta^2\mathcal{A}_\delta^1||\boldsymbol{x_1}^1-\boldsymbol{x_2}^1||_\infty + \\ &\quad \mathcal{B}_\delta^2\mathcal{B}_\delta^1||\boldsymbol{u_1}-\boldsymbol{u_2}||_\infty + \\ &\quad \mathcal{B}_\delta^2\mathcal{W}^1||S_{K1}-S_{K2}||_\infty + \mathcal{W}^2(||S_{K1}-S_{K2}||_\infty\end{aligned} \tag{29}$$

Denoting $\Delta X = \begin{bmatrix}\boldsymbol{x_1}^1-\boldsymbol{x_2}^1 & \dots & \boldsymbol{x_1}^M-\boldsymbol{x_2}^M\end{bmatrix}^T$, $\Delta U = \boldsymbol{u_1}-\boldsymbol{u_2}$, $\Delta S_K = S_{K1}-S_{K2}$ iterating the same reasoning for M layers we get

$$\begin{aligned}||\Delta X^+||_\infty \leq \begin{bmatrix} A_\delta^1 & 0 & \dots & 0 \\ B_\delta^2 A_\delta^1 & A_\delta^2 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ A_\delta^1 \prod_{h=2}^{M} B_\delta^h & \dots & \dots & A_\delta^M \end{bmatrix} ||\Delta X||_\infty + \\ \begin{bmatrix} B_\delta^1 \\ B_\delta^2 \\ \vdots \\ \prod_{h=1}^{M} B_\delta^h \end{bmatrix} ||\Delta U||_\infty + \begin{bmatrix} \mathcal{W}^1 \\ \mathcal{W}^1 B_\delta^2 + \mathcal{W}^2 \\ \vdots \\ \prod_{h=1}^{M-1} \mathcal{W}^h B_\delta^{h+1} + \mathcal{W}^M \end{bmatrix} ||\Delta S||_\infty \\ \leq M_\delta||\Delta X||_\infty + M_{B\delta}||\boldsymbol{u_1}-\boldsymbol{u_2}||_\infty + M_{\mathcal{W}\delta}||S_{K1}-S_{K2}||_\infty\end{aligned} \tag{30}$$

Since the matrix M_δ is lower triangular, its eigenvalues are the elements of its diagonal $\mathcal{A}_\delta^i$ and are equal to the eigenvalues of the matrix M_δ. □

REFERENCES

[1] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[2] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *Ieee Access*, vol. 6, pp. 28 573–28 593, 2018.

[3] J. Cortés and M. Egerstedt, "Coordinated control of multi-robot systems: A survey," *SICE Journal of Control, Measurement, and System Integration*, vol. 10, no. 6, pp. 495–503, 2017.

[4] G.-B. Huang, L. Chen, C. K. Siew, *et al.*, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 879–892, 2006.

[5] S. Batra, Z. Huang, A. Petrenko, T. Kumar, A. Molchanov, and G. S. Sukhatme, "Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 576–586.

[6] C. He, Y. Wan, Y. Gu, and F. L. Lewis, "Integral reinforcement learning-based multi-robot minimum time-energy path planning subject to collision avoidance and unknown environmental disturbances," *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 983–988, 2020.

[7] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*. Springer, 2017, pp. 66–83.

[8] N. Majcherczyk, N. Srishankar, and C. Pinciroli, "Flow-fl: Data-driven federated learning for spatio-temporal predictions in multi-robot systems," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8836–8842.

[9] L. Wu, P. Cui, J. Pei, L. Zhao, and L. Song, "Graph neural networks," in *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer, 2022, pp. 27–37.

[10] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[11] F. Gama and S. Sojoudi, "Graph neural networks for distributed linear-quadratic control," in *Learning for Dynamics and Control*. PMLR, 2021, pp. 111–124.

[12] F. Gama, E. Tolstaya, and A. Ribeiro, "Graph neural networks for decentralized controllers," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 5260–5264.

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

[14] E. Tolstaya, J. Paulos, V. Kumar, and A. Ribeiro, "Multi-robot coverage and exploration using spatial graph neural networks," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 8944–8950.

[15] A. Khan, A. Ribeiro, V. Kumar, and A. G. Francis, "Graph neural networks for motion planning," *arXiv preprint arXiv:2006.06248*, 2020.

[16] X. Ji, H. Li, Z. Pan, X. Gao, and C. Tu, "Decentralized, unlabeled multi-agent navigation in obstacle-rich environments using graph neural networks," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 8936–8943.

[17] Y. Zhou, J. Xiao, Y. Zhou, and G. Loianno, "Multi-robot collaborative perception with graph neural networks," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2289–2296, 2022.

[18] M. Tzes, N. Bousias, E. Chatzipantazis, and G. J. Pappas, "Graph neural networks for multi-robot active information acquisition," *arXiv preprint arXiv:2209.12091*, 2022.

[19] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 269–296, 2020.

[20] A. Davydov, A. V. Proskurnikov, and F. Bullo, "Non-euclidean contractivity of recurrent neural networks," in *2022 American Control Conference (ACC)*. IEEE, 2022, pp. 1527–1534.

[21] B. Song, J.-J. Slotine, and Q.-C. Pham, "Stability guarantees for continuous rl control," *arXiv preprint arXiv:2209.07324*, 2022.

[22] H. Tsukamoto, S.-J. Chung, and J.-J. Slotine, "Learning-based adaptive control using contraction theory," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 2533–2538.

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

[24] F. Bonassi, M. Farina, and R. Scattolini, "On the stability properties of gated recurrent units neural networks," *Systems & Control Letters*, vol. 157, p. 105049, 2021.

[25] E. Terzi, F. Bonassi, M. Farina, and R. Scattolini, "Learning model predictive control with long short-term memory networks," *International Journal of Robust and Nonlinear Control*, vol. 31, no. 18, pp. 8877–8896, 2021.

[26] F. Bayer, M. Bürger, and F. Allgöwer, "Discrete-time incremental iss: A framework for robust nmpc," in *2013 European Control Conference (ECC)*. IEEE, 2013, pp. 2068–2073.

[27] L. Ruiz, F. Gama, and A. Ribeiro, "Gated graph recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 6303–6318, 2020.

[28] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 5680–5695, 2020.

[29] 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 Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[30] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 5680–5695, 2020.

[31] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized path planning," in *Proc. International Conference on Autonomous Agents and Multiagent Systems*, 2020, pp. 1901–1903.

[32] E. Tolstaya, J. Paulos, V. Kumar, and A. Ribeiro, "Multi-robot coverage and exploration using spatial graph neural networks," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 8944–8950.

[33] A. Sandryhaila and J. M. Moura, "Discrete signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

[34] A. Nicolicioiu, I. Duta, and M. Leordeanu, "Recurrent space-time graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[35] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017.

[36] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*. PMLR, 2013, pp. 1310–1318.

[37] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[38] D. Lukovnikov, J. Lehmann, and A. Fischer, "Improving the long-range performance of gated graph neural networks," *arXiv preprint arXiv:2007.09668*, 2020.

[39] Z.-P. Jiang and Y. Wang, "Input-to-state stability for discrete-time nonlinear systems," *Automatica*, vol. 37, no. 6, pp. 857–869, 2001.

[40] D. Angeli, "A lyapunov approach to incremental stability properties," *IEEE Transactions on Automatic Control*, vol. 47, no. 3, pp. 410–421, 2002.

[41] J. Jouffroy, "A simple extension of contraction theory to study incremental stability properties," in *2003 European Control Conference (ECC)*. IEEE, 2003, pp. 1315–1321.

[42] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Conference on robot learning*. PMLR, 2020, pp. 671–682.

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

[44] W. Yu, G. Chen, and M. Cao, "Distributed leader–follower flocking control for multi-agent dynamical systems with time-varying velocities," *Systems & Control Letters*, vol. 59, no. 9, pp. 543–552, 2010.

[45] L. E. Beaver and A. A. Malikopoulos, "An overview on optimal flocking," *Annual Reviews in Control*, vol. 51, pp. 88–99, 2021.

[46] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Stable flocking of mobile agents part i: dynamic topology," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, vol. 2. IEEE, 2003, pp. 2016–2021.

[47] 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*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[49] D. Le and E. Plaku, "Multi-robot motion planning with dynamics via coordinated sampling-based expansion guided by multi-agent search," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1868–1875, 2019.

[50] J. Tordesillas and J. P. How, "Mader: Trajectory planner in multiagent and dynamic environments," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 463–476, 2021.

[51] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 11 785–11 792.

[52] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 2997–3004.

Antonio Marino obtained his M.Sc. degree in robotics engineering at the University of Genova in 2020. A. Marino is currently a PhD candidate in the Rainbow team at IRISA/Inria Rennes working on learning-based multi-robot formation control.

Claudio Pacchierotti (SM'20) is a tenured researcher at CNRS-IRISA in Rennes, France, since 2016. He was previously a postdoctoral researcher at the Italian Institute of Technology, Genova, Italy. Pacchierotti earned his PhD at the University of Siena in 2014. He was Visiting Researcher in the Penn Haptics Group at University of Pennsylvania in 2014, the Dept. of Innovation in Mechanics and Management at University of Padua in 2013, the Institute for Biomedical Technology and Technical Medicine (MIRA) at University of Twente in 2014, and the Dept. Computer, Control and Management Engineering of the Sapienza University of Rome in 2022. Pacchierotti received the 2014 EuroHaptics Best PhD Thesis Award and the 2022 CNRS Bronze Medal. He is Senior Chair of the IEEE Technical Committee on Haptics and Secretary of the Eurohaptics Society.

Paolo Robuffo Giordano (M'08-SM'16) received his M.Sc. degree in Computer Science Engineering in 2001, and his Ph.D. degree in Systems Engineer- ing in 2008, both from the University of Rome "La Sapienza". In 2007 and 2008 he spent one year as a PostDoc at the Institute of Robotics and Mechatronics of the German Aerospace Center (DLR), and from 2008 to 2012 he was Senior Research Scientist at the Max Planck Institute for Biological Cybernetics in Tübingen, Germany. He is currently a senior CNRS researcher head of the Rainbow group at Irisa and Inria in Rennes, France.