

Clone Swarms: Learning to Predict and Control Multi-Robot Systems by Imitation

Siyu Zhou¹, Mariano J. Phielipp², Jorge A. Sefair³, Sara I. Walker⁴ and Heni Ben Amor³

Abstract—In this paper, we propose SwarmNet – a neural network architecture that can learn to predict and imitate the behavior of an observed swarm of agents in a centralized manner. Tested on artificially generated swarm motion data, the network achieves high levels of prediction accuracy and imitation authenticity. We compare our model to previous approaches for modelling interaction systems and show how modifying components of other models gradually approaches the performance of ours. Finally, we also discuss an extension of SwarmNet that can deal with nondeterministic, noisy, and uncertain environments, as often found in robotics applications.

I. INTRODUCTION

Multi-Robot Systems (MRS) [1] describe groups of robotic agents that collectively perform complex tasks in a distributed and parallel manner through repeated interactions among each other and the environment. Such systems have attracted considerable attention in recent years with remarkable successes in a number of application domains, including defense, agriculture, logistics, disaster management, and entertainment. In particular, today’s fast-paced online economy is largely fuelled by tens of thousands of warehouse robots that transport millions of items across fulfillment centers all over the world.

Despite this progress, programming groups of robots to perform a joint task is still considered a complex, time-consuming, and extremely challenging endeavour. One prominent formalism for the specification of MRS is based on the identification of cost functions [2] governing the group behavior. However, this approach is not intuitive and requires a deep understanding of complex theoretical concepts across a number of mathematical fields, e.g., graph theory, manifold theory, nonlinear optimization, etc. In addition, the real-world ramifications of even small changes in a given cost function are extremely difficult to foresee. An alternative approach is to provide a set of building-block behaviors [3] that are combined to produce overall group strategies, e.g., formation or area covering. An important aspect in this regard is the notion of *emergence* in which simple rules interact with each other to generate a whole that is more complex than the sum of its parts. Higher levels of complexity emerge as a result of repeated simple local interactions.

¹Department of Physics, Arizona State University
siyu.zhou@asu.edu

²Intel Corporation mariano.j.phielipp@intel.com

³School of Computing, Informatics, and Decision Systems Engineering, Arizona State University {jorge.sefair, hbenamor}@asu.edu

⁴School of Earth and Space Exploration, Arizona State University
sara.i.walker@asu.edu

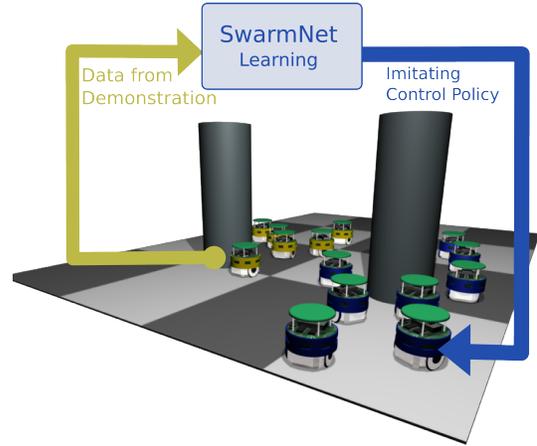


Fig. 1. Visualization of the SwarmNet approach: data regarding the position of agents is recorded from an existing artificial or natural swarm (light colored robots). In turn, a graph neural network is learned that models the observed behavior. The trained SwarmNet can then be used as a policy to synthesize similar behavior for a multi-robot system (dark colored robots).

This transition in complexity due to emergent properties is difficult to predict and introduces substantial challenges in the design of the right set of building-block behaviors.

In this paper, we propose an alternative approach based on learning-from-demonstration (LfD) [4] for specifying group behavior in multi-robot systems. The LfD methodology has a rich history in single-robot systems, with a number of successful applications in real-world tasks such as table-tennis, manipulation, locomotion, and helicopter flying [5]. We extend this methodology to the MRS case by introducing a novel graph neural network architecture that can be trained from execution traces of a swarm. Swarm systems can be naturally modeled as graphs, with nodes representing swarm members and edges describing interactions between these agents. The introduced graph neural network, called *SwarmNet*, extracts all rules governing the group behavior from data alone, i.e., sequences of agent positions and velocities. Once a SwarmNet is extracted, it can be used to perform complex inferences including prediction, imitation or replacement of an existing swarm from a centralized view point (see Fig. 1).

Moving beyond hand-coding the rules for inter-robot and robot-environment interactions, the approach presented in this paper allows for a data-driven methodology for the specification of swarm behavior. In particular, the contributions of this paper include:

- SwarmNet – a graph neural network that can be trained from observations of a natural or artificial swarm. After

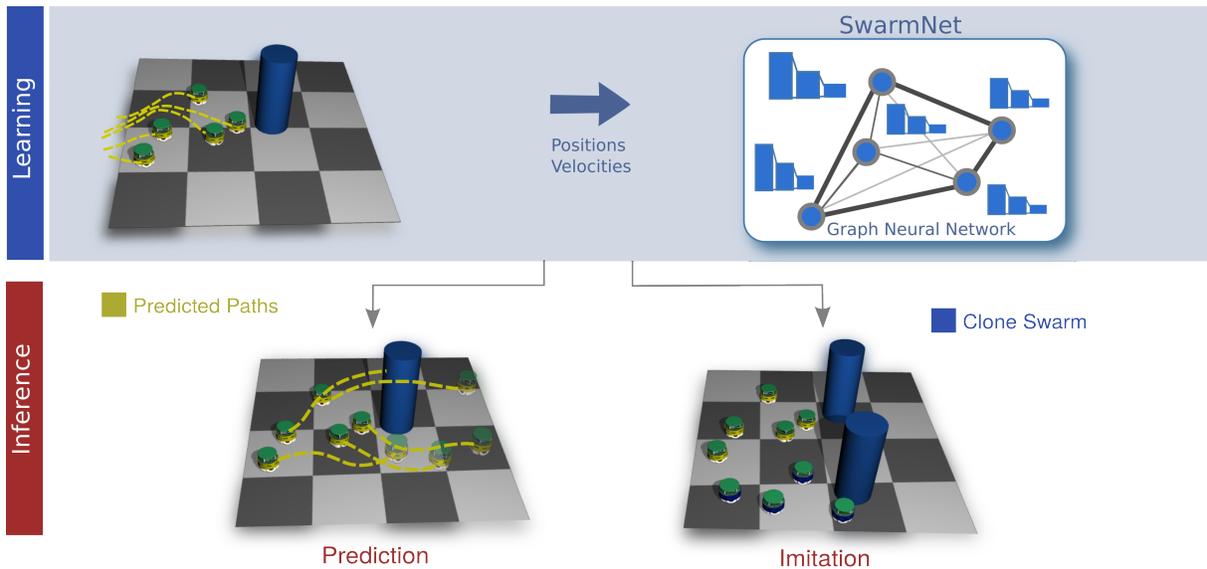


Fig. 2. Overview of the proposed methodology: we collect training data from an observed swarm to learn a compact graph neural network representation called SwarmNet. In turn, it can be used to predict future behavior, augment an existing swarm with more agents, or create a clone of the swarm with similar behavior.

training, SwarmNet can be used to (a) predict the behavior of all members of an observed swarm or (b) synthesize a *clone swarm* – a new swarm mimicking the trained behavior.

- Incorporation of contextual and environmental information, e.g., the location of obstacles and goals, into the learning and inference process of the swarm. As a result, a trained model of the swarm can be used as a reactive policy to control all agents.
- An extension of our approach, called SwarmNet⁺ which can model nondeterministic swarms and environments. SwarmNet⁺ captures the underlying probability distributions of group behavior. Sampling from this distribution generates anticipated trajectories for all agents along with inherent uncertainties.

We evaluate our approach on a number of data sets generated from common models for flocking and swarming, e.g., the Boids model [3] and the Helbing model [6].

II. RELATED WORK

Research on modeling, prediction and control of multi-agent systems, e.g., teams of quadcopters, autonomous cars, or other forms of unmanned vehicles, has gained considerable attention in a variety of research disciplines. For an excellent recent survey of how such teams can be formalized, coordinated, and controlled, we refer the reader to [7]. Among the formalisms most widely used for representing and studying MRS and swarms are graph-theoretic [8] approaches and methods based on cost functions and auctioning processes [2]. In contrast to the explicit modeling of MRS strategies presented in these approaches, one could also employ imitation and LfD to extract such policies [4]. An early approach investigating the potential to copy the behavior of an MRS via imitation learning

was presented in [9]. This approach used Gaussian mixture models to extract probability distributions underlying the agents’ interactions. However, this approach was limited to discrete action spaces only. More recently, the work in [10] used deep neural networks to produce generative models of multi-agent behavior. In contrast to graph theory based methodologies, the approach in [10] builds upon traditional, feedforward, and recurrent neural networks that do not explicitly model the team structure. Hence, the modelling and prediction task is reduced to a pure function approximation framework, which may neglect critical structural dependencies between the members of an MRS. This approach also assumes that macro-goals, e.g., a discrete set of sub-tasks an agent can take on, are available. In addition to imitation learning, modern reinforcement learning methods have also been used to generate MRS behavior [11]. The approach in [12] combines both reinforcement learning and imitation learning for extracting multi-agent navigation policies. The approach can deal with partially-observable domains, variable team-sizes, as well as complex environments and mazes. However, it also assumes a discrete set of actions and focuses on structured environments, such as warehouses and factories. In contrast, we focus on continuous action spaces and potentially unstructured nondeterministic environments. Further, our approach allows for the prediction of the most likely behavior of an observed swarm given previously seen behaviors.

III. METHODOLOGY

In this section, we describe our methodology that is at the core of our approach. Fig. 2 depicts an overview of both the learning and inference process for MRS¹. Learning is achieved by recording execution traces of an observed

¹We will henceforth use the term “swarm” and MRS interchangeably.

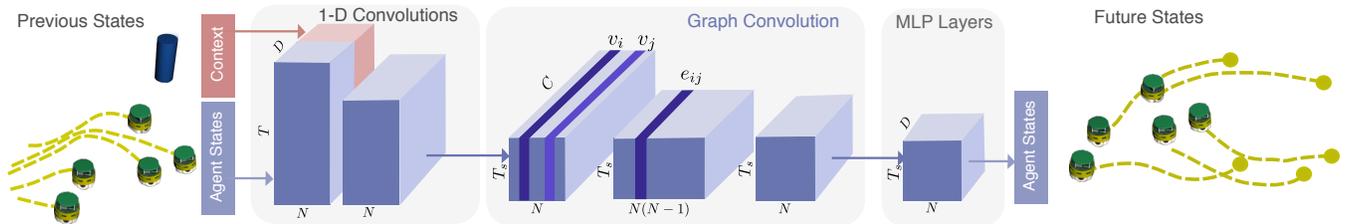


Fig. 3. Diagram of the network architecture for SwarmNet: Letters along sides indicate dimensions. T is the number of time steps, N is the number of agents in the system, and D is the length of the state vector. H is the size of the encoded state vector, which is the size of the last layer of an MLP. Node states v_i and v_j are passed through Graph Convolution to produce interactions e_{ij}

swarm. Execution traces are discretely sampled trajectories specifying the position and velocity of each agent at time step t . Each trace represents a demonstration of the group behavior as observed in the swarm. The set of traces is, in turn, used to train *SwarmNet* – our novel graph neural network that encodes the dynamics among the agents of a swarm. Note that *SwarmNet* needs to account for both inter-agent interactions (e.g. how the members influence each other’s behavior), as well as agent-environment interactions. Typically, swarms are also influenced by the current context and environmental variables. For example, in Fig. 2, the obstacle needs to be taken into account to generate reasonable behaviors that avoid collisions with the environment.

After training, the extracted *SwarmNet* can be used to implement two core functionalities, namely (a) prediction or (b) imitation of a swarm. In the prediction mode, we monitor the ongoing behavior of a swarm and predict the future locations of all involved agents. Such functionality is helpful, for example, in interdiction tasks [13], [14], [15] in which a system has to generate recommendations on the least-cost and most-effective actions needed to thwart or disrupt the threat posed by an adversarial swarm. *SwarmNet* can also be used as a policy to generate a completely new swarm, which we refer to as *clone swarms*. A clone swarm imitates the behavior seen during training. Since *SwarmNet* is trained in a data-driven fashion, a variety of input sources can be used to train models of multi-robot behavior, including previous swarming models, expert robot users, environmental and contextual attributes, and data from a biological swarm.

A. Network Architecture

Fig. 3 describes the full architecture of *SwarmNet*. Input to the network is a set of previous robot states, as well as contextual information regarding the environment. Robot states are specified by a time window of positions and velocities of either (1) observed agents, or (2) controlled agents, depending on the mode of operation. Assuming an N -agent swarm, the dynamical state of agent $i \in \mathbb{N}$ can be written as $s_i(t) = [\mathbf{x}_i(t), \dot{\mathbf{x}}_i(t)]$, where $\mathbf{x}_i(t)$ is the position and $\dot{\mathbf{x}}_i(t)$ is the velocity of agent i at timestep t . Without loss of generality and for notational clarity, we will assume subsequently that agents live in a 2D space. The output of the network are the future state vectors $s_i(t+1)$, given the state history of the swarm system and context.

The states $s_i(t)$ and context $c(t)$ are concatenated and

first passed through a series of one-dimensional convolution layers along the time axis. These convolutions allow the network to extract information regarding short-term dynamics of a single trajectory. In the Markovian case where the state of the next step only depends on the present step, the kernel of 1D convolutions would just assign a weight of 1 to the current state and 0s for all earlier steps. The evolution of the agents’ dynamic states is a result of pair-wise interactions between them. The dynamics of the system and the underlying elementary interactions can be formulated as a graph, with nodes being the agents, their dynamical states being node states, the interaction relations being edges, and the interaction effects being edge states. To avoid any limiting assumptions or constraints, the states are embedded in a fully-connected directed graph which models all relationships, i.e., the learnable functions along the graph edges define whether agents are interacting or not. To process the interactions and update the node states for predicting the next step, a graph convolution over our representation is employed. The final set of operations in *SwarmNet* is a set of traditional fully-connected neural network layers. These final layers take the result of the graph convolution as input and generate a window of predictions over the future states of all agents. Subsequently, we will describe each one of these three sub-components in more detail.

B. 1D Convolutions

The set of one-dimensional convolutions in *SwarmNet* aims at incorporating a temporal context to the decision-making process. The approach used here avoids the usage of recurrent connections [16], which often introduces challenging nonlinearities thereby increasing the complexity of training and reducing the interpretability of the learned operations.

In our approach, motion data is organized as a tensor (see Fig 3) with an overall dimension of $T \times N \times D$, where T is the number of timesteps, N is the number of agents, and D is the dimension of the state vector, e.g., $s_i(t) \in \mathbb{R}^4$ in 2D space. The time-series data is first passed through a series of 1D convolution layers along the time axis without padding, see Fig. 4. L layers each with a kernel size of K would condense a time-series of length $T_w = L(K-1) + 1$ to a higher-order feature of length 1. This can be considered an abstraction of the windowed temporal history into a more concise form used for prediction. A time-series of length

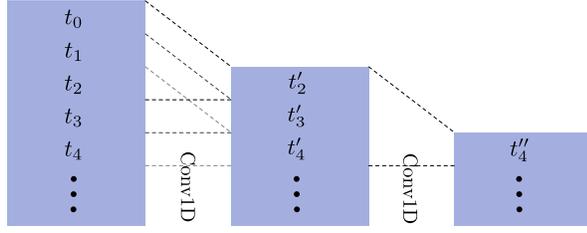


Fig. 4. 1D Convolution: for a kernel of size $K = 3$, the time series is transformed by replacing each time step with the weighted sum of itself and its 2 previous steps. Since the time series is not padded beyond step 0, every convolution layer removes 2 steps from the beginning. A filter with weights $(0, 0, 1)$ ignores earlier histories and only considers the present time step, thereby implementing a Markov assumption. A filter with weights $(0, -1, 1)$ effectively approximates the first order derivative, and one with $(1, -2, 1)$ approximates the second order derivative, etc. A group of such filters would be sufficient to capture the dynamics of histories.

T will produce $T_s = T + 1 - T_w = T - L(K - 1)$ such values after the kernel slides through every timestep in T . In particular, we chose $L = 3$ layers with kernel size $K = 3$, effectively having a window length $T_w = 7$. Note, that similar to convolutional filters in image processing, we can use multiple 1D convolution filters in every layer. In our case, the different filters focus on different aspects of the trajectories and identify temporal patterns that can be identified to generate better predictions (Fig. 4). We define C to be the number of different 1D convolutional filters. Overall, the output of the entire 1D convolution step is condensed tensor of dimensionality $T_s \times N \times C$, which is then processed via a graph convolution.

C. Graph Convolution

The next step in SwarmNet is a graph convolution (GC) over the condensed state vectors. A GC is an operation applied uniformly across all nodes of a graph along with their local neighborhoods and is used to update the node and edge states of the next step, similar to the convolution in CNNs. As opposed to CNN, where the neighborhood of an element is the spatially neighboring elements inside an array, a matrix or a tensor, the neighborhood of a node is dictated by the connectivity of the underlying graph.

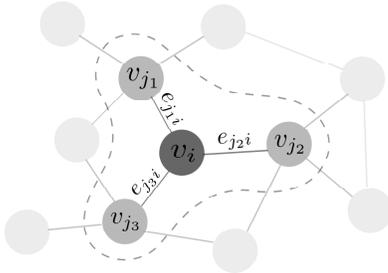


Fig. 5. An example of graph convolution applied on the neighborhood of vertex i which includes vertices j_1, j_2 and j_3 . The dashed line encloses the neighborhood of vertex i . In a directed graph, edges sourcing from i have no effect on i , and only edges targeting i are considered. Graph convolution updates node attributes v_i according to Eq. (1)-(3)

The agent system is embedded in a directed graph $\mathcal{G} =$

(N, E) , where N is the set of nodes and $E = \{(i, j) : i, j \in N, i \neq j\}$ is the set of edges. In \mathcal{G} , each node represents an agent and each of the $N(N - 1)$ edges represent an interaction between a pair of distinct agents. The vector of attributes of node $i \in N$ is given by $v_i \in \mathbb{R}^{d_v}$, where d_v is the number of node attributes. Edges are uniquely identified by their connected nodes, thus we represent the attributes of the edge targeting node j from i (i.e., edge $(i, j) \in E$) by $e_{ij} \in \mathbb{R}^{d_e}$, where d_e is the number of edge attributes. While node attributes model the state of the embedded agent, edge attributes model the interactive influence of source nodes onto target nodes. These interactions include pulling, pushing, and steering. For a thorough discussion of graph convolutions we refer the reader to [17].

Building upon the previous 1D convolution step, we use the condensed state vectors as the node states v_i , see Fig. 5. The node states for each member are determined by looking up the corresponding entries in the tensor which was generated from 1D convolutions, as can be seen in the graph convolution section Fig. 3. Over time, agent states change due to the influence of interactions, which is in turn dependent on the new agent states. This alternating update of node states and edge states in the graph is propagated by GC. The GC process updates the node state as well as the edge state from step t to step $t + 1$ using the following functions:

$$e_{ij} \leftarrow \phi^e(v_i, v_j) \quad (1)$$

$$\bar{e}_i \leftarrow \psi^e\left(\sum_{j \in \mathcal{N}_i} e_{ji}\right) \quad (2)$$

$$v_i \leftarrow \phi^v(v_i, \bar{e}_i) \quad (3)$$

where i and j are node labels. In Eq. (2), \mathcal{N}_i is the set of neighboring source nodes' states, so \bar{e} is an aggregation of the states of edges connected to node i .

The GC process involves three steps:

- 1) Apply Eq. (1) to all edges
- 2) Aggregate states of connecting edges using Eq. (2)
- 3) Apply Eq. (3) to update states of all nodes

Note that in Eq.(1), we determine the edge state only by the present states of the connected nodes, and completely ignore the edge's past state. The GNN module treats the condensed time-series by 1D convolution as node states on an N -node graph. For each starting point of shape $N \times C$, each node's state is sent to the edges attached to it for edge update according to Eq (1). The edges effectively pull information from the two ends to deduce the interaction between them. We call this process *node aggregation*. Following Eq. (3), each node then aggregates the effect from incoming edges, resulting in a process called *edge aggregation* along its previous state to compute the outcome. Further GNN layers would repeat the process until the output $v_i(t + 1)$ of the last layer is taken as the prediction of dynamical states, i.e. positions and velocities, of the agents. In the proposed SwarmNet architecture, we perform only a single such GC operation as described above. The functions ϕ^e , ϕ^v and ψ^e are approximated using traditional multi-layer perceptrons

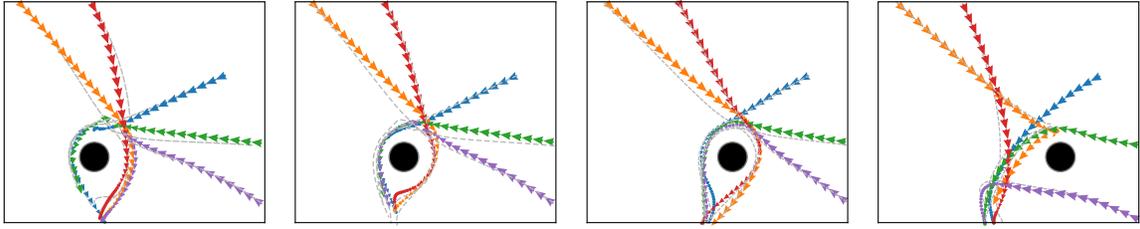


Fig. 6. From left to right: four different predictions of the swarm behavior using SwamNet. In each one of the four experiments, the obstacle is placed slightly more to the right. The colored arrows show the movement of 5 boids, while the gray dashed lines are the ground truth trajectories. The black circle represents the obstacle.

(MLP), i.e., feed forward neural networks. The MLPs apply on the last dimension. A final MLP after edge aggregation transforms the tensor to one with shape $T_s \times N \times D$, giving each of the T_s starting points the prediction of their next steps.

D. Loss Function

Training of the network is performed in a supervised fashion by comparing the predicted agent states (containing positions and velocities) with the ground-truth states in the time series. The mean squared-error (MSE) is used as the loss function L for training and as the metric for evaluation, with

$$L = \frac{1}{2DN T_s} \sum_{t=1}^{T_s} \sum_{i=1}^N (s_i(t) - s_i^*(t))^2 \quad (4)$$

where $s_i^*(t)$ is the ground-truth state vector of agent i at step t . The loss is normalized over the "natural skip", i.e., the MSE of state vectors between two consecutive steps in the ground truth trajectories, \bar{L} .

$$\bar{L} = \frac{1}{2DN(T-1)} \sum_{t=1}^{T-1} \sum_{i=1}^N (s_i^*(t+1) - s_i^*(t))^2 \quad (5)$$

Given the above normalization factor, the normalized loss can be calculated using the fraction $L_{norm} = \frac{L}{\bar{L}}$. Normalization rectifies the dependence of prediction error on the intrinsic spacing of the ground-truth trajectories.

E. Multistep Predictions and Curriculum Learning

To enable multistep prediction, the shifting windows of the time-series leading to the T_s starting points are stacked such that the restructured time-series has the shape $T_s \times T_w \times N \times D$, where $T_s = T - L(K-1)$ and $T_w = L(K-1) + 1$ as discussed earlier. Now that 1D convolution acts along T_w , eventually the prediction of next steps has the shape $T_s \times 1 \times N \times D$ and is appended to the input along the T_s dimension. Then, with the first state dropped, the time-window is shifted by one-step and is the temporal history of states for the prediction of the next step. This procedure can be iteratively applied to create prediction horizons of arbitrary length.

We scaffold the learning process by training with an increasing horizon for the predictions. More specifically, we increase the required number of prediction steps gradually through the course of training, from 1 to 10 steps. This

curriculum learning scheme has the benefit of starting with a simpler version of the task (i.e. using limited prediction horizon) and slowly exposing the network to more complex, long-term predictions. The approach also encourages the model to learn the true dynamics of the system than can be unfolded for arbitrary time steps, rather than potentially overfitting to fixed-term predictions.

F. Uncertainty, Noise and Nondeterminism

Swarms and multi-robot systems are typically acting in nondeterministic environments in which perceptual data is noisy and the effect of actions uncertain. We extend our methodology to nondeterministic environments by leveraging recent theoretical insights connected to *Bayesian* deep learning. In particular, the work in [18] shows that estimates of model uncertainty can be generated from a neural network using the Dropout [19] algorithm. Dropout is a training algorithm in which connections of a neural network are randomly activated and deactivated. This is achieved using a dropout probability p which describes the probability of an input activation being dropped. After training a network with Dropout, we can use the same technique to generate different outputs for the same input, i.e., deactivate layer input with probability p . In such a case, each forward pass through the network is called a *stochastic forward pass* and can be seen as a sample from the underlying probability distribution. According to [18], such stochastic forward passes in a deep network will be an approximation of variational inference in a Gaussian process. In our case, the set of stochastic forward passes $\mathcal{S} = \{\hat{s}_i^1(t+1), \dots, \hat{s}_i^S(t+1)\}$ represents S samples from the probability distribution over the future states of agent i , i.e., the distribution defining the range of different states the agent can be as a result of nondeterminism. Recursively sampling for longer horizons of the future generates the anticipated trajectories for all agents along with inherent uncertainties. In the remainder of this paper, we will refer to versions of our network that use this approach as *SwarmNet⁺*. It is important to note, that *SwarmNet⁺* can generate multiple different, potentially conflicting or bifurcating predictions for the same input state to the network – a property that is typically not possible in standard neural network training approaches.

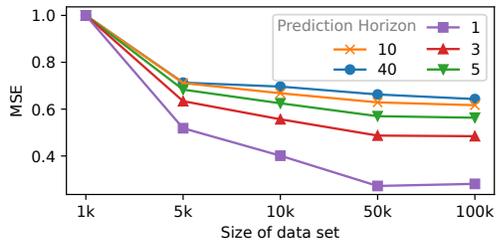


Fig. 7. Prediction error for different horizon lengths and sizes of training set with boids.

IV. EXPERIMENTS AND RESULTS

We generated training data using popular techniques for synthesizing swarm and MRS behavior, e.g., flocking and swarming models. In turn, we compared different versions of SwarmNet to the graph neural network (GNN) in [20] and the LSTM network method in [16]. Since SwarmNet shares strong similarities with GNN approaches such as [20], we tested different ablations and additions to [20] based on the insights in this paper. In addition, we performed a variety of experiments to investigate sample-efficiency and other critical aspects of SwarmNet.

A. Swarm Data Sets

Our data set consists of motion data produced using the Boids model [3], the Helbing model [6], and a simple chaser model. In the Boids model, $N = 5$ agents demonstrate flocking behaviors while approaching a common goal location and avoiding obstacles. The Helbing model [6] generates swarm behavior via repulsive forces only. The final model, called chaser, places a set of agents on a circle. Each of these agents generates steering actions that make it chase another agent of the swarm. For each of the above models we generate data sets by running the simulation for about 50 time steps while recording the agent positions and velocities, as well as the environmental variables. In both the Boids and Helbing model, the environmental variable is the positions of obstacles.

B. Prediction Accuracy

The first experiment focuses on the prediction accuracy when compared to other methods, in particular the method in [20] and LSTMs [16]. Table I summarizes the loss of the methods across the three data sets. We performed the experiment for prediction horizons of 5 steps and 40 steps. All methods were trained with 50K demonstrations of the corresponding swarm behavior. The best performance is achieved by SwarmNet with contextual inputs. Removing the context information has a significant impact on the SwarmNet performance, in particular in the long-term prediction case (right side of the table). On the Boids data set, the error for 40 step predictions jumps from 2.778 to about 10.47 when contextual information is omitted. In general, SwarmNet outperforms all other methods on all data sets and in both conditions. An interesting aspect of these results is that SwarmNet was only trained on data for predictions of

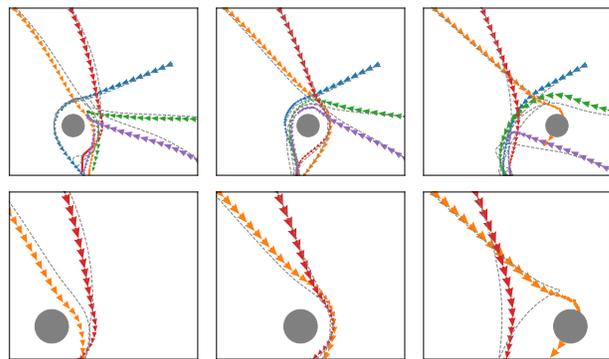


Fig. 8. Predictions for the behavior of a swarm generated from a network trained on 1000 demonstrations of boids simulation. The top row shows the movements of all agents. The bottom row highlights (for visibility) the movements of only two agents.

up to 10 steps, as previously described in Sec. III-E. Despite that, it correctly learned to make accurate predictions for 40 steps and beyond indicating that it focused on the true dynamics of the task.

Fig. 6 shows the prediction results for different locations of the obstacle (black circle). We see that the SwarmNet predictions are qualitatively implementing the correct swarming and avoidance behavior, even if small deviations from the ground truth occur. The rightmost example in Fig. 6 shows an interesting behavior in which the orange agent is first trying to circumvent the object on the right side and then turns to head back to goal location. While unintuitive, this behavior is also existent in the ground truth data.

Next, we investigated the influence of the *training set size* and the *prediction horizon* on the prediction results. Fig. 7 shows the MSE for five different horizon lengths (1-40 steps into the future) trained with data sets ranging from 100K demonstrations down to only 1K demonstrations. It is interesting to note that, in the case of long-term prediction, only when trained with less than 5K demonstrations does the MSE deteriorate. Even in the case of training on only 1K samples, SwarmNet still generates (qualitatively) reasonable swarm behavior that executes the intended task. Fig. 8 shows the behavior of a network trained on 1K examples of the Boids task. The generated predictions still follow the trend of the ground truth data, with one exception in which an agent trajectory (orange) is predicted to go through the obstacle.

C. Comparison to the Kipf Model

SwarmNet builds upon recent developments and advances in the field of graph neural networks [17]. In particular, it shares similarities with the method proposed by Kipf [20]. However, our method incorporates new components, in particular the 1D convolutions, use of context, and curriculum-based training. To better understand the effects of the individual components of our network, we make changes (ablations and additions) to the original Kipf method to see how they affect performance. First, we noted that SwarmNet can be seen as a variant of only the decoder part of Kipf’s approach.

Method	5 Steps			40 Steps		
	Boids	Helbing	Chaser	Boids	Helbing	Chaser
Kipf’s GNN	0.4288 ± 0.0182	0.4374 ± 0.0179	0.1391 ± 0.0006	17.45 ± 1.00	19.61 ± 0.43	14.23 ± 0.05
LSTM	0.8992 ± 0.0098	1.2241 ± 0.0098	0.2013 ± 0.0020	41.20 ± 0.24	42.88 ± 0.75	126.3 ± 3.5
SwarmNet	0.2813 ± 0.0012	0.1000 ± 0.0050	0.0152 ± 0.0002	10.47 ± 0.91	3.855 ± 0.305	3.691 ± 0.042
SwarmNet (Context)	0.2338 ± 0.0024	0.0317 ± 0.0019	0.0152 ± 0.0002	2.778 ± 0.066	0.484 ± 0.023	3.691 ± 0.042

TABLE I
NORMALIZED MSE LOSS FOR SHORT TERM PREDICTION (LEFT, 5 STEPS) AND LONG TERM PREDICTION (RIGHT, 40 STEPS)

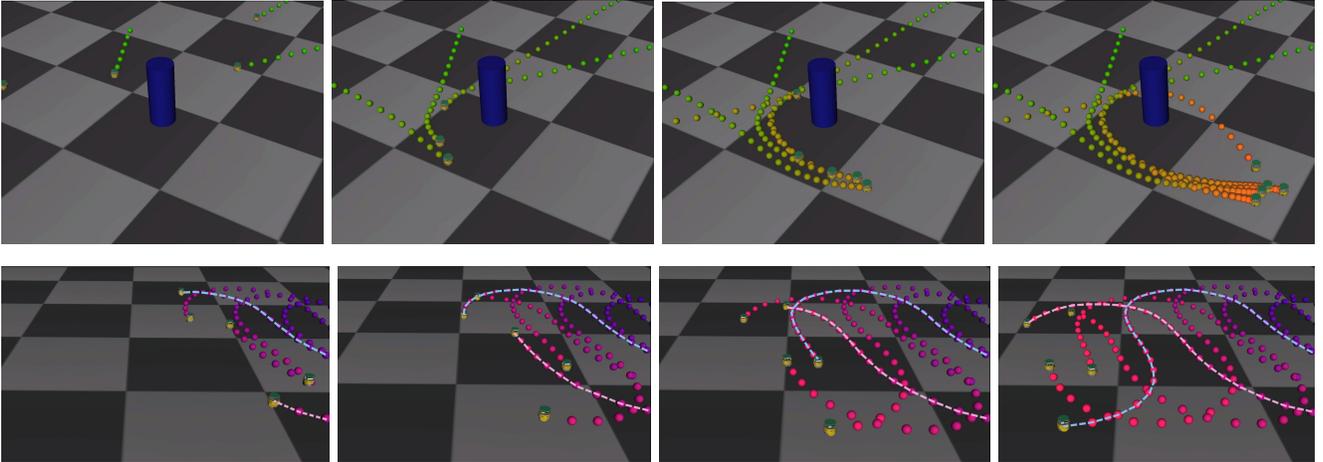


Fig. 9. Top sequence: Image sequence from the robot simulation of an ePuck MRS. All robots are controlled by a SwarmNet which was trained on the Boids model. The colored dots represent the states of the agents at different moments in time. Bottom sequence: A SwarmNet control policy implementing the chase behavior.

Hence, we used the decoder as a starting point and then performed repeated experiments in which we added (a) MLP layers for edge aggregation and context (b) 1D convolutions. Tab. II, shows the results of this comparison. We can see that the decoder of Kipf’s model, when combined with both 1D convolutions and context variables, yields MSE error values comparable to our results. The different neural network architectures shown in Tab. II can be seen as a (discrete) spectrum that shows the effects of gradually transitioning from the original Kipf GNN to our SwarmNet model. Our model still slightly outpaces Kipf’s decoder augmented with Conv1D and context. This last difference is due to the curriculum learning approach to training. For a qualitative comparison between SwarmNet predictions and the Kipf GNN predictions, see Fig. 10.

Method	5 Steps	40 Steps
Kipf’s GNN	0.4288 ± 0.0182	17.45 ± 1.00
Decoder	0.2654 ± 0.0070	3.639 ± 0.075
Decoder (Context)	0.2717 ± 0.0034	3.873 ± 0.090
Decoder+Conv1D	0.2584 ± 0.0040	3.916 ± 0.388
Decoder+Conv1D (Context)	0.2491 ± 0.0018	3.101 ± 0.046
SwarmNet (Context)	0.2338 ± 0.0024	2.778 ± 0.066

TABLE II
COMPARISON OF MSE ERROR ON BOIDS DATA OF SWARMNET, THE ORIGINAL KIPF MODEL, AS WELL AS DIFFERENT NEW MODELS THAT ARE CREATED FROM ABLATIONS AND ADDITIONS TO THE DECODER PART OF THE NETWORK IN [20].

D. SwarmNet as a Policy for Controlling Clone Swarms

As introduced in Sec. I, a trained SwarmNet can also be used to control a new swarm. The objective in this case would be to generate new behavior that is similar to that observed in the training set. To test this property, we used the output velocities generated by SwarmNet as control signals for a simulated swarm of robots. In particular, we are using a simulation of the ePuck robot. Fig. 9 shows the result of using a SwarmNet, trained on Boids data and the Chaser data respectively, as a control policy for the ePuck MRS. In both cases, generated behavior of the clone swarm reproduces the dynamics inherent to the training data as discussed before.

E. Uncertainty

Finally, we also investigated how well the SwarmNet⁺ extension can deal with uncertainty, noise and nondeterminism. In particular, we simulated both perceptual noise, as well as actuation noise. Perceptual noise was incorporated by adding a value sampled from a univariate normal distribution $\mathcal{N}(0, 1)$ to input for each dimension separately. Actuation noise is simulated by randomly dropping out neurons in test time. Fig. 11 depicts the stochastic outputs of our SwarmNet⁺ model for the Boids data. We can see in Fig. 11 (left) that the predictions now form envelopes according to the uncertainty at different time steps in the future. In general, the uncertainty appears to grow with larger prediction horizons. In the case of the red agent, the predictions slightly bifurcate around the obstacle. This

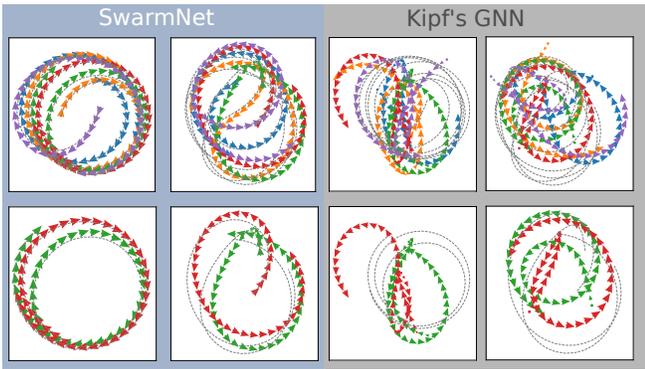


Fig. 10. Qualitative comparison of SwarmNet and the approach in [20]. Two samples of predicted trajectories (scattered arrows) against the ground truth trajectories (grey dashed lines) for a 5 agent system performing chasing motions. Top row: traces of all 5 agents. Bottom: traces of only two agents for visibility. Only the starting T_{seg} states are provided to the model, and the prediction is done consecutively to the end. All graphs use data from test set.

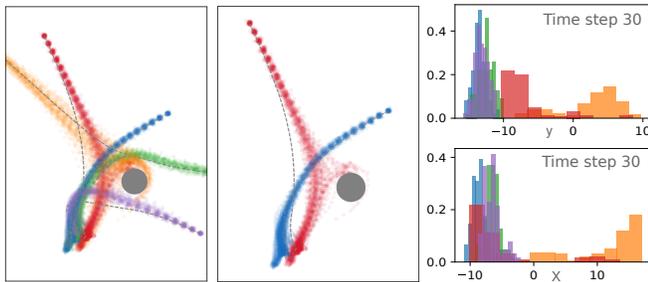


Fig. 11. Predictions for the *nondeterministic* behavior of a swarm with 5 boids in the presence of uncertainty. Left graph: the stochastic output predictions of the SwarmNet⁺ network for all agents. Middle: the predictions for only the blue and the red agent, for visual clarity. Right: The probability distributions over x and y coordinates of all agents at time step 30. We can see that the predictions for the red agent bifurcate to the left and right side of the obstacle.

clearly shows that the model is able to predict multiple potential futures of an agent (and the swarm) based on the inherent uncertainty of the task and environment. On the right, we see the probability distributions for discretized x- and y-coordinates of all agents. Again, some distributions are multimodal, which reinforces the insight that our predictions can produce multiple, diverse, and potentially conflicting future states.

V. CONCLUSION

In this paper, we presented a neural network architecture, called SwarmNet, which learns to predict and imitate behavior of an observed swarm. The network uses a combination of one-dimensional convolutions, graph convolutions, contextual inputs, along with a curriculum learning scheme to efficiently extract the swarm dynamics from positions and velocities of a set of agents. We showed that SwarmNet achieves high levels of prediction accuracy and that it can even be applied to nondeterministic and uncertain environments. For future work, we want to investigate how different application domains and tasks affect the sample complexity

of the method, and the effectiveness of a decentralized implementation of the network.

VI. ACKNOWLEDGEMENT

This research was funded by a gift from the Intel Corporation. Partial funding was also provided by the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement Number HR0011-18-2-0022. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. Approved for public release; distribution is unlimited.

REFERENCES

- [1] T. Arai, E. Pagello, L. E. Parker, *et al.*, “Advances in multi-robot systems,” *IEEE Transactions on robotics and automation*, vol. 18, no. 5, pp. 655–661, 2002.
- [2] M. G. Lagoudakis, M. Berhault, S. Koenig, P. Keskinocak, and A. J. Kleywegt, “Simple auctions with performance guarantees for multi-robot task allocation,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 1. IEEE, 2004, pp. 698–705.
- [3] C. W. Reynolds, “Steering behaviors for autonomous characters,” *Game Developers Conference*, pp. 763–782, 1999.
- [4] S. Schaal, “Is imitation learning the route to humanoid robots?” *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.
- [5] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, *et al.*, “An algorithmic perspective on imitation learning,” *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.
- [6] D. Helbing, I. Farkas, and T. Vicsek, “Simulating dynamical features of escape panic,” *Nature*, vol. 407, no. 6803, pp. 487–490, 2000.
- [7] 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.
- [8] M. Mesbahi and M. Egerstedt, *Graph theoretic methods in multiagent networks*. Princeton University Press, 2010, vol. 33.
- [9] S. Chernova and M. Veloso, “Multiagent collaborative task learning through imitation,” in *Proceedings of the fourth International Symposium on Imitation in Animals and Artifacts*, 2007, pp. 74–79.
- [10] E. Zhan, S. Zheng, Y. Yue, and P. Lucey, “Generative multi-agent behavioral cloning,” *arXiv preprint arXiv:1803.07612*, 2018.
- [11] M. Hüttenrauch, A. Šošić, and G. Neumann, “Guided deep reinforcement learning for swarm systems,” *arXiv preprint arXiv:1709.06011*, 2017.
- [12] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. Kumar, S. Koenig, and H. Choset, “Primal: Pathfinding via reinforcement and imitation multi-agent learning,” *arXiv preprint arXiv:1809.03531*, 2018.
- [13] M. F. Hocaoglu, “Weapon target assignment optimization for land based multi-air defense systems: A goal programming approach,” *Computers & Industrial Engineering*, vol. 128, pp. 681–689, 2019.
- [14] A. Kline, D. Ahner, and R. Hill, “The weapon-target assignment problem,” *Computers & Operations Research*, 2018.
- [15] I. Guvenc, F. Koohifar, S. Singh, M. L. Sichitiu, and D. Matolak, “Detection, tracking, and interdiction for amateur drones,” *IEEE Communications Magazine*, vol. 56, no. 4, pp. 75–81, 2018.
- [16] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [17] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, “Relational inductive biases, deep learning, and graph networks,” 2018. [Online]. Available: <http://arxiv.org/abs/1806.01261>
- [18] Y. Gal, “Uncertainty in deep learning,” Ph.D. dissertation, University of Cambridge, 2016.
- [19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.

[20] T. Kipf, E. Fetaya, K.-c. W. Max, and W. Richard, "Neural Relational Inference for Interacting Systems," 2018.