Control and Coordination of Multi-Drone Systems Using Graph Neural Networks

by

Parth Khopkar

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved April 2021 by the
Graduate Supervisory Committee:

Heni Ben Amor, Chair
Theodore Pavlic
Siyu Zhou

ARIZONA STATE UNIVERSITY

May 2021

ABSTRACT

Multi-robot systems show great promise in performing complex tasks in areas ranging from search and rescue to interplanetary exploration. Yet controlling and coordinating the behaviors of these robots effectively is an open research problem. This research investigates techniques to control a multi-drone system where the drones learn to act in a physics-based simulator using demonstrations from artificially generated motion data that simulate flocking behavior in biological swarms. Using these demonstrations enables faster training than approaches where the agents start learning from scratch. The Graph Neural Network (GNN) controller used for the drones learns an efficient representation of low-level interactions in the system, allowing the proposed method to scale to more agents than in training data. This work also discusses techniques to improve performance in the face of real-world challenges such as sensor noise.

To my parents, my first mentors

ACKNOWLEDGEMENTS

Several people have played extremely important roles in making sure I reach this milestone. First and foremost, I would like to thank Dr. Heni Ben Amor for his constant support and giving me an opportunity to work in his lab. It was my sheer luck that Dr. Siyu Zhou was finishing his doctorate under Dr. Ben Amor when I started my work. This work would not be possible without his prior work on Graph Neural Networks. I owe Dr. Zhou a huge debt of gratitude for his mentorship and constant support throughout my time working at the lab. I would like to thank Dr. Theodore Pavlic for his support and push to better my thesis.

I am grateful to Dr. Stephanie Gil for inviting me to her lab meetings in the first semester of my Masters from where I became a part of her lab and got firsthand experience on what research actually entails. Through my interactions with my lab mates at REACT lab - Sahil, Ninad, Sushmita, Calvin and Philip, I learnt the importance of having peers whom you can learn from and discuss your ideas with. I have been fortunate to be a part of many courses at ASU as a Teaching Assistant or course staff which allowed me to explore my interest in teaching along with gaining financial support. I would especially like to thank Professor Navabi for her continued trust in me and letting me be a part of the teaching staff for her courses every semester.

This acknowledgement would not be complete if I did not thank my friends - Aekansh, Paige, Siddhant and many more who have been my cheerleaders on many occasions. Without their support, it would have been near impossible for me to focus on work in times of uncertainty. Their friendship and encouragement have carried me throughout my degree.

TABLE OF CONTENTS

Page

LIST OF TABLES v

LIST OF FIGURES vi

CHAPTER

1 INTRODUCTION 1

2 GRAPH NEURAL NETWORKS AND HOW THEY MODEL MULTI-AGENT SYSTEMS 4

2.1 Graph Neural Networks 4

2.1.1 Operations in Graph Neural Networks 5

2.2 The Boids Model 7

3 LEARNING TO FLY BY IMITATING A SWARM 10

3.1 Imitation Learning 10

3.1.1 Training Process 11

3.2 Bullet Physics Based Simulator as a Testbed 12

3.3 Experiments and Results 14

4 SCALABILITY AND ROBUSTNESS 18

4.1 Reward Function as Evaluation Metric 18

4.2 Experiments on Scalability 19

4.3 Experiments on Robustness 20

4.3.1 Discussion: Reinforcement Learning after Imitation Learning 22

5 CONCLUSION AND FUTURE WORK 27

REFERENCES 29

APPENDIX

A CODE REPOSITORY 31

LIST OF TABLES

Table Page

3.1 Curriculum Training on Boids Trajectories 15

3.2 Parameter Comparison Between Boids Trajectories and Pybullet Environment 16

LIST OF FIGURES

Figure		Page
2.1	Concepts Used in Operations in Message Passing Neural Networks	7
2.2	The Different Boid Behaviors	8
3.1	Visualization of the Trajectories Generated by the Boids Model	11
3.2	Graph Neural Network Architecture	12
3.3	The PyBullet Simulator	13
3.4	Drone Controller System Architecture	15
3.5	Drones Flying in Pybullet Simulator	17
4.1	Scalability Results with Drones	20
4.2	Reward Received by Drones During Scaling	21
4.3	Comparison Between Different Controllers	23
4.4	Controllers Trained With and Without Noise in Tested in Pybullet Environment	24
4.5	Reward During Reinforcement Learning Training	25

Chapter 1

INTRODUCTION

The prevalence of multi-robot systems (MRS) has increased over the years across many domains. These include logistics in manufacturing plants and warehouses, environmental monitoring, agriculture and many more (Cortés and Egerstedt (2017)). The systems that are currently in use are often deployed in carefully controlled environments and work independently from humans. The behaviors that dictate the motion of these robots are pre-specified functions and may use AI-based techniques for perception. Lately, efforts have been made by researchers to bring the capabilities of these systems to more dynamic environments where uncertainty and hence risk, are increased. Efforts are also being made in the field of human-robot interaction (HRI) to address the challenges that we need to overcome when humans are in the loop (Kolling *et al.* (2016)).

With the advent of deep learning, the functions that specify robot behaviors can be approximated using neural network based architectures. The benefit of being able to do this approximation is that the data that the behaviors are learnt from can come from multiple sources rather than just relying on human experts. Many solutions using deep reinforcement learning (DRL) have been proposed where agents learn the desired behaviors using a reward signal. The issue with these approaches is that they are usually not sample efficient, with some requiring tens of millions of examples to learn. This problem is further exacerbated in multi-agent learning where the non-stationarity of the environment makes the task of reward assignment difficult (Foerster *et al.* (2016)). The imitation learning (IL) paradigm that we use in this work takes advantage of the fact that the learning process does not need to start from scratch.

We can take advantage of expert demonstrations to capture the interaction dynamics of a system. One such source of expert demonstrations, is the Boids model proposed by Reynolds (1987). This model simulates the flocking pattern of birds and other animals. A further improvement to this model was made by Reynolds (1999) which added predator and prey dynamics and elaborated on steering behaviors.

After the initial success of fully connected deep neural networks, recent breakthroughs in many domains have been possible because of neural network architectures that take a more structured approach. Here, the architecture depends on the structure of the data that the network operates on. The most prominent examples are convolutional neural networks (CNNs) that operate on spatially related data such as images and recurrent neural networks (RNNs) that operate on temporally related data such as time series. These architectures allow for a more expressive representation that enables efficient reasoning. The inductive biases in the structure help the learning algorithm to prioritize good solutions in the solution space by decreasing ambiguity. (Battaglia *et al.* (2018))

The class of structured neural networks that we use in this work are called graph neural networks (GNNs). As the name suggests, GNNs explicitly model the interactions that take place within data using a graph structure. This is useful to our problem since graph representations of multi-robot systems allow us to capture not only the interactions amongst the robots but also their interactions with entities present in their surroundings. In this formalism, the nodes of a graph represent static and non-static entities in the environment and the edges of the graph represent how different entities influence each other. In particular, we use SwarmNet, the GNN architecture proposed by Zhou *et al.* (2019) which explicitly models the different types of edges in the system, allowing it to act as an efficient controller by understanding the different interaction dynamics at play in the environment.

We use the terms multi-robot, multi-agent, and multi-drone systems interchangeably throughout this work. Chapter 2 introduces Graph Neural Networks and shows how they take advantage of the graph structure inherent in multi-agent systems. In Chapter 3, we show how a GNN model that learns only from demonstrations can be used as the backbone of a controller for multiple drones coordinating with each other in a physics-based simulator. Chapter 4 discusses the generalization capabilities of the controller which enables it to scale to more drones than present in the training data. For such a controller to work in the real world, it is paramount that it can act optimally even with perception noise. This aspect is addressed in Chapter 4 as well.

Chapter 2

GRAPH NEURAL NETWORKS AND HOW THEY MODEL MULTI-AGENT SYSTEMS

2.1 Graph Neural Networks

Graph neural networks (Scarselli *et al.* (2008)) are a class of function approximators that operate on graph-based data structures while explicitly modeling the interaction dynamics of the entities present in the graphs. Graphs as a structure have high expressive power which enables modeling of problems across many domains. Previous works have used graph networks in machine learning paradigms for applications such as physical scene understanding (Sanchez-Gonzalez *et al.* (2018)), predicting chemical properties of molecules (Gilmer *et al.* (2017)), predicting traffic on roads (Cui *et al.* (2019)), and an application that is closely related to the work in this thesis, modeling the interactions in multi-agent systems (Kipf *et al.* (2018)). What makes GNNs attractive for these tasks are the properties of permutation invariance and locality. Graph neural networks are a generalization of convolutional neural networks (that operate on Euclidean spatial data) to non-Euclidean data which can be modeled as graphs. Using this explicit modeling allows GNNs to take advantage of inductive biases in the problem structure. This can help speed up learning and enable more efficient reasoning through a better internal representation than traditional fully connected Neural Networks.

In the context of multi-agent systems (MAS), graphs can be used to model the connections between the agents and other entities present in the environment. A graph $\mathcal{G}$ which consists of a set of nodes $\mathcal{N}$ (which represent the agents) and a set

of edges $\mathcal{E}$ (which represent the connections between the agents) can be written as $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. Here, $\mathcal{E} = \{(i,j)|i,j \in \mathcal{N},\ i \neq j\}$ is the set of edges where (i,j) represents the influence from agent i to agent j through interactions. The neighborhood of a node i is denoted by $\mathcal{N}_i$ which is a set of nodes connected to i, i.e $\mathcal{N}_i = \{j \mid (j,i) \in \mathcal{E}\}$

2.1.1 Operations in Graph Neural Networks

The most basic form of operations in a GNN proposed by Scarselli *et al.* (2008) is as follows:

$$\boldsymbol{v}_i = f(\boldsymbol{V}_i, \boldsymbol{E}_i) \tag{2.1}$$

$$\boldsymbol{o}_i = g(\boldsymbol{v}_i) \tag{2.2}$$

$\boldsymbol{v_i}$ is the state of node i and $\boldsymbol{o_i}$ is the output of a read-out function. Eq. (2.1) is applied till convergence starting from an initial state and then passed through the read-out function in Eq. (2.2). $\boldsymbol{V}_i$ represents the set of node states and $\boldsymbol{E}_i$ represents the edge states of the edges connected to node i. f and g are both approximated by neural networks. It is important to note that this version of the GNN update equations did not allow for updating edge states.

More recently Battaglia *et al.* (2018) proposed a generalized version of the update equations which allow for edge states to be updated in addition to the nodes. It also includes a global level state vector $\boldsymbol{u}$. Here $\boldsymbol{v}_i$ represents the state vector for node i and $\boldsymbol{e}_{ij}$ represents the state vector for edge (i,j).

$$\boldsymbol{e}'_{ij} = \phi^e(\boldsymbol{e_{ij}}, \boldsymbol{v}_i, \boldsymbol{v}_j, \boldsymbol{u}) \tag{2.3}$$

$$\boldsymbol{v}'_i = \phi^v(\bar{\boldsymbol{e}}'_i, \boldsymbol{v}_i, \boldsymbol{u}) \tag{2.4}$$

$$\boldsymbol{u}' = \phi^u(\bar{\boldsymbol{e}}', \bar{\boldsymbol{v}}', \boldsymbol{u}) \tag{2.5}$$

$\phi^{\{e,v,u\}}$ functions are usually approximated by neural networks. $\boldsymbol{e}'_{ij}$, $\boldsymbol{v}'_i$ and $\boldsymbol{u}'$ are the updated edge, node and global states respectively.

$$\bar{\boldsymbol{e}}'_i = \rho^e(\boldsymbol{E}'_i) \tag{2.6}$$

$$\bar{\boldsymbol{e}}' = \rho^v(\boldsymbol{E}') \tag{2.7}$$

$$\bar{\boldsymbol{v}}' = \rho^u(\boldsymbol{V}') \tag{2.8}$$

$\rho^{\{e,v,u\}}$ functions here are generally chosen to be simple reduction functions such as summation or average. $\bar{\boldsymbol{e}}'_i$, $\bar{\boldsymbol{e}}'$ and $\bar{\boldsymbol{v}}'$ are the aggregated messages which are the result of these functions.

In our work, we use a variant of GNN which is known as a Message Passing Neural Network (MPNN). More specifically, the GNN update operations used by Zhou *et al.* (2019) in the SwarmNet architecture are used, for which the update equations are as follows:

$$\boldsymbol{e}_{ij} = \phi^e(\boldsymbol{v}_i, \boldsymbol{v}_j), \tag{2.9}$$

$$\bar{\boldsymbol{e}}_i = \psi^{\bar{e}}(\sum_{j \in \mathcal{N}_i} \boldsymbol{e}_{ji}), \tag{2.10}$$

$$\boldsymbol{v}'_i = \phi^v(\boldsymbol{v}_i, \bar{\boldsymbol{e}}_i), \tag{2.11}$$

Eq. 2.9 is the edge update equation where each edge pulls information from the two edges that it is connected to. After the edge is updated, each node aggregates the messages received from its in neighbors in Eq. 2.10. After edge aggregation, the node update step shown in Eq. 2.11 updates each node according to the current state of the node and the aggregated messages received from its neighbors. Fig. 2.1 illustrates the concepts shown in these equations. For a more complete survey of Graph Neural Networks, we refer the reader to Zhou *et al.* (2018).

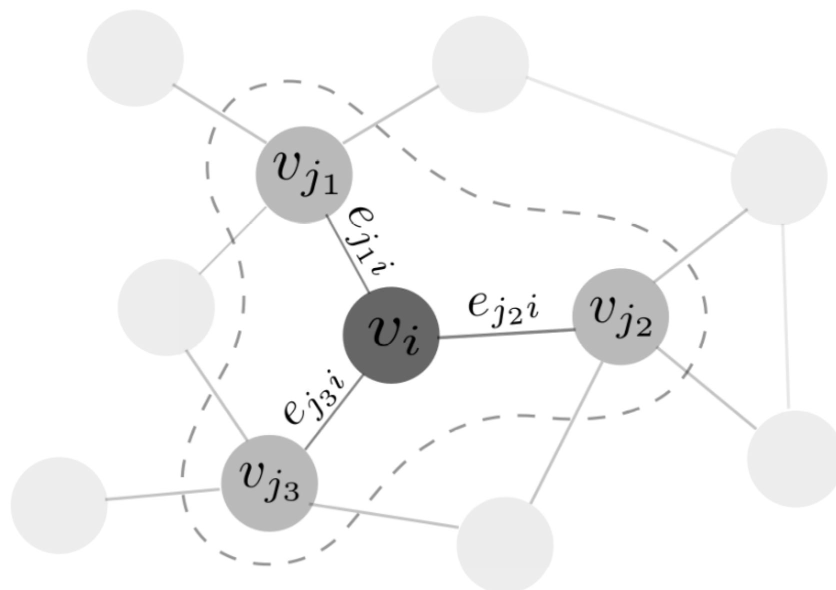


Figure 2.1: Illustration of concepts used in operations in message passing neural networks

2.2 The Boids Model

Reynolds (1987) proposed the Boids model as a way to simulate the aggregate motion of a flock of animals (such as birds) where the behavior of each member of the flock is the result of interactions with their neighbors. The high-level behavior of such a system can be described as emergent, which means that the behavior of the entire system depends on the relationships each of the members have with each other rather than their individual properties. This model was developed as a way to simulate a natural flock in computer animation as an alternative to scripting the behavior of individual members of the flock. The original paper referred to these simulated bird-like or "bird-oid" objects as "boids". In this model, different behaviors influence the acceleration of each boid at a particular time step. These behaviors (shown in Fig. 2.2) are separation, alignment, cohesion, goal-seeking, and obstacle avoidance. The superposition of the acceleration given by each of these behaviors produces the

final acceleration that a boid will have. This combination can either be a summation of the different behaviors or can be determined by prioritized acceleration allocation where acceleration from each component is treated in priority order and added into an accumulator. We use the first approach in our work. While other models of particle-based simulation like Helbing *et al.* (2000), Balch and Hybinette (2000) and Vicsek *et al.* (1995) exist, it can be argued that the flocking model of Boids is the most generic one and subsumes the interaction dynamics in these models.

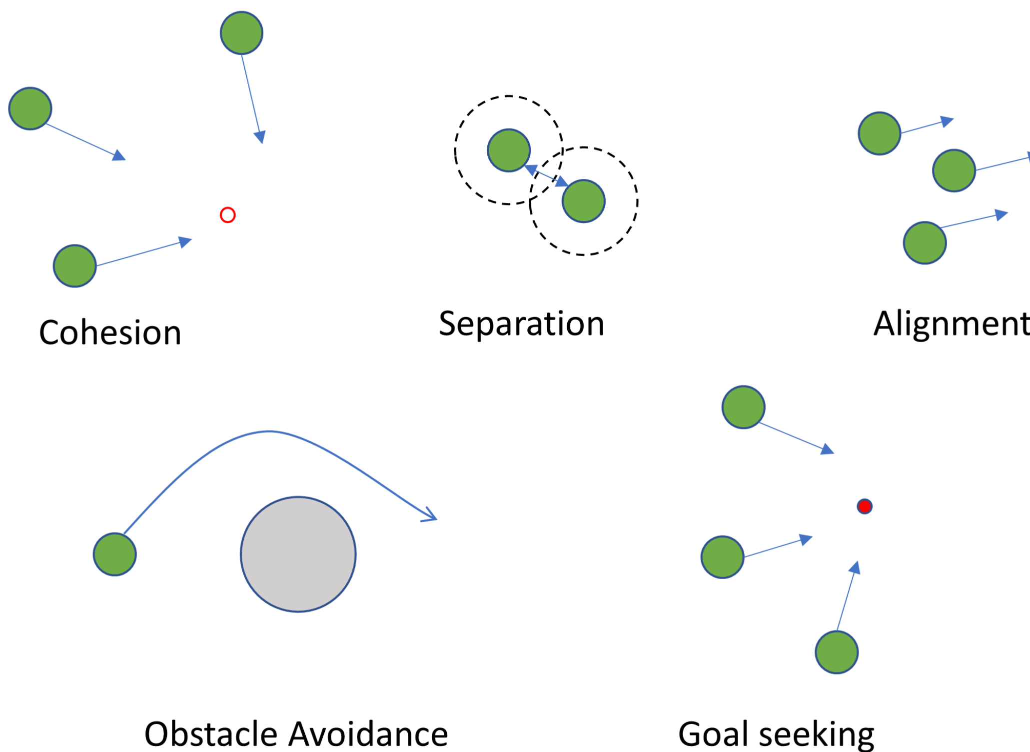


Figure 2.2: The different boid behaviors: The green circles represent boids, the grey circle represents an obstacle and the red dot is the goal location

We consider three types of entities in our system – boids, obstacles, and goals. Since the boids are the only non-static entities in the system, we assign separate function approximators (represented by ϕ^e in Eq. 2.9) for the interactions from all entities to boids: $\phi^e_{o\to b}$, $\phi^e_{g\to b}$ and $\phi^e_{b\to b}$. Here $x \to b, x \in \{o, g, b\}$ denotes the directed

edge from an obstacle (o), goal (g) or another boid (b) to the central boid of the neighborhood respectively. We expect the GNN to learn a unique function for each type of interaction. This separation of functions is what allows for a more efficient internal representation and is similar to the filter functionality in convolution neural networks (CNNs).

Chapter 3

LEARNING TO FLY BY IMITATING A SWARM

3.1 Imitation Learning

Developing a distributed controller for multi-robot motion and behavior is a challenging task. Early approaches relied on handcrafting algorithms to program the robots to complete the desired objective. More recently, control strategies that involve the robots learning the desired behavior have emerged where the learning takes place either through the use of expert demonstrations or self-exploration guided by a signal that tells the agents how well they are doing. This work uses the first approach with the expert demonstrations being provided by the Boids model discussed in the previous chapter. We train a GNN controller to imitate the behaviors demonstrated by the Boids model. The end goal here is to directly use a controller trained on these trajectories to provide desired velocity inputs to a PID controller for drones. This allows us to focus on learning high-level behaviors while letting the PID controller take care of the finer details of drone flight dynamics.

The trajectories generated by the Boids model are of the form $T \times N \times D$ where T is the length of the time series, N is the number of entities in the environment and D is the dimension of the space in which these entities exist. A graph which represents how the entities influence each other is associated with each step of a trajectory. The state of each entity is given by the vector $s_i(t) = [x_i(t), \dot{x}_i(t)]$ where $x_i(t)$ is the coordinate vector of an entity and $\dot{x}_i(t)$ is the velocity vector. The square brackets denote concatenation. In our experiments, we use trajectories from boids that exist in a two-dimensional world, which means that $s \in \mathbb{R}^4$. Fig 3.1 shows a sample of the

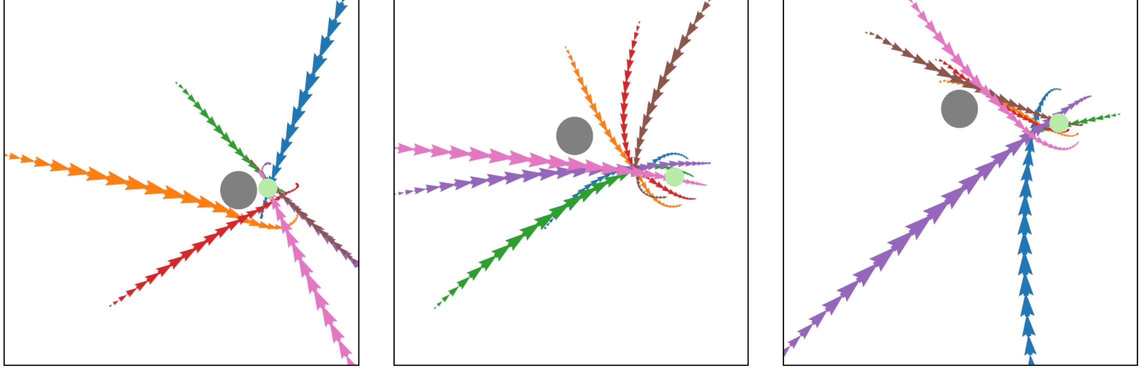


Figure 3.1: Visualization of the trajectories generated by the Boids model

trajectories generated by the Boids model.

3.1.1 Training Process

In order to train our GNN controller to imitate the behaviors from the trajectories, we use the supervised loss function given in Eq. 3.1 . Here, $s_i^*(t)$ denotes the ground truth trajectory.

$$L = \frac{1}{2DNT_s} \sum_{t=1}^{T_s} \sum_{i=1}^{N} (s_i(t) - s_i^*(t))^2 \tag{3.1}$$

$$\bar{L} = \frac{1}{2DN(T-1)} \sum_{t=1}^{T-1} \sum_{i=1}^{N} (s_i^*(t+1) - s_i^*(t))^2 \tag{3.2}$$

Since we sample states discretely using the Boids model, the error in Eq. 3.1 is dependent on the sampling frequency. To alleviate this dependency, we normalize the error using the normalization factor $\bar{L}$ which is the mean squared error (MSE) of the state vectors between two consecutive steps in ground truth data. The normalized error is given by $L_{norm} = \frac{L}{\bar{L}}$. Fig. 3.2 shows the network architecture of our graph neural network.

When using the trained GNN model as a controller, it is asked to predict the

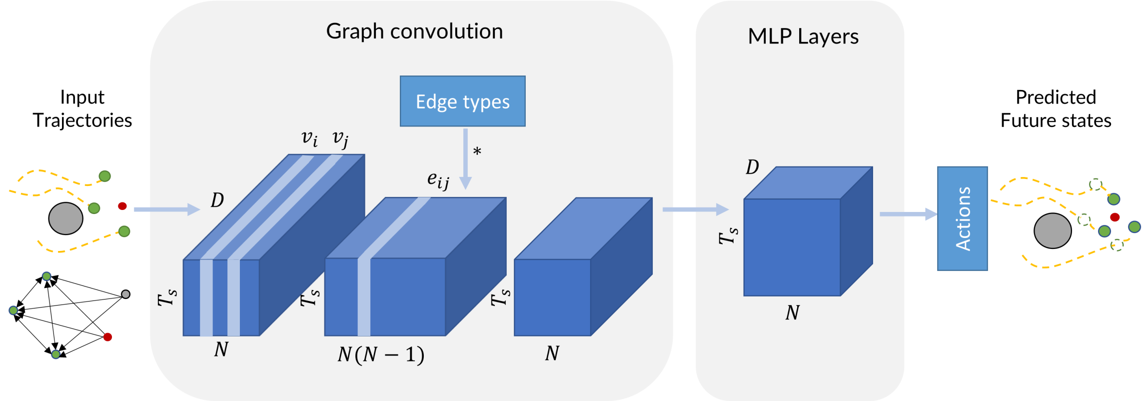


Figure 3.2: The architecture of the Graph Neural Network

next step given the current state of the entities in the system, where the predicted velocity serves as the input to a PID controller. In the work by Zhou *et al.* (2019) on the SwarmNet framework, multi-step prediction capability for the GNN model was achieved by gradually training the model for an increasing number of prediction steps. This is necessary to enable the model to reduce the single step prediction error and learn collision avoidance, both of which are crucial to our use case.

3.2 Bullet Physics Based Simulator as a Testbed

We use a Bullet physics-based simulator [1] for multi-agent quadcopter control introduced by Panerati *et al.* (2021) to test our GNN controller's effectiveness in an environment that is closer to the real world than simple boid trajectories. Fig. 3.3 shows a screenshot from the simulator. This simulator was developed to provide a standardized environment for comparing reinforcement learning results for a multi-quadcopter scenario. We take advantage of the fact that reinforcement learning's paradigm of observation-action-reward is similar to the feedback loop in control theory and test our imitation learning based controller using this simulator as a testbed. The

[1] `https://github.com/utiasDSL/gym-pybullet-drones`

use of Bullet physics engine provides support for realistic collisions and aerodynamics effects. The environments used in this simulator are parallelizable and can run in GUI or headless mode unlike simulators such as AirSim (Shah *et al.* (2018)) which has tight integration between rendering and its dynamics. Other simulators like Flightmare (Song *et al.* (2020)) do not provide multi-agent Gym-like APIs like the simulator we use. These APIs streamline the development process and facilitate quicker iteration between solutions. The simulator also provides access to a PID controller to which we supply the velocity that each drone should target. The dynamics for the drones that are used in the experiments are based on Bitcraze's Crazyflie 2.x nano-quadrotor [2] .

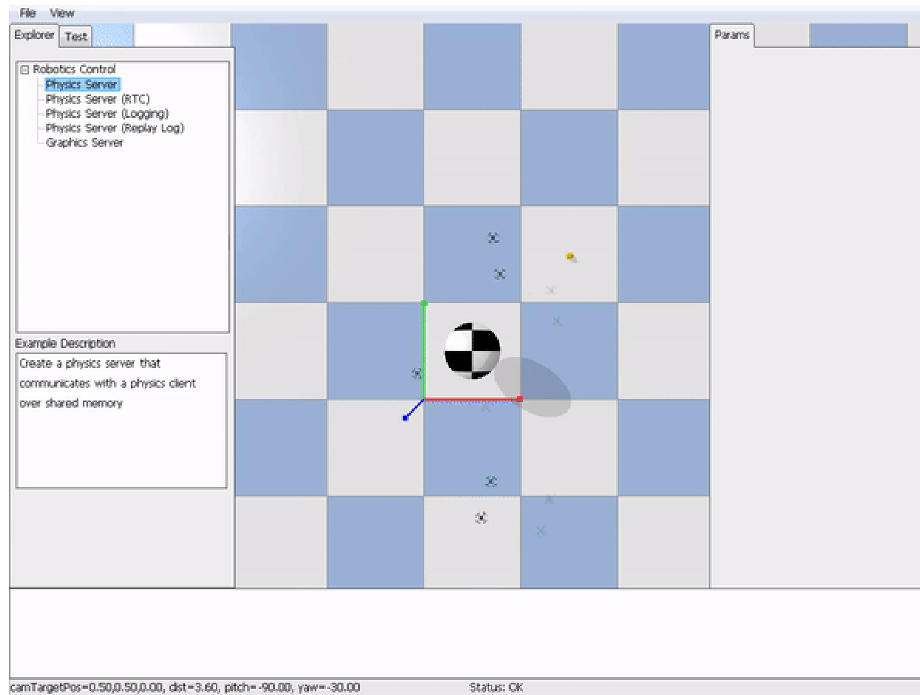


Figure 3.3: A screenshot of the PyBullet simulator

[2] `https://www.bitcraze.io/documentation/hardware/crazyflie_2_1/crazyflie_2_1-datasheet.pdf`

3.3 Experiments and Results

We start by training the GNN controller to learn to imitate the behavior demonstrated in the trajectories from the Boids model. Our goal is to directly use this controller with the simulator and thus it is necessary to match the parameters in demonstrations to what the controller expects to see in the simulator. Table 3.2 lists the parameters for the boids trajectories and the PyBullet environment. The value of a key parameter - the stepping frequency for the simulator is set to 240 Hz to make the simulation as realistic as possible. Setting the same high frequency for the boids trajectories means that we would need more steps (higher value of T) in a single trajectory since it is important to ensure that the trajectories generated by the Boids model are long enough such that all the boids can reach the goal from their starting positions. This would result in a higher number of steps being required for the GNN controller to learn about the interactions that result in motion patterns in the demonstrations. Our trials with matching the simulation frequency between the two environments resulted in the GNN not being able to learn the correct behaviors due to the size of the data involved. Choosing a low simulation frequency (5Hz) meant that the value of T was substantially reduced but the trajectories still contained the behaviors we would like our GNN to learn.

In the Boids model, the the boid is considered to be a point particle with the size attribute being a "comfortable distance" that the boid should maintain from other physical entities rather than a hard-shell value. Thus, to avoid collisions in the PyBullet environment, it is important to set the value of size in the Boids trajectories to be much larger than the actual size of the drones. The values of maximum speed and acceleration were picked by empirical analysis and adjusted by observing the performance of the GNN controller with the drones. Fig. 3.4 presents the architecture

Epochs	Prediction steps	Training loss at end	Normalized test error for single step prediction
80	1	0.0844	0.58
40	2	0.0510	0.27
20	4	0.0767	0.25
10	8	0.1442	0.25
5	16	0.3960	0.26
160	1	0.0100	0.06

Table 3.1: Curriculum training on Boids trajectories: The GNN controller is trained on an increasing prediction step horizon so that it can learn to imitate the motion patterns in the trajectories. The training loss specified here is the loss at the end of training epochs with specified prediction steps. The ultimate goal is to reduce the test error for single-step prediction which is why we train again for single-step prediction at the end.

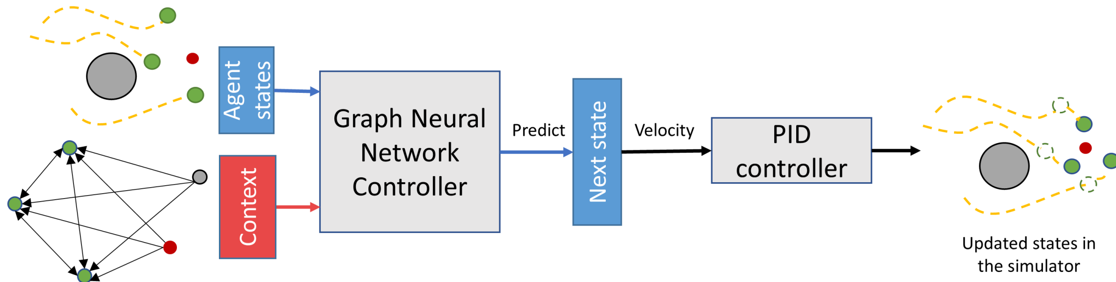
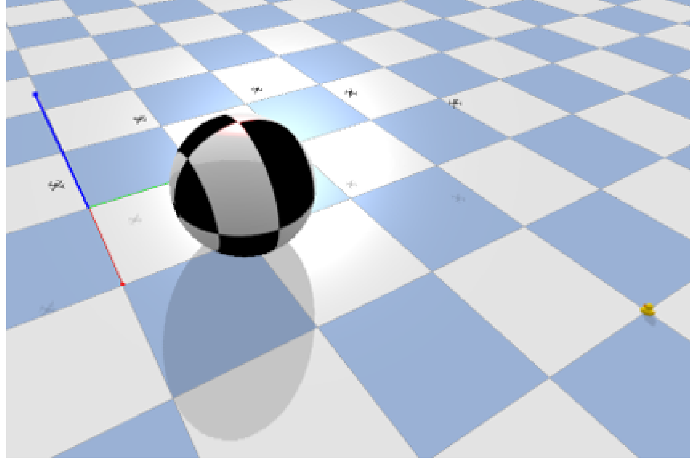


Figure 3.4: System architecture of the drone controller

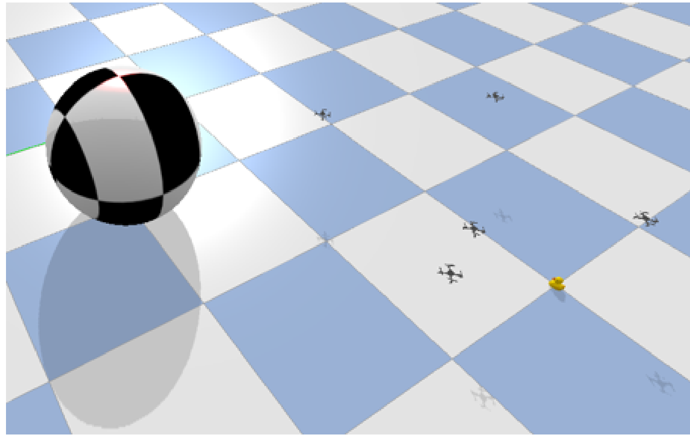
of the full controller and Fig. 3.5 shows the drones in the PyBullet simulator being controlled by the GNN controller. Table 3.1 shows the training method followed to train a controller using 2048 trajectories of 35 steps each. These trajectories contain five boids and a single obstacle which are randomly placed at initialization.

Parameter	Boids trajectories	PyBullet environment
Agent size (radius)	1 unit	0.16 units
Obstacle size (radius)	1.5 units	1.5 units
Max speed	10 units per unit time	unbounded
Max acceleration	5 units per unit time^2	unbounded
Stepping frequency	5 Hz	240 Hz

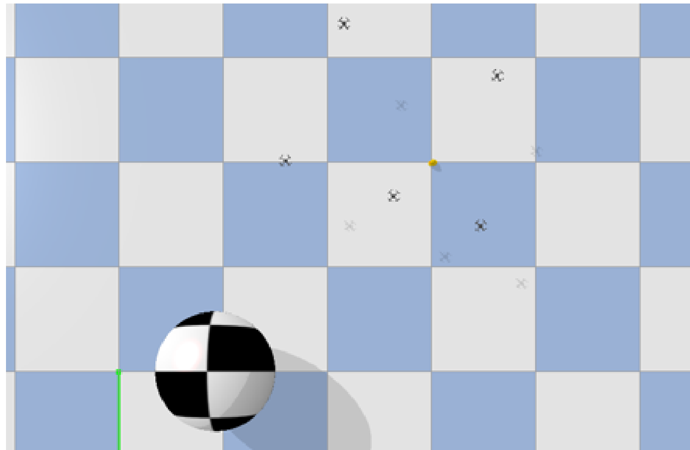
Table 3.2: Parameter comparison between Boids trajectories and PyBullet environment



(a) The drones maneuver to avoid obstacles



(b) The drones reach the goal and hover near it



(c) Top view of the drones hovering near the goal

Figure 3.5: The drones flying in the PyBullet simulator using the GNN controller

Chapter 4

SCALABILITY AND ROBUSTNESS

We saw in the previous chapter how a GNN controller trained only on demonstration data can be used to control multiple drones in a physics-based simulator, giving an insight into the effectiveness of GNNs. In this chapter, we present further results on their generalization capabilities as we test the scalability and robustness of the trained controller.

4.1 Reward Function as Evaluation Metric

To evaluate the performance of the controller as we test its scalability and robustness, we need a metric that can gauge the relative performance for different scenarios or training methods. Since the task of the drones is to navigate towards the goal as efficiently as possible while maintaining a tight formation and avoiding collisions amongst each other or with the obstacle, we define a reward function with three components that capture these aspects:

$$r_d(d_{ij}) = \begin{cases} -c, & d_{ij} < 2a_d \\ -\rho d_{ij}, & d_{ij} \geq 2a_d \end{cases}, \; r_o(d_{io}) = \begin{cases} -c, & d_{ij} < a_d + a_o \\ 0, & d_{ij} \geq a_d + a_o \end{cases}, \; r_g(d_{ig}) = \frac{\gamma}{d_{ig} + \alpha} + \beta \tag{4.1}$$

Here, r_d is a function of d_{ij} (distance between a drone i and another drone j), r_o is a function of d_{io} (distance between an drone i and the obstacle o), and r_g is a function of d_{ig} (distance between a drone i and the goal g). $c > 0$ is a large cost for collision, a_d and a_o are the radii of the drone and the obstacle. $\rho, \alpha, \beta, \gamma > 0$ are

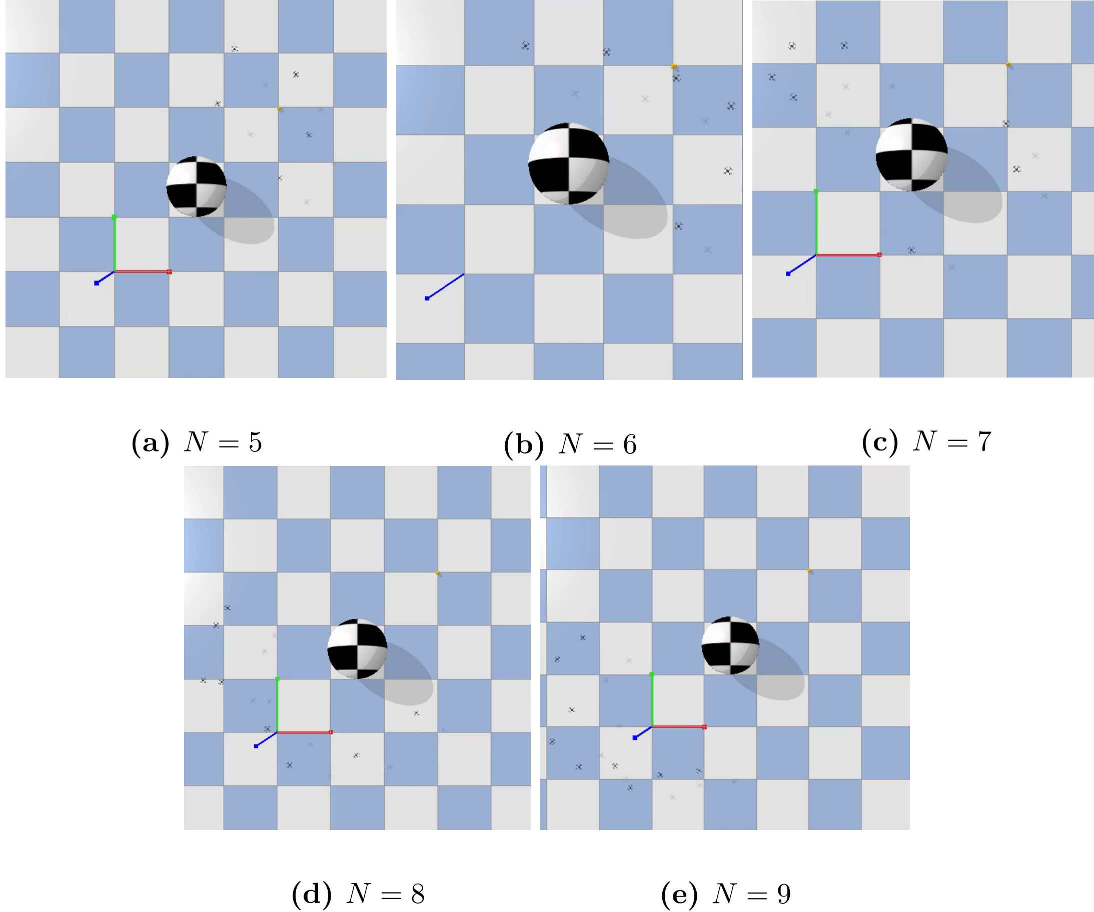
coefficients that shape the relative importance of the subtasks. Finally, the reward collected in the environment from all the three sources is combined to get the final reward for each agent at a particular time step:

$$r_i = r_o(d_{io}) + r_g(d_{ig}) + \sum_{j \in \mathcal{N}_i} r_d(d_{ij}) \tag{4.2}$$

4.2 Experiments on Scalability

The scalability of the GNN controller is tested by deploying it on more drones than the number of boids in the trajectories it was trained on. This scaling up is possible since the size of the graph is not a part of the update equations presented in Chapter 2 (Eq. 2.9 - 2.11). The results are shown in Fig. 4.1 where a controller trained for $N = 5$ drones is tested on $N = 5, 6, 7, 8, 9$ drones. Fig. 4.2 shows the normalized reward received by the drones for the instances shown in the figures.

As expected, we see a decline in the reward received as the number of drones is increased. When $N = 6$, the drones reach the goal from their starting position but do not hover near the goal in a regular pattern as seen for $N = 5$. When $N = 7$, the drones do not reach the goal but still do better than $N = 8$ and $N = 9$ where the drones congregate together but are unable to reach the goal. The drones still move around together and show cohesion, separation and alignment behaviors of the Boids model. More importantly, the drones do not collide with the obstacle or each other, meaning that the control is still stable when scaled up. The most likely explanation for why the drones are not able to reach the goal is that with a higher number of drones, the arbitration between the different behaviors is not scaled up. This can be attributed to the poor extrapolation abilities of the multi layer perceptrons (MLPs) that encode and decode information as part of the GNN controller. A possible solution to this problem is using padded trajectory data with a varying number of boids with a maximum equal to target upper limit on number of drones.



(a) $N = 5$ **(b)** $N = 6$ **(c)** $N = 7$

(d) $N = 8$ **(e)** $N = 9$

Figure 4.1: Controller trained with $N = 5$ boids trajectories tested with more drones

4.3 Experiments on Robustness

Until now, we have used a centralized controller in our experiments that has access to the perfect states for all entities present in the environment. However, our GNN controller is decentralizable since a controller running on each drone only needs access to the state of drones in its neighborhood and other entities in its vicinity. We consider a scenario where a decentralized controller running on each of the drones can perceive its state perfectly but receives noisy readings for other entities due to a noisy communication channel or due to perception error. Randomly sampled Gaussian noise is introduced at each time step in the observations each drone receives about the

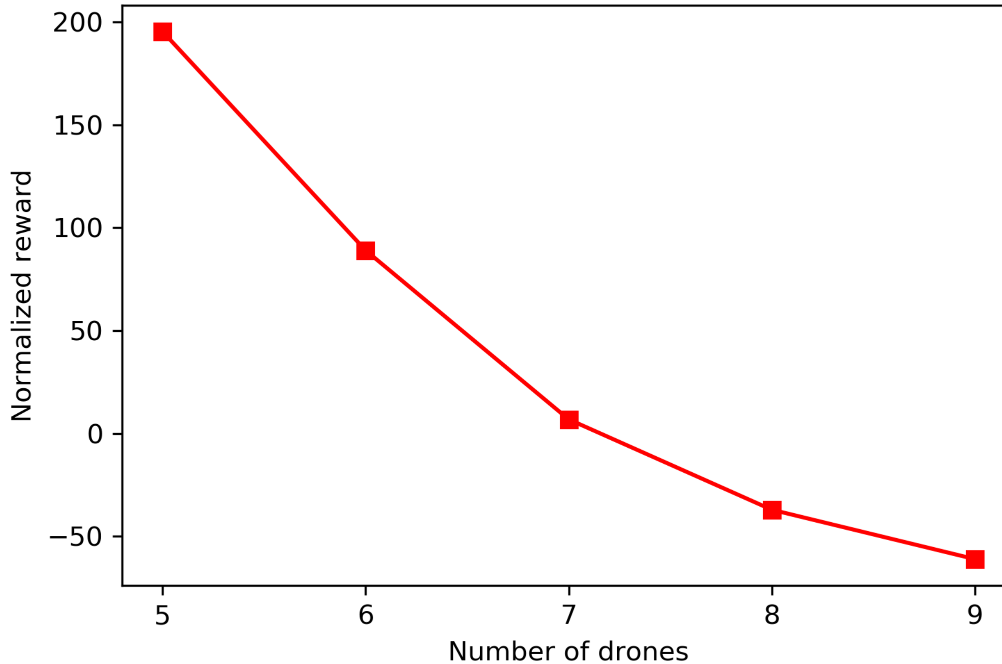


Figure 4.2: Normalized reward received by the drones for a controller trained on $N = 5$

position and velocity of other drones as well as other static entities in the environment, such as the goal and obstacles.

To make the controller robust to noise, we add randomly sampled Gaussian noise to the Boids trajectories while training. Since training becomes more difficult on these noisy trajectories, we limit the amount of noise introduced during training to a random sample from $\mathcal{N}(0, 0.1)$ for each data point. Fig. 4.3 compares the results between the controller trained with and without noise, averaged over 10 trials for each data point. We find that while the controller trained without noise does better when the noise is low, it fails when the noise becomes high. Whereas, the controller trained with noise learns to take more conservative paths, avoiding obstacles and thus decreasing the number of collisions even at high noise. Although the second controller

does take a performance hit since the drones take longer to reach the goal. Given this difference in performance between the two controllers, it would be appropriate for the drones to switch control strategies when noise is high rather than just using one controller all the time. Fig. 4.4 shows the two controllers being tested in the PyBullet simulator.

To determine if the trend in behavior seen for controller trained without noise is introduced due to the approximation done by GNN or is inherent in the Boids model, we also test the Boids model as a controller in noisy environments. We found that the Boids model shows a similar dip in performance when the noise is increased. This dip is not as sharp as the controller trained without noise but still sharper than the controller trained on noisy trajectories. This result also goes to show that the GNN is faithfully able to generalize to the trajectories learned from the Boids model.

4.3.1 Discussion: Reinforcement Learning after Imitation Learning

To combat the issue of noise, we also tried reinforcement learning as a solution using the reward function discussed earlier to train the GNN controller in an environment with noisy observations after it has been trained using imitation learning. The idea here is to take advantage of the function separation property of GNNs. Since we aim to reduce the chance of collisions amongst the drones as well as between drones and obstacles, we tune the parameters for $\phi^e_{o \to b}$ and $\phi^e_{b \to b}$ MLPs during reinforcement learning while keeping the parameters for $\phi^e_{g \to b}$ MLP locked. For our experiments, we chose the Proximal Policy Optimization (PPO) (Schulman *et al.* (2017)) algorithm due to its versatility and ability to prevent drastic change in parameters during

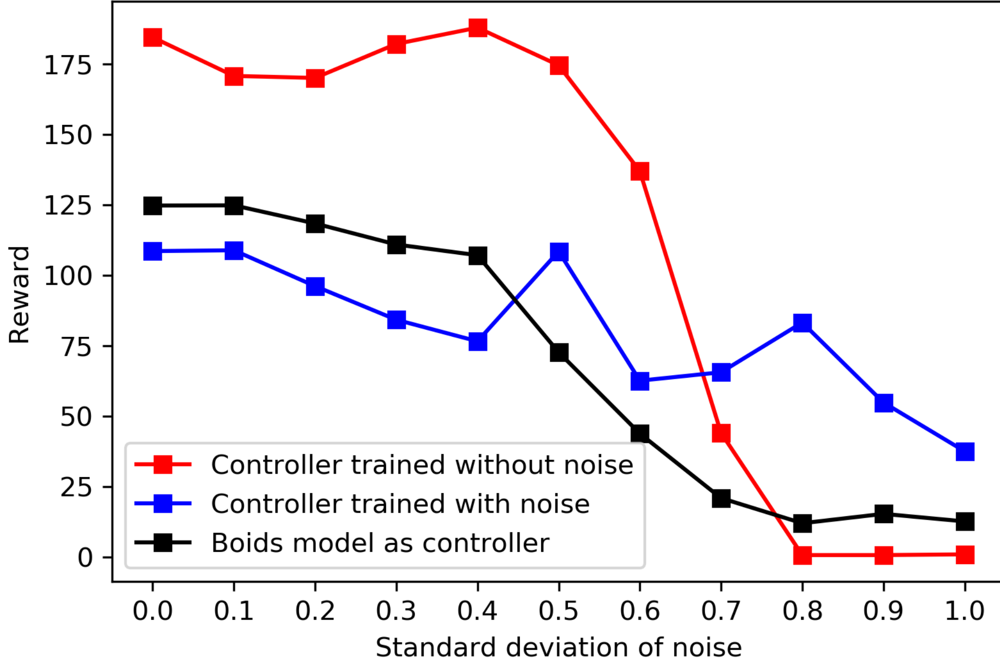


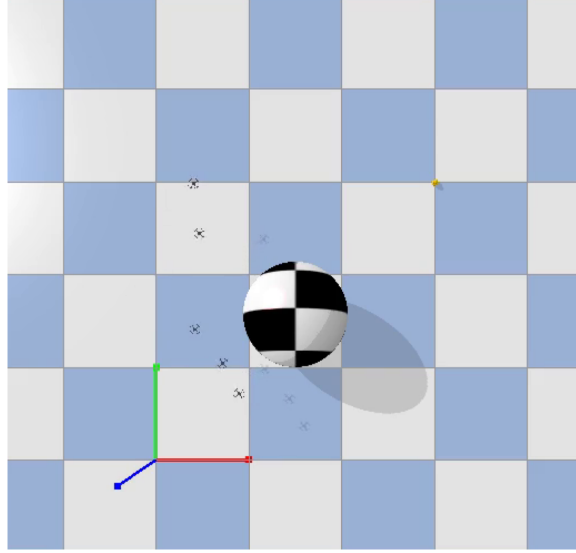
Figure 4.3: Comparison between controllers: The controller trained without noise degrades completely, crashing at early time steps but the controller trained with noise gives a more consistent performance even at higher noise. The Boids model performs similarly to the controller trained without noise.

on-policy learning. The objective function for PPO is as follows:

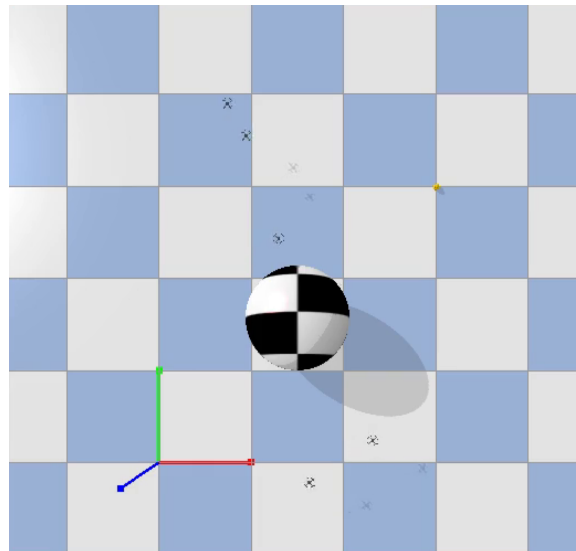
$$L(s, a, \theta', \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)} A^{\pi_{\theta'}}(s, a),\ \ \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta'}}(s, a)\right), \tag{4.3}$$

Here $\pi_{\theta'}(a|s)$ is the old policy and $\pi_\theta(a|s)$ is the current policy. $A(s, a)$ is the advantage function which is an estimation of how much better it is to take action a in state s rather than randomly selecting an action from policy π. ϵ is a hyperparameter which dictates how far the new policy is allowed to go away from the current policy.

We pre-train the value function for PPO through self-play using the imitation learning trained GNN as the control policy. During pre-training, only the added MLP



(a) Imminent collision for controller trained without noise



(b) An instance where controller trained with noise avoids collision for the same initial conditions

Figure 4.4: Controller trained without and with noise tested in the PyBullet simulator at $\mathcal{N}(0, 0.8)$ test noise

for value function is trained. After pre-training, the $\phi^e_{g \to b}$ MLP is still kept locked while other parameters are free to be trained. We found that this approach to training

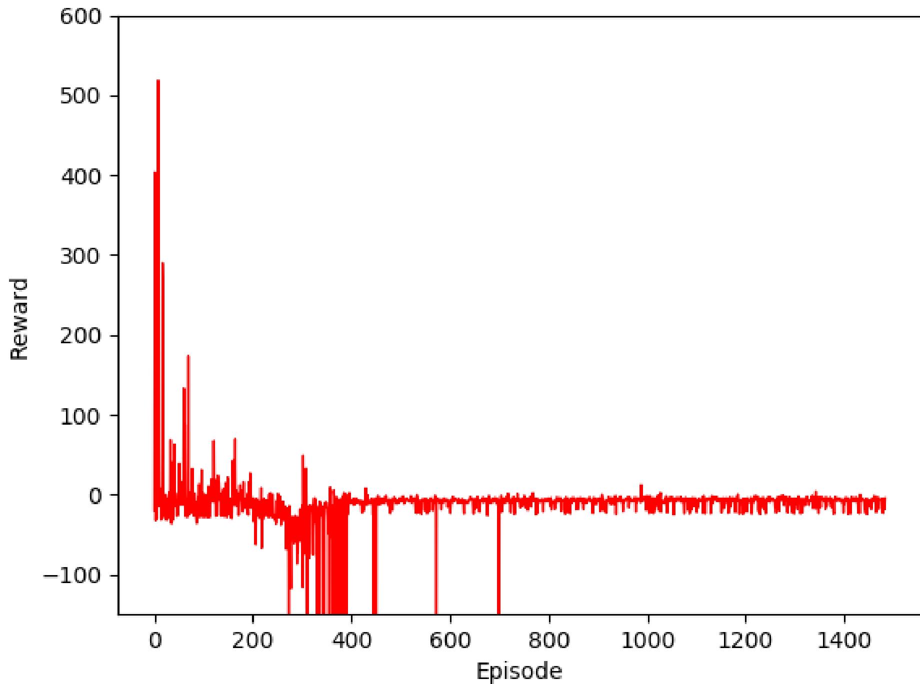


Figure 4.5: Reward during reinforcement learning training

the controller using RL after training it with Imitation Learning results in a rather quick degradation in performance with RL working to erode the behaviors learned during imitation learning. This results in a sharp decline in the reward obtained per episode (shown in Fig. 4.5) which is initially high after IL training. This leads to the conclusion that directly switching to reinforcement learning after training using imitation learning is not ideal for our use case. This is because:

1. Since reinforcement learning erodes behaviors learned during training on expert demonstrations, it is counter-intuitive to let the controller forget and learn new behaviors according to the reward signal.

2. For use cases such as flying drones which is non-ergodic, it is imperative that the initial training performance is not poor so as to avoid failure when training

in the real world.

An intuitive reason why RL training gives poor results is that the supervised learning loss and the reward signal during RL are two fundamentally different mathematical signals. The switch to the latter from the former results in the degradation in the behavior that is described earlier. This result is in line with the findings in works from the Learning from Demonstration (LfD) literature such as Gao *et al.* (2018) and Hester *et al.* (2017). These works propose using a unified reinforcement and imitation learning algorithm that uses a joint objective function for demonstrations and experiences collected through self-play. We leave investigating the use of these techniques to our application for future work.

Chapter 5

CONCLUSION AND FUTURE WORK

We showed that Graph Neural Networks can be used to leverage the problem structure inherent in multi-agent systems to effectively control and coordinate between multiple agents. This thesis presented results on using a GNN-based controller for a multi-drone system in a physics-based environment. Through the use of expert demonstrations, GNNs can learn efficient internal representations that enable robust control capabilities. The generalization capabilities of a GNN can be used to scale to more agents than the model is exposed to during training. This upscaling shows a graceful degradation in performance and avoids catastrophic events such as collisions. We also presented results on the robustness of the controller, including a discussion of techniques to decrease susceptibility to perception noise that is bound to be present in real-world environments.

In the future, we would like to implement the controller on actual drones to test the effectiveness of the proposed method in the real world. A promising direction is testing and adapting the GNN controller for specific application scenarios (such as search and rescue). This would also open up the possibilities of adding perception capabilities (such as vision) and exploring communication techniques that work well with the decentralized controller. The Boids model is just one source of expert demonstrations that can be used for training. Other models that are more specialized for individual task domains may perform better for different applications. Finally, we would also like to explore the possibility of boosting performance during reinforcement learning by using a mix of demonstration data and self-exploration guided by a combined objective function. It is worth exploring if this method can prevent the erosion of

behaviors that happens due to the dissimilar nature of supervised loss and reward signals during training as seen in the previous chapter.

REFERENCES

Balch, T. and M. Hybinette, "Social potentials for scalable multi-robot formations", in "Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)", vol. 1, pp. 73–80 (IEEE, 2000).

Battaglia, P. W., J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks", arXiv preprint arXiv:1806.01261 (2018).

Cortés, J. and M. Egerstedt, "Coordinated control of multi-robot systems: A survey", SICE Journal of Control, Measurement, and System Integration **10**, 6, 495–503 (2017).

Cui, Z., K. Henrickson, R. Ke and Y. Wang, "Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting", IEEE Transactions on Intelligent Transportation Systems **21**, 11, 4883–4894 (2019).

Foerster, J. N., Y. M. Assael, N. De Freitas and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning", arXiv preprint arXiv:1605.06676 (2016).

Gao, Y., H. Xu, J. Lin, F. Yu, S. Levine and T. Darrell, "Reinforcement learning from imperfect demonstrations", ArXiv **abs/1802.05313** (2018).

Gilmer, J., S. S. Schoenholz, P. F. Riley, O. Vinyals and G. E. Dahl, "Neural message passing for quantum chemistry", in "International Conference on Machine Learning", pp. 1263–1272 (PMLR, 2017).

Helbing, D., I. Farkas and T. Vicsek, "Simulating dynamical features of escape panic", Nature **407**, 6803, 487–490 (2000).

Hester, T., M. Vecerík, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo and A. Gruslys, "Learning from demonstrations for real world reinforcement learning", ArXiv **abs/1704.03732** (2017).

Kipf, T., E. Fetaya, K.-C. Wang, M. Welling and R. Zemel, "Neural relational inference for interacting systems", in "International Conference on Machine Learning", pp. 2688–2697 (PMLR, 2018).

Kolling, A., P. Walker, N. Chakraborty, K. Sycara and M. Lewis, "Human interaction with robot swarms: A survey", IEEE Transactions on Human-Machine Systems **46**, 1, 9–26 (2016).

Panerati, J., H. Zheng, S. Zhou, J. Xu, A. Prorok, A. P. S. U. of Toronto Institute for A Studies, V. I. for Artificial Intelligence and U. Cambridge, “Learning to fly - a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control”, ArXiv **abs/2103.02142** (2021).

Reynolds, C. W., “Flocks, herds and schools: A distributed behavioral model”, in “Proceedings of the 14th annual conference on Computer graphics and interactive techniques”, pp. 25–34 (1987).

Reynolds, C. W., “Steering behaviors for autonomous characters”, in “Game developers conference”, vol. 1999, pp. 763–782 (Citeseer, 1999).

Sanchez-Gonzalez, A., N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell and P. Battaglia, “Graph networks as learnable physics engines for inference and control”, in “International Conference on Machine Learning”, pp. 4470–4479 (PMLR, 2018).

Scarselli, F., M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, “The graph neural network model”, IEEE transactions on neural networks **20**, 1, 61–80 (2008).

Schulman, J., F. Wolski, P. Dhariwal, A. Radford and O. Klimov, “Proximal policy optimization algorithms”, arXiv preprint arXiv:1707.06347 (2017).

Shah, S., D. Dey, C. Lovett and A. Kapoor, “Airsim: High-fidelity visual and physical simulation for autonomous vehicles”, in “Field and service robotics”, pp. 621–635 (Springer, 2018).

Song, Y., S. Naji, E. Kaufmann, A. Loquercio and D. Scaramuzza, “Flightmare: A flexible quadrotor simulator”, (2020).

Vicsek, T., A. Czirók, E. Ben-Jacob, I. Cohen and O. Shochet, “Novel type of phase transition in a system of self-driven particles”, Physical review letters **75**, 6, 1226 (1995).

Zhou, J., G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li and M. Sun, “Graph neural networks: A review of methods and applications”, arXiv preprint arXiv:1812.08434 (2018).

Zhou, S., M. Phielipp, J. Sefair, S. Walker and H. Amor, “Clone swarms: Learning to predict and control multi-robot systems by imitation”, in “2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019”, IEEE International Conference on Intelligent Robots and Systems, pp. 4092–4099 (Institute of Electrical and Electronics Engineers Inc., 2019).

APPENDIX A

CODE REPOSITORY

Experiments and other code: `https://github.com/parthkhopkar/drone_swarm`