UNIVERSITY OF CALIFORNIA,
IRVINE

Improving the Robustness of Drone Swarm Control Systems with Graph Learning

THESIS

submitted in partial satisfaction of the requirements
for the degree of

BACHELOR

in Electrical Engineering

by

Jorge De La Torre Martín

2023

Contents

List of Figures

List of Tables

Page

List of Abbreviations

AI Artificial Intelligence. 4

AvC Average Cohesion. 13

CoM Center of Mass. 12, 13, 16, 18, 19

DAgger Dataset Aggregation. 23

GCS Ground Control Station. 5

GNNs Graph Neural Networks. iv, viii, 3, 4, 7–12, 15, 17, 19, 20, 28, 29

GSO Graph Shift Operator. 20

L Loss Function. 21, 22

LoC Level of Cohesion. iv, v, 12, 13, 24, 25

ML Machine Learning. 9, 29

NN Neural Network. 19, 21–23

SI Swarm Intelligence. iv, 1, 4–7, 28

UAS Unmanned Airborne System. 1, 3, 4, 12, 19

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my thesis advisor, Shih-Yuan Yu, for his invaluable support and encouragement throughout this research project. I am really grateful for his guidance, insights, and feedback throughout the writing of this thesis.

I am also grateful to the faculty members of the Electrical Engineering and Computer Science Department, for their knowledge, expertise, and academic support, which have allowed me to be involved in this thesis project and jump into the research world.

I would like to express my thankfulness to my family and friends for their help and motivation, which have sustained me throughout this challenging process and without whom it would have been psychologically harder.

I am immensely thankful to all who have contributed to make this project and thesis possible, in particular to the UCI Professor Mohammad Al Faruque, their support has been invaluable to my academic journey.

Thank you.

ABSTRACT OF THE THESIS

Improving the Robustness of Drone Swarm Control Systems with Graph Learning

By

Jorge De La Torre Martín

Bachelor in Electrical Engineering

University of California, Irvine, 2023

We propose a novel approach to control a swarm of drones. Leveraging ***Graph Neural Networks*** (GNNs), our approach aims to improve the robustness of the drone swarm system, making the swarm accomplish tasks in adverse and disturbing conditions. In our project, one example of such harsh conditions can be when one or more agents are biased or compromised. Related works exist leveraging GNNs to decentralize the global controllers and bring many benefits to the swarm control system for drones. However, whether applying GNNs can improve the robustness still needs to be explored. Therefore, our objective is to investigate this problem and verify if using GNNs can enhance the robustness of the systems.

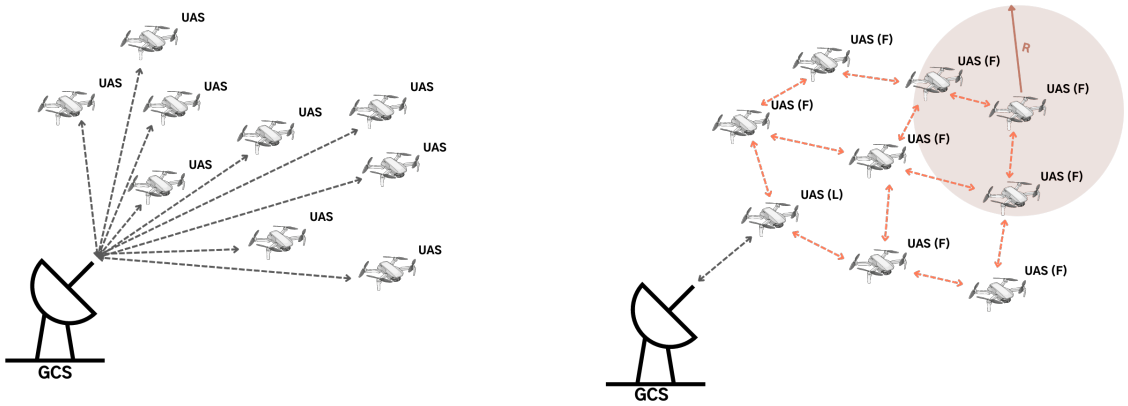
Chapter 1

Introduction

Unmanned Airborne System (UAS) reduces life risks by performing tasks that manned systems cannot do. The UAS development has attracted research attention due to its high capacity and applications, as shown in military and commercial uses, such as photography, cinematography, precision agriculture, surveillance security, natural disaster recovery, and search and rescue operations [2]. An individual UAS can have limited capabilities and thus cannot perform large-scale tasks. Many natural examples exist, such as bees coordinating to complete a task or flocks of migrating geese coordinating in an efficient flight pattern to achieve their migration. Therefore, ***Swarm Intelligence*** (SI) has become increasingly popular, where drones are coordinated collaboratively to achieve their mission. Applying SI for a UAS swarm can have advantages, such as faster task completion due to the parallelization or execution of collaborative tasks. However, the development of these systems is still in its early stages.

Several research challenges exist in developing SI for UAS. However, the main focus is, as introduced by [4], on ***Robustness*** (the swarm has the capability of completing the tasks even being affected by failures), ***Flexibility*** (the swarm can be rearranged and readjusted

to form different patterns), or ***Scalability*** (the swarm can grow and shrink in size, and still be able to complete its task). These challenges may also lead to the issue of what controller architecture might be the most suitable, as depending on the application's specific requirements, it may be necessary to choose between different communication structures. As introduced by [2] and [3], there are some common approaches used, such as Infrastructure-based Swarm communication architecture (centralized, at the cloud level) represented in Figure 1.1a, or Flying Ad-hoc NETwork (FANET) which supports higher autonomy as the follower agents, of the leader-follower structure, make their own decision in a distributed manner (at the edge level) as in Figure 1.1b.



(a) Centralized communication architecture. (b) Decentralized communication architecture.

Figure 1.1: UAV Swarm communication architectures.

On the one hand, centralized controllers are based on the idea that a single entity controls the swarm, exchanging real-time information with each agent and making decisions for the swarm by itself. Indeed, centralized controllers make it possible to access and manage all the swarm information globally, enabling them to achieve more precise tasks.

On the other hand, decentralized controllers are systems where decision-making is distributed across all the agents. Every agent gathers local information from sensors and actuators (local

observation) from its neighbors, enabling these systems to be used in more flexible environments. The problem arises when a UAS swarm grows in scale, making a decentralized controller more beneficial than a centralized one. Decentralized control may become even more critical when operating in restricted communication areas where global communication may seem impossible.

Graph Neural Networks can be an excellent tool for accomplishing these tasks, as they can represent swarm data as a graph. Indeed, they are specifically designed to process graph-structured data so that they can have a better understanding of complex scenarios and relationships between agents, as well as be able to handle them in a very efficient manner. Besides that, they can be trained to clone global policies via imitation learning, as introduced by [16], bringing together the beneficial points of the centralized controller but performing their actions from a decentralized perspective. For the sake of this study, it all comes down to flocking tasks in which dynamic communication networks play their role. As mentioned before, there are many examples of natural swarms, such as geese taking advantage of collaborative control to carry out flocking tasks more efficiently, making it more attractive to keep exploiting these areas unstudied.

Recent works have made important progress in the usage of GNN techniques to improve information exchanges and study procedures to avoid using the ***Global Navigation Satellite System*** through implementing vision-based controllers [5], or even more, developing decentralized path planing controllers as in [7]. Even so, few studies have approached the robustness problem and how using GNNs could enhance performance in the area of UAS swarm.

In our work, we focus on decentralized control architectures for controlling the flocking behavior of a group of flying drones. The overall idea or architecture can be illustrated in

Figure 1.1b. This type of architecture uses local information about the neighbors to output some control over the agents. As a general introduction to our approach, we aim to assess how different types of controllers affect the robustness of the swarm while carrying out flying tasks. As will be later defined, the main concept of robustness for swarm application is a system's capability to perform even if failure involves it. Therefore, we aim to compare and demonstrate how GNNs controllers can help enhance a swarm's robustness while flying. As already stated, as a graph learning model, this project needs some inputs to extract its outputs. The model's inputs will be all the sensor state data gathered from each drone, and the generated outcomes will be an action prediction. This work offers to enhance the robustness and enable better Swarm Intelligence by embedding ***Artificial Intelligence*** (AI) into communication controllers. More precisely, it is proposed to integrate GNNs into UAS swarms in a decentralized manner, making it possible to adapt the flocking formation to the faults, avoiding disconnections or other desirable swarm behavior. Doing so can help the swarm have a more robust behavior and better agent communication.

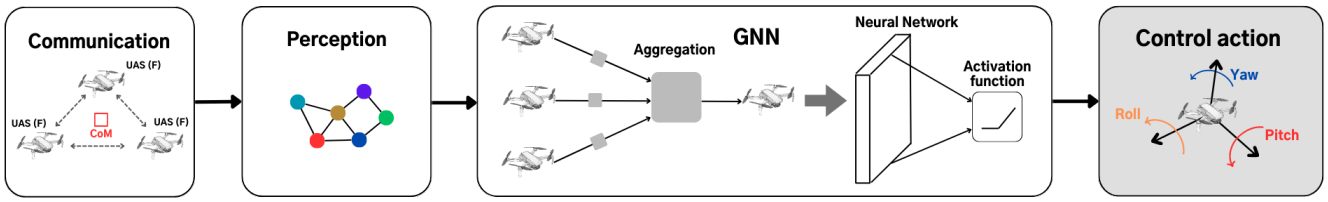


Figure 1.2: Graph learning architecture for UAV swarm controller.

Chapter 2

Background

2.1 Swarm intelligence

As a first introduction to the concept of swarm control, the concept of a swarm should be defined and explained why it is so relevant nowadays. A swarm is often seen as a cluster of agents moving together or just being together and standing by. However, this might not be the exact definition of a swarm. A swarm is a group of agents coordinated individually to carry out a particular global task, without a central controller or ***Ground Control Station*** (GCS) having to send a unique control action for all the individuals. Indeed, it is a decentralized collaborative behavior where everyone shares the same mindset and objective.

Some models have been developed to make SI possible. Craig Reynolds was a pioneer in this area of study, introducing what is known as the Boids model, where all the agents should follow some rules to complete their global task, more precisely, the flocking task. This model was first introduced in 1987 by [14] as an approach to simulate the flocking behavior of natural swarms, such as birds. It is mainly based on three rules defining the flocking

behavior that every agent should follow: separation, cohesion, and alignment. The Boids model can also be applied to situations where goal-seeking is needed, an additional rule for the model that makes the agents move toward a specified target. More recently, there have been improvements on those former rules, introducing new rules that target other aspects of the flocking behavior, such as prey and predator interaction [1]. The three rules defined by [14][6] to approach the flocking problem

- Separation: keep a minimum distance between agents so everybody has enough room to fly, avoiding collisions.
- Alignment: steer towards the average path of the neighbors so that every single agent will be heading in the same direction.
- Cohesion: move toward the average position of the neighbors, keeping the group together and avoiding disturbances in the swarm.

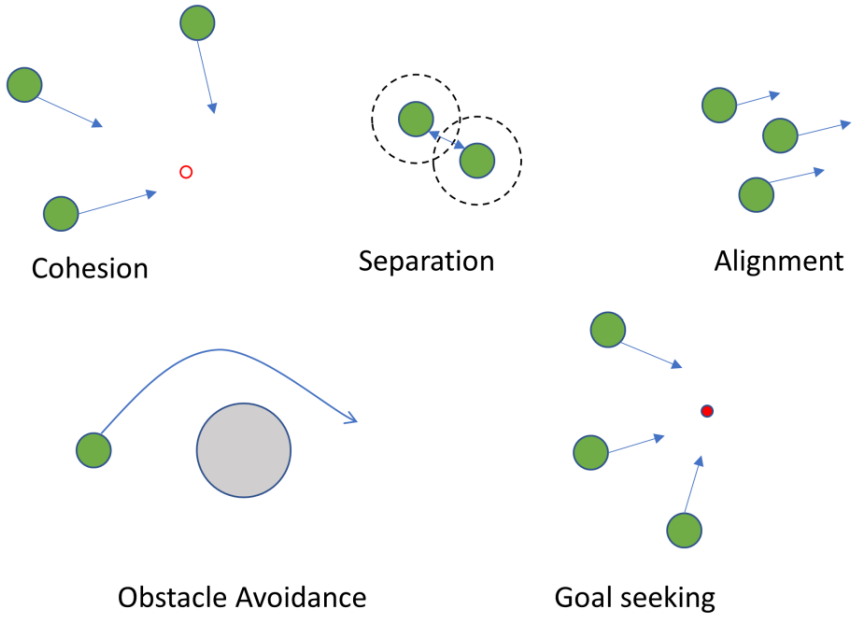


Figure 2.1: Representation of rules for rule-based SI models, introduced by [6].

Studying this natural phenomenon in some animals, such as birds, has inspired new intellectual brains to replicate this phenomenon in the drone area of study. It is known that drone swarms can help enhance numerous daily activities, or even more, military activities such as drone attacks. In fact, in the case of military activities, swarms can be a very powerful and harmful tool since they can spread out in a broader area, collaborating all together. Therefore, it can be inferred that SI is a potent tool where all the agents can make their own choices but always follow a specific set of rules that makes them fly in a structured and organized manner.

2.2 Robustness

In the context of drone swarm control, robustness can be defined as the ability to demonstrate vigor, strength, firmness, or even more, the capability of performing a task in the presence, or not, of failure. Moving this property into a system, the robustness of a system is determined by its ability to tolerate perturbations that may arise in the system. These disturbances or perturbations in the system may come from multiple sources, such as environmental factors, sensor noise or failure, disconnections or connection delays, etc. How these perturbations are handled and their assessment is essential for the system to carry out its task without any breakdown.

2.3 Graph Neural Networks

By definition, GNNs processes graphical data, making node, edge, or graph-level predictions. Graph Neural Networks can generally be used for several purposes, mainly when the information is gathered in graphs. The applications of GNNs can be adjusted to node level predictions, where node features are updated based on the training; edge level predictions,

where links between nodes are the ones being updated; or graph level predictions, where updated node features are pooled to generate a global feature that represents the entire graph.

Graph Neural Networks can also be expressed or understood as a message-passing neural network with a message-passing function, a node update function, and a readout function, as illustrated in Figure 2.2. Graphs consist of two components: vertices, usually represented as V, and edges, represented as $\mathcal{E}$. Vertices depict a set of nodes containing data or features that the GNNs will process to generate predictions. Edges represent the relationships or links between nodes which can be unidirectional or bidirectional. Usually, the way of reproducing graphs more mathematically is by using an adjacency matrix named "A." This is a set of numbers arranged in rows and columns to form a rectangular array. This set of numbers in the adjacency matrix represents the relation between nodes within a graph. Hence, its dimension is (NxN). Moreover, a second matrix defined on graphs consists of the feature matrix, named "F," and dimensions (NxF). Moreover, problems solved by GNNs can be categorized into four groups

- Node classification: the task is to predict the node embedding for every node in the graph.
- Link prediction: the task is to understand and predict the relationships or connections between nodes in the graph.
- Graph classification: the task is to classify the whole graph into different categories. Given a graph, some features could be extracted from it.
- Clustering: the task is to detect the different clusters formed within a graph.

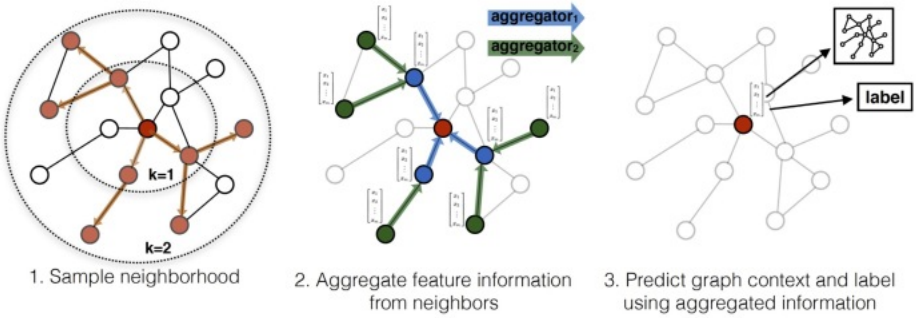


Figure 2.2: Illustration of message passing in GNNs between multi-hop neighbors and node level prediction.

Graph Neural Networks are a novel ***Machine Learning*** (ML) technique that is still under research. Many recent studies have been focusing on this topic, such as [22, 8, 9, 10] leveraging GNNs to produce Spatio-temporal embeddings, used in this work, which can potentially lead to an enhancement in the safety of Autonomous Driving Systems and yield better transferability. Moreover, there have been other studies regarding security assurance using GNNs [20, 19, 23, 18] which can result in an interesting topic for the drone swarm control field to be studied in future works. In essence, Graph Neural Networks are a good solution to many industrial applications [21, 12] with or without graph classification and clustering [17].

2.3.1 Graph Neural Network Usage on Swarm Intelligence

In the drone swarm control domain, GNNs can help to enhance the robustness of a swarm of drones for several reasons. Graph Neural Networks can pass information in multi-hop exchanges, which makes each agent gather more information about the global group. Graph Neural Networks, due to its prediction task, can detect when a drone is not working as it should, also known as fault detection. By the behavior of a particular agent not operating on its correct path, the GNNs can learn that these variations are due to a failed agent.

Graph Neural Networks can learn how to deal with this failing drone so it won't affect the swarm performance by making a simple change in the trajectory of the swarm, attempting to isolate the failed agent. The decision-making is carried out by the GNNs, which is the main advantage of using GNNs for this type of task. Graph Neural Networks can make predictions very quickly based on the position and velocity of the agents themselves, as well as the observations of the neighbors, making the response time much shorter due to their intrinsic capability of storing the data provided at the time of training. This storing feature results in the ability to remember certain behaviors, making the model adjust better to the problem. Hence, the robustness of the swarm increases as the model learns to avoid efficiently failing agents on the swarm. Besides that, Graph Neural Networks can be trained in the presence of disturbances so that the model learns how to respond in certain situations with acting perturbations.

Chapter 3

Problem formulation

In this work, we advocate demonstrating the possibility of cloning an optimal policy from a Ground Truth model into GNNs to enhance the robustness of the flocking behavior. In the work, we focus on the follower side of the leader-follower structure represented in Figure 1.1b, which is the side with the highest autonomy and ability to make decisions based on local information. The Ground Truth model is based on a centralized architecture where full access to global information is assumed. Hence, the objective of the proposed GNN control system attempts to choose among all the possible outputs the most optimal one, known as an optimal policy $u_i^*(t) = \pi(s_i(t))$. In such a case, the optimal policy depends on the global information of the swarm obtained from the Ground Truth model.

Consider a graph formed by a set of N agents distributed on the space forming a set of vertices of a graph V=$\{1,2,3,...,N\}$, and a set of edges $\mathcal{E}$. For the sake of our work, consider a dynamic networked system where these edges represent the flow of information among agents, depending on the range of communication of a certain sensor with other agents. In this way, the neighborhood of an agent can be defined as those agents positioned within the radius of communication of one agent of the swarm, indeed $R < 4m$. Therefore, the set of

neighbors of a node i at t time steps, is defined by (3.1)

$$\mathcal{N}_i(t) = \{j \in V : (i,j) \in \mathcal{E}\} \tag{3.1}$$

Moreover, consider the dynamics of each agent i described by its position vector $s_i(t) = [x_i(t), y_i(t)]^T \in \mathbb{R}^2$, relative position with respect to their neighbors $r_{ij}(t) = [x_j(t) - x_i(t), y_j(t) - y_i(t)]^T \in \mathbb{R}^2$, a velocity vector $v_i(t) = [v_{x,i}(t), v_{y,i}(t)]^T \in \mathbb{R}^2$, and an acceleration vector $u_i(t) = [u_{x,i}(t), u_{y,i}(t)]^T \in \mathbb{R}^2$. For each drone in the network, the acceleration $u_i(t) = \dot{v}_i(t)$ can be considered controllable so that changes in the velocity of each agent i can be achieved. The parameter t is considered a time-discrete index representing consecutive time samples of interval T.

Our primary focus is to compare the performance of GNNs against the Boids model. Indeed, we have demonstrated the ability of the new controller to outperform the Boids model, handling the problem of flocking behavior. To do so, one of the metrics from the literature which measures the flying performance of a UAS swarm is from [6], where they used the average of the rewards as their primary metrics. One of the main rules of the flocking problem is flock centering, where all the agents attempt to stay close to their nearby flock-mates [13]. As our metrics, we have defined flock centering as the ***Level of Cohesion*** (LoC), calculated by averaging the distances between every flying drone and the ***Center of Mass*** (CoM) of the group. Ideally, a swarm flocks cohesively [14]. In order words, agents should stay as close as they can to the CoM, which can be calculated as in (3.2), by summing up the position of each agent and averaging the positions of the whole network by the number of agents N.

$$CoM(t) = \left(\frac{x_1(t) + ... + x_N(t)}{N}, \frac{y_1(t) + ... + y_N(t)}{N}, \frac{z_1(t) + ... + z_N(t)}{N} \right) \tag{3.2}$$

Moreover, the distance between each drone of the swarm and the $CoM(t)$ has to be calculated. Where the distance is the module of the vector or the Euclidean norm $||\,\cdot\,||$, between

the current position of each drone $s_i(t)$ and the $CoM(t)$. As a result, a cohesion vector is generated, illustrated in (3.3).

$$Coh = [||[x_1(t), y_1(t), z_1(t)], CoM(t)||, ..., ||[x_N(t), y_N(t), z_N(t)], CoM(t)||] \tag{3.3}$$

By averaging the cohesion vector by the number of agents, the ***Average Cohesion*** (AvC) can be computed. This is a main representation of the LoC on each time step. Equation (3.4) introduces the mathematical expression of the AvC.

$$AvC(t) = \frac{1}{N} \sum_{i=1}^{N} ||[x_i(t), y_i(t), z_i(t)], CoM(t)|| \tag{3.4}$$

Finally, AvC has to be averaged over the whole analyzed time period, giving a measurement of the global AvC among all the time steps, defined as LoC, which takes into account those time steps where some of the drones are acting under failure.

$$LoC = \frac{1}{t} \sum_{t=0}^{t} AvC(t) \tag{3.5}$$

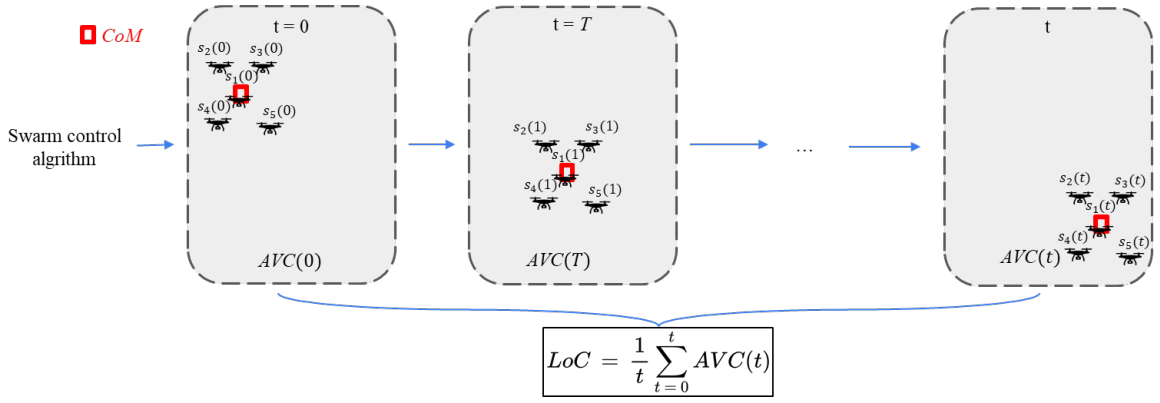


Figure 3.1: Swarm state evolution and LoC calculation.

Chapter 4

Experimental part

4.1 Experimental Setup

In our work, we demonstrate that the GNN approach can improve robustness. To do so, it is important to define first a comparative model to which GNN outperforms. This former model is the baseline of our approach, the Boids model previously introduced. Currently, [16] gives some insight into robustness and how flocking tasks can be made. Additionally, the authors of [16] have open-sourced the implementation[1] for public access, which we have made use of as a base for our algorithm implementation.

As for the evaluating platform, we selected ***AirSim*** because evaluation with actual drone flyers might be too costly to accomplish similar results. ***AirSim*** is a real-time open-source simulator that enables us to test the controllers with higher-order dynamics and lower delay time between control actions. As introduced in [16], controllers can be modeled in different scenarios and simulators, but ***AirSim*** gives an excellent approximation of the real flying tasks.

[1] `https://github.com/katetolstaya/multiagent_gnn_policies`

4.1.1 Ground truth controller

This work defines a Ground Truth model to be cloned by GNNs. A graphical representation of the algorithm used for the Ground Truth model is described in Figure 4.1. Inspired from [14], we present a centralized version of the Boids model in which global information of the graph is assumed; indeed, the range of vision is amplified to the whole graph. The dynamics of the agents on this model can be characterized by the already introduced separation, alignment, and cohesion rules, illustrated in Figure 4.2, which have also been used for the Boids model implementation to achieve the flocking behavior on a decentralized architecture.

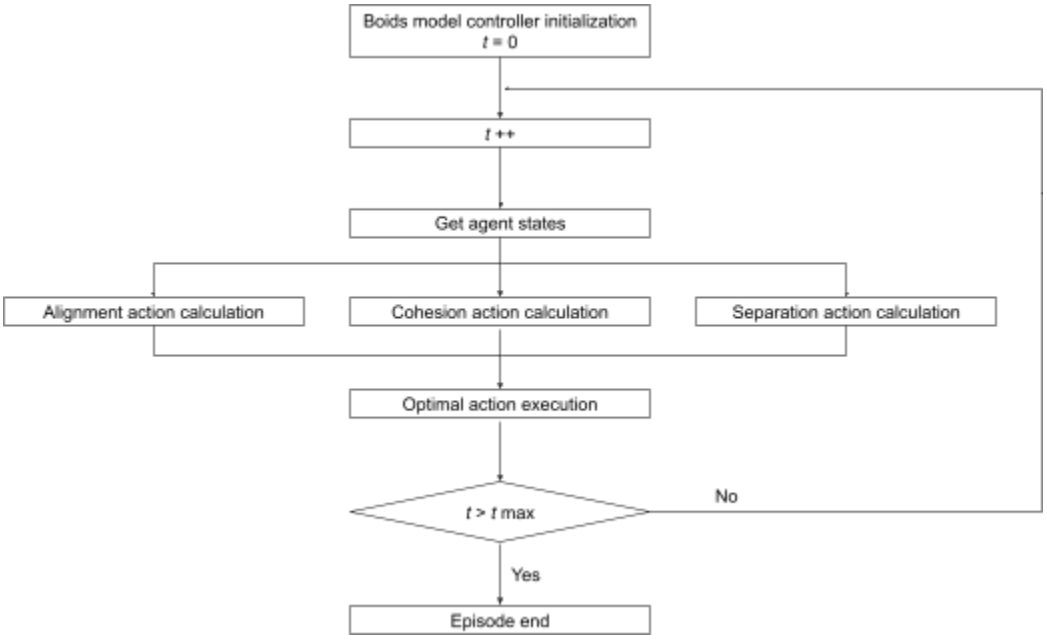


Figure 4.1: Ground Truth model implementation algorithm.

Consider the previously defined position, velocity, and acceleration vectors for each agent in the network at each simulation time step. To implement these three rules into our algorithm, we use the Ground Truth math model defined below, where the control actions are described using differential equations; indeed, velocity and position vectors for each of the agents in

the swarm can be defined and updated as in (4.1).

$$\ddot{s_i}(t) = \dot{v_i}(t) = u_i(t) \tag{4.1}$$

Consider the separation rule, which accounts for the minimum distance between Boids or agents so that every agent can avoid collisions and leave some space for them to move and react to changing actions freely. Repulsion force can be calculated as indicated in (4.2), summing up the weighted inverted distance (defined by the Euclidean norm $||\,.\,||$) between the agent and its neighbors, being stronger as agents come closer.

$$\dot{v}^*_{separation,i}(t) = w_{sep}\left(\sum_{j \in \mathcal{N}_i} \frac{s_i(t) - s_j(t)}{||s_i(t) - s_j(t)||}\right) \tag{4.2}$$

The alignment rule aims to ensure that each swarm agent aligns its velocity with its neighbors' velocities moving in the same direction as a group. Indeed, it represents the averaged velocity of its neighbors as introduced in (4.3).

$$\dot{v}^*_{alignment,i}(t) = w_{align}\left(\frac{\sum_j v_j(t)}{N}\right) \tag{4.3}$$

The cohesion rule is responsible for moving each agent towards the Center of Mass ($CoM_{\mathcal{N}_i}(t)$) of its neighbors at each time step t. Indeed, it ensures that all the agents flock cohesively within their neighborhood. Equation (4.4) describes the attraction force between agents in a neighborhood.

$$\dot{v}^*_{cohesion,i}(t) = w_{coh}(CoM(t) - s_i(t)) \tag{4.4}$$

The three rules are adjustable by their corresponding weights w_{sep}, w_{align}, and w_{coh}, describing the interaction between agents. These weights are used to determine and control the influence of each of the three rule forces, making it possible to achieve the desired flocking behavior. In our work, we adjust the weights so that the declarative model matches the centralized model behavior without agent failures. Indeed, flocking more clustered enables

maintaining an active connection between agents while flocking without dealing with swarm segmentation. Equation 4.5 represents the sum of the three rules. Even more, it represents the control action to be taken by each agent on the Ground Truth model, in essence, the optimal action to be cloned or imitated by GNNs.

$$u_i^*(t) = \dot{v}_{separation,i}^*(t) + \dot{v}_{alignment,i}^*(t) + \dot{v}_{cohesion,i}^*(t) \tag{4.5}$$

4.1.2 Baseline controller

The Boids model has been defined as the baseline approach of our work as it is a commonly used algorithm for simulating the collective and collaborative motion of agents. Moreover, this model has low computational requirements and high flexibility to be adjusted to different environments, making it even more attractive for use.

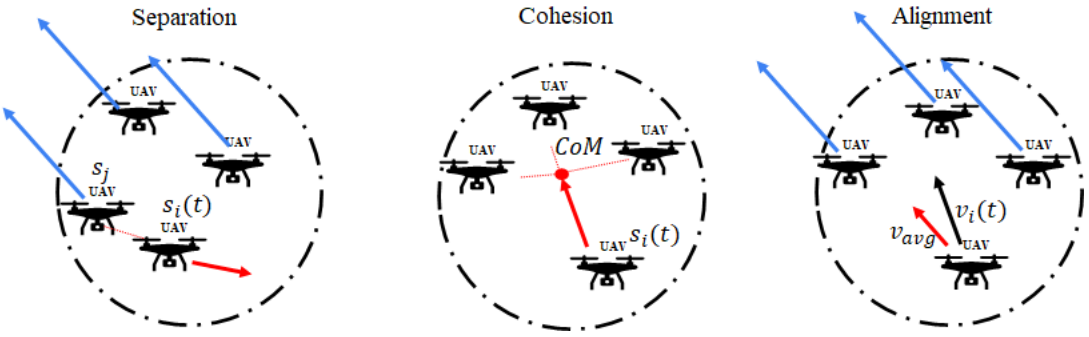


Figure 4.2: Illustration of the three rules that govern the flocking behavior.

Consider the three rules of the Boids model, represented in Figure 4.2, whose mathematical representation was introduced by [11]. The mathematical expression for the separation rule is described by (4.6). As in the Ground Truth model, this rule accounts for the minimum distance between Boids or agents so that agents can avoid collisions and leave some space to

move and react to changing actions freely.

$$\dot{v}_{separation,i}(t) = w_{sep}\left(\sum_{j\in\mathcal{N}_i}\frac{s_i(t)-s_j(t)}{||s_i(t)-s_j(t)||}\right) \tag{4.6}$$

In contrast to the Ground Truth model, the alignment rule only considers the velocity of its closest neighbors, so the alignment is only produced within the radius R. Equation (4.7) introduces the alignment rule of the decentralized architecture.

$$\dot{v}_{alignment,i}(t) = w_{align}\left(\frac{\sum_{j\in\mathcal{N}_i} v_j(t)}{N_{\mathcal{N}_i}}\right) \tag{4.7}$$

Equation (4.8) describes the attraction force between agents in the neighborhood. The cohesion rule is responsible for moving each agent towards the Center of Mass ($CoM_{\mathcal{N}_i}(t)$) of its neighbors at each time step t, as already defined. However, in the Boids model, the neighborhood of each agent $\mathcal{N}_i$ is reduced so that only the agents within the radius of communication between agents are considered.

$$\dot{v}_{cohesion,i}(t) = w_{coh}(CoM_{\mathcal{N}_i}(t) - s_i(t)) \tag{4.8}$$

As described for the Ground Truth model, the output or control action can be obtained from the sum of the three rules of the flocking behavior, as in (4.9). However, for the Boids model, the action is not the optimal one, as explained further on, enabling us to make comparisons between the optimal actions from the Ground Truth model against the actions to be taken by the agents following the Boids model policy, Tables 4.1 and 4.2.

$$u_i(t) = \dot{v}_{separation,i}(t) + \dot{v}_{alignment,i}(t) + \dot{v}_{cohesion,i}(t) \tag{4.9}$$

Algorithm 1: Boids model implementation

Input: Agent relative positions and velocities.

Output: Agent accelerations.

System initialization $\rightarrow steps = 0$;

while $steps < 500$ **do**

  Get states (AirSim)$\rightarrow s_i(t), v_i(t)$;

  $\dot{v}_{separation,i}(t) = w_{sep}\left(\sum_{j\in\mathcal{N}_i} \frac{s_i(t)-s_j(t)}{||s_i(t)-s_j(t)||}\right)$;

  $\dot{v}_{alignment,i}(t) = w_{align}\left(\frac{\sum_{j\in\mathcal{N}_i} v_j}{N}\right)$;

  $\dot{v}_{cohesion,i}(t) = w_{coh}(CoM_{\mathcal{N}_i}(t) - s_i(t))$;

  $u_i(t) = \dot{v}_{separation,i}(t) + \dot{v}_{alignment,i}(t) + \dot{v}_{cohesion,i}(t)$;

  $u_i(t) \rightarrow$ Send accelerations (*AirSim*);

end

4.1.3 GNN controller

Our proposed controller is based on GNNs configured to perform an imitation learning task. GNN controller leverages the network's structure and positioning to predict each agent's actions. In a UAS swarm network, every node has some features of dimensions F, which can be gathered as a feature matrix of dimensions (NxF), denoted in (4.10). We have used the features defined in [16] as our base for the GNN approach, even though we have added an extra feature considering the velocity of each agent itself, that when introduced to the NN is normalized as well as the rest of features to obtain better convergence and avoid the overweight of one feature over the others.

$$X(t) = \begin{bmatrix} x_1(t) \\ . \\ . \\ . \\ x_N(t) \end{bmatrix} \tag{4.10}$$

For the primary purpose of this work, we introduce the concept of ***Graph Shift Operator*** (GSO), which is a critical component of GNNs used to propagate information within the network, and, is a learnable parameter. The basic functioning of GSO is that it is a nonzero value if and only if $(i, j) \in \mathcal{E}(t)$, $\mathcal{E}(t)$ meaning the connection between agents or edges of the graph. The propagation equation is described in (4.11).

$$[S(t)X(t-1)]_i = \sum_{j \in \mathcal{N}_i(t)} s_{ij}(t) x_j(t-1) \tag{4.11}$$

A valid GSO for this application can be a weighted and unweighted adjacency matrix or, even more, weighted, unweighted, or normalized Laplacians. The Graph Shift Operator may help in recursion to extract a sequence of signals from it, as defined in the following equation.

$$Y_k(t) = S(t)Y_{(k-1)}(t-1) \tag{4.12}$$

Meaning that each agent at k-hop distance will receive the information from their neighbors at a (k-1)-hop distance following the previous equation, indeed, following the GSO at that fixed time. Generally, an agent located at a k-hop distance of an agent considered the initial one $Y_o(t) = X(t)$ will receive the information in the following fashion.

$$Y_k(t) = [S(t)S(t-1)...S(t-k+1)]X(t-k) \tag{4.13}$$

Thus, the sequence of signals is aggregated into a single output matrix for each node i present in the graph. Which can be expressed with the following mathematical representation (4.14).

$$Z(t) = [X(t), S(t)X(t-1), ..., S(t-(k-2))X(t-(k-1))] \tag{4.14}$$

The matrix $Z(t)$ results from aggregating all the graph states. Its dimensions are N x k x F, where N is the number of nodes in the graph, k is the number of hops made to gather all the information, and F is the number of features of each node in the graph. As can be seen, there is a delay in the matrix, known as ***delayed aggregation***.

Finally, once all the information is aggregated in matrix $Z(t)$. The last step is to apply a ***Neural Network*** (NN) in which the input would be the aggregation matrix. The layers of the NN are represented by (4.15). The output is given as the predicted action to be taken by each agent in the network, defined by the matrix $U(t)$.

$$Z_l = \sigma_l(\theta_l Z_{l-1}) \tag{4.15}$$

Each layer's l output is denoted by Z_l, σ_l accounts for the activation function used for each layer of the NN, and θ_l represent the learnable parameters of the NN. The overall expression of the NN is given in (4.16).

$$U(t) = NN_\Theta(Z(t)) \tag{4.16}$$

Training the GNN is a way of modeling the network to obtain the desired predictions. The training aims to find the optimal previously defined controller with the highest possible accuracy. To do so, it is necessary to introduce the concept of ***loss function*** (L), which it is compared the optimal output with the predicted output resulting from the computation of the GNN given a set of inputs. For this training task, we need training samples to be learned by the controller. In this training phase, all the learnable parameters of the GNN and NN

are adjusted to have the lowest prediction errors and the lowest value of L. For training purposes, the training policy used to find the optimum parameters Θ^* can be expressed in (4.17).

$$\Theta^* = \arg\min_{\Theta} \left(\sum_{X(t),U^*(t)\in\mathcal{D}} L(U(t), U^*(t)) \right) \tag{4.17}$$

Algorithm 2: GNN model implementation

Input: Agent relative positions and velocities.

Output: Agent accelerations.

System initialization $\rightarrow$ *steps* $= 0$;

while *steps* < 500 **do**

  Get features (AirSim)$\rightarrow r_{ij}(t), v_i(t)$;

  Start the message passing;

  Aggregation$\rightarrow Z(t) = [X(t), S(t)X(t-1), ..., S(t-(k-2))X(t-(k-1))]$;

  Action prediction$\rightarrow U(t) = NN_{\Theta}(Z(t))$;

  $u_i(t) \rightarrow$ Send accelerations (*AirSim*);

end

4.2 Experimental design

Our default scenario considers a flock of $N = 9$ agents with a communication radius of $R = 4m$. We have evaluated a sampling and step time of $T = 0.01s$, based on the simulation updating rate of the ***AirSim*** software. Agent initial positions are uniformly located in a grid formation, with a distance $d_{ij} = R - 0.25m$ between agents to ensure complete communication. The velocity of the agents is initialized with a module of $||v_i(0)|| = 5m/s$, where directions are randomly selected, and acceleration is bounded for all the agents throughout the entire simulation to $||u_i(t)|| \leq 3m/s^2$. Even more, it should be considered that random initialization is replicated over experiments using seeds so that the results are as comparable as possible. To handle heavy computational loads and ***AirSim*** simulator, we have used an

NVIDIA GeForce GTX 1080 GPU.

For the baseline approach, we set the number of testing episodes to 50 and the number of steps to 500. For each weight, we have made experimental adjustments, leading to the most optimal result when $w_{sep} = 0.875$, $w_{align} = 1.1$, and $w_{coh} = 1.25$. Whereas for the Ground Truth model, we have achieved the best result with $w_{sep} = 15.75$, $w_{align} = 1.1$, and $w_{coh} = 0.79$, where the separation rule has a greater weight to balance the focus on the nearest neighbors while the other rules account for the whole swarm.

On the other hand, for the GNN model, we set the number of k-hops over which the agent information is passed to $k = 4$. The NN was built on convolutional layers, the input, the first hidden layer, and the second hidden layer activated by the hyperbolic tangent. For the output layer, a linear activation has been applied. Each hidden layer has been provided with 128 neurons so that the model can understand every non-linear behavior of the flocking model. The GNN model has been trained with a learning rate of $5 \cdot 10^{-4}$, over 300 episodes with 500-time steps each; indeed, the total time spent on an episode was $t = 500T$.

To test the validity of our work, drone failures are randomly generated following several patterns, as depicted in Tables 4.1 and 4.2. We have simulated a sequential failure pattern, where only one agent failed at a time. Failures have been generated periodically with a controllable frequency depending on the desired total number of failures. The second technique for simulating failures is a one-time failure of a controllable number of drones so that a specified number of drones fail simultaneously at a certain time. We have used ***AirSim*** software for gathering training data in an online learning setting described in [7]; more precisely, we have used the ***Dataset Aggregation*** (DAgger) algorithm defined by [15], with a probability $\beta = 1$ of choosing the Ground Truth model. Hence, the Ground Truth action is always preferred to be aggregated to the dataset $\mathcal{D}$.

4.3 Results

This section reports the most representative results obtained from the simulations. Comparisons have been made between the ground truth, baseline, and GNN approach with the metrics (e.g., LoC). The expected results of these comparisons are the GNN model having a higher level of accuracy when dealing with failure, indeed, being able to maintain the group's unity, as shown in Tables 4.1 and 4.2. First, we have compared the optimal actions of the Ground Truth model against the Boids model in terms of LoC evaluating their performance on different scenarios. Moreover, this comparison enables us to establish a range over which the GNN model should be performing to prove the enhancement of the robustness of the swarm.

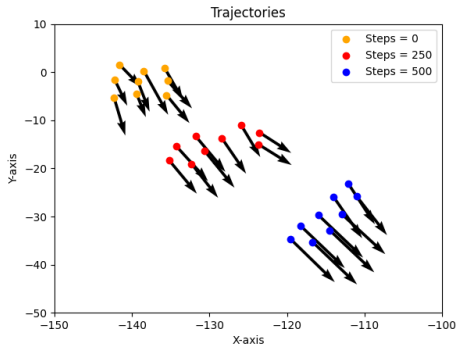
				Performance (%)	
Agent failures	Ground truth	Boids	GNN	Ground truth vs Boids	Ground truth vs GNN
4	2.952	3.377	3.08	87.424	95.840
3	3.025	3.381	3.12	89.470	97.031
2	3.104	3.436	3.16	90.337	98.273
1	3.163	3.466	3.22	91.236	98.276
0	3.237	3.552	3.28	91.140	98.581

Table 4.1: Performance evaluation under sequential failure scenario as a function of LoC.

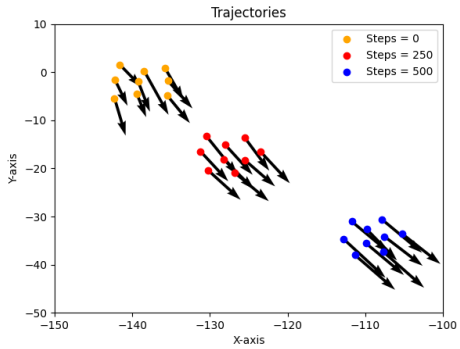
				Performance (%)	
Agent failures	Ground truth	Boids	GNN	Ground truth vs Boids	Ground truth vs GNN
4	2.947	3.386	3.240	87.033	90.952
3	3.024	3.468	3.142	87.206	96.240
2	3.103	3.450	3.210	89.955	96.658
1	3.163	3.466	3.235	91.236	97.772
0	3.237	3.552	3.284	91.140	98.581

Table 4.2: Performance evaluation under one-time failure scenario as a function of LoC.

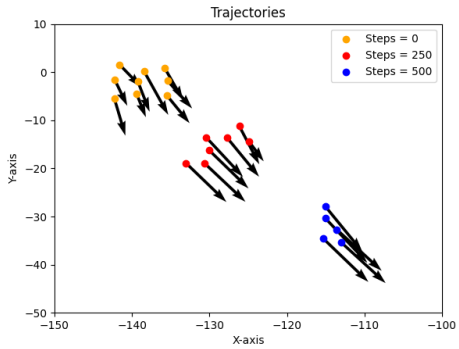
As shown in Tables 4.1 and 4.2, performance in the Boids model, as well as in the GNN model, starts to decay upon agent failures. However, the GNN model maintains a better performance even with a large number of failures. As represented in the tables, LoC starts to decrease with the number of agents, as they spread over a smaller area. However, as we are not comparing the performance of the swarm on a single model but comparing several models between each other, this fact can be neglected.



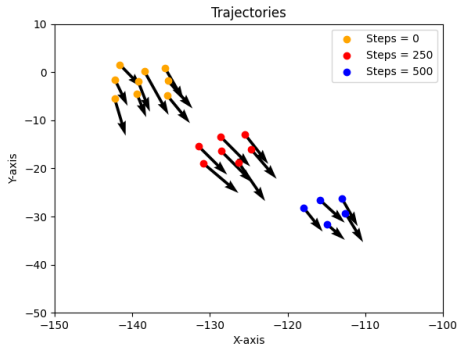
(a) Boids model simulation with no agent failures.



(b) GNN model simulation with no agent failures.



(c) Boids model simulation with 4 agent failures.

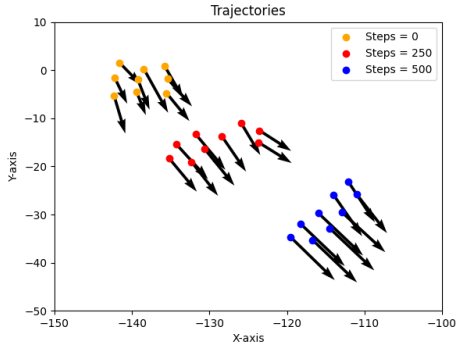


(d) GNN model simulation with 4 agent failures.

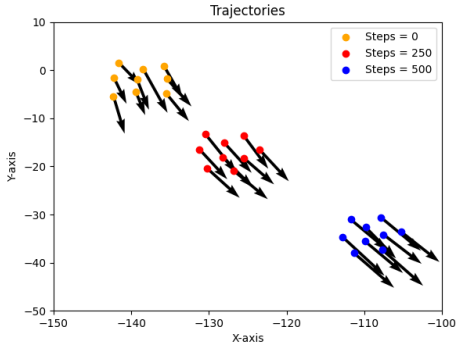
Figure 4.3: Airsim test simulation with sequential agent failures.

These two tables represent that the Boids model has a similar performance under different failing patterns due to the fact that the Boids model is more sensitive to disconnections between agents, leading to the segmentation of the initial group. In both the Boids and GNN

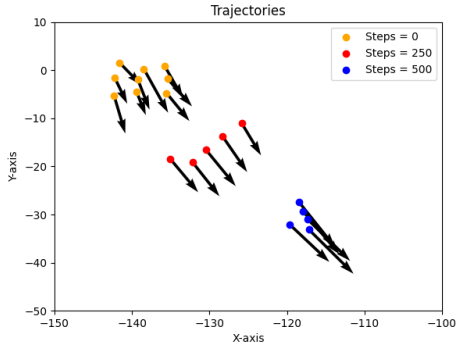
models, disconnections are irreversible, meaning that once the segmentation of the swarm is reached, the group cannot get together again.



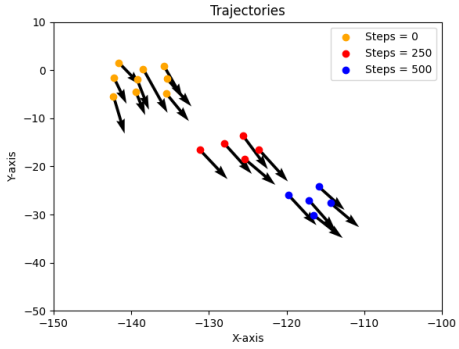
(a) Boids model simulation with no agent failures.



(b) GNN model simulation with no agent failures.



(c) Boids model simulation with four agent failures.



(d) GNN model simulation with 4 agent failures.

Figure 4.4: Airsim test simulation with one-time agent failures.

The potential benefit of the GNN model could be reduced to the fact that information is passed across the group so that every agent has information about k-hop neighbors, indeed, local and global information about the group. Therefore, every agent can make decisions based on the aggregated states of the k-hop neighbors. In fact, it eventually leads to better performance and cohesive flocking due to the increased information collected by every drone. Figures 4.3 and 4.4 give a good insight into how the GNN model is capable of maintaining the flock in a more cohesive manner making disconnections less harmful to the swarm and,

therefore more robust to failures. Nevertheless, Table 4.2 shows that the more critical the failures are, the less likely the GNN model is to keep performing well, as more critical disconnections occur in the group that makes it impossible to maintain the cohesive behavior and, therefore, more cluster separations. Hence, the benefits of using the GNN model become smaller when critical failures arise even though it still outperforms the Boids model.

Chapter 5

Conclusion

The current research aims to improve the robustness of drone swarm control systems using Graph Neural Networks (GNNs). This project has been carried out with the help of ***AirSim*** simulator, a software developed by Microsoft. It has been combined with Python 3.10 programming language, using the available Python API for ***AirSim***, to build the required scripts for running the simulations. To test the validity of our research, we have built three models: the Ground Truth, Boids, and GNN. We have introduced two well-known communication structures used for SI applications, Infrastructure-based (Centralized) and FANET (Decentralized). Even though our work has focused on the decentralized architecture because of its autonomous capability. In the decentralized structure, we have introduced the idea of a leader-follower structure. The follower side is our area of interest, on which we have built the Boids and GNN model. The Boids model is a rule-based approach that simulates animal and robot swarm collective behavior. The GNN model is the solution for handling complex situations more robustly compared to the Boids model, as illustrated in Table 4.1 and 4.2. The main idea behind GNNs is leveraging the information passing throughout the group by multi-hop exchanges to make predictions of each agent's actions as accurately as possible.

Besides, this work aims to answer the question of why we should attempt to compare the Boids model directly against the GNN model instead of comparing it to another ML model. The advantage of Graph Neural Networks is that they are specifically designed to process graph-structured data so that they have a better understanding of complex scenarios and relationships between agents and can handle them very efficiently. Therefore, they seem to be an excellent approach to address the problem of flocking or, even more, flying in a graph-structured manner.

This research has shown that Graph Neural Networks are an excellent solution to improve the robustness of a swarm while flying autonomously from the follower side. They leverage their capability to process graph-structured data to outperform previously used control algorithms such as the Boids model. Therefore, GNNs may be a desirable algorithm for dealing with collective behavior tasks in autonomous systems with communication constraints.

5.1 Acquired competences

The development of this thesis has contributed to acquiring the following competencies:

- Graph Neural Networks and its implementation in programming languages such as Python, with the help of libraries such as PyTorch Geometric.
- Model-based programming techniques in Python software environment.
- Swarm intelligence models and communication structures.
- Improved knowledge of drone flying physics and control strategies.

5.2 Future work

While this work has provided valuable insights into drone swarm control, there are several avenues for future research and exploration. Some of the ideas for future works are:

- Transferring the knowledge from this work to real life, testing and evaluating the models in real-life conditions.
- Search for new ways GNN can help improve the performance of the overall systems working collectively, such as improving energy consumption.
- Study ways of achieving a more robust system on the leader side, as it has been done on the follower side so that the overall robustness of the system can be improved.

Bibliography

[1] O. Blomqvist, S. Bremberg, and R. Zauer. Mathematical modeling of flocking behavior., 2012.

[2] M. Campion, P. Ranganathan, and S. Faruque. Uav swarm communication and control architectures: a review. *Journal of Unmanned Vehicle Systems*, 7(2):93–106, 2018.

[3] X. Chen, J. Tang, and S. Lao. Review of unmanned aerial vehicle swarm communication architectures and routing protocols. *Applied Sciences*, 10(10):3661, 2020.

[4] M. Duarte, J. Gomes, V. Costa, T. Rodrigues, F. Silva, V. Lobo, M. M. Marques, S. M. Oliveira, and A. L. Christensen. Application of swarm robotics systems to marine environmental monitoring. In *OCEANS 2016-Shanghai*, pages 1–8. IEEE, 2016.

[5] T.-K. Hu, F. Gama, T. Chen, Z. Wang, A. Ribeiro, and B. M. Sadler. Vgai: End-to-end learning of vision-based decentralized controllers for robot swarms. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4900–4904. IEEE, 2021.

[6] P. Khopkar. *Control and Coordination of Multi-Drone Systems Using Graph Neural Networks*. PhD thesis, Arizona State University, 2021.

[7] 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)*, pages 11785–11792. IEEE, 2020.

[8] A. V. Malawade, S.-Y. Yu, B. Hsu, H. Kaeley, A. Karra, and M. A. Al Faruque. Roadscene2vec: A tool for extracting and embedding road scene-graphs. *Knowledge-Based Systems*, 242:108245, 2022.

[9] A. V. Malawade, S.-Y. Yu, B. Hsu, D. Muthirayan, P. P. Khargonekar, and M. A. Al Faruque. Spatiotemporal scene-graph embedding for autonomous vehicle collision prediction. *IEEE Internet of Things Journal*, 9(12):9379–9388, 2022.

[10] A. V. Malawade, S.-Y. Yu, J. Wang, and M. A. A. Faruque. Rs2g: Data-driven scene-graph extraction and embedding for robust autonomous perception and scenario understanding. *arXiv preprint arXiv:2304.08600*, 2023.

[11] U. Mehmood, N. Paoletti, D. Phan, R. Grosu, S. Lin, S. D. Stoller, A. Tiwari, J. Yang, and S. A. Smolka. Declarative vs rule-based control for flocking dynamics. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 816–823, 2018.

[12] T. Mortlock, D. Muthirayan, S.-Y. Yu, P. P. Khargonekar, and M. A. Al Faruque. Graph learning for cognitive digital twins in manufacturing systems. *IEEE Transactions on Emerging Topics in Computing*, 10(1):34–45, 2021.

[13] R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on automatic control*, 51(3):401–420, 2006.

[14] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, 1987.

[15] S. Ross, G. J. Gordon, and J. A. Bagnell. No-regret reductions for imitation learning and structured prediction. corr abs/1011.0686 (2010). *arXiv preprint arXiv:1011.0686*, 2010.

[16] 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*, pages 671–682. PMLR, 2020.

[17] J. Wan, B. S. Pollard, S. R. Chhetri, P. Goyal, M. A. A. Faruque, and A. Canedo. Future automation engineering using structural graph convolutional neural networks. *arXiv preprint arXiv:1808.08213*, 2018.

[18] R. Yasaei, S. Faezi, and M. A. Al Faruque. Golden reference-free hardware trojan localization using graph convolutional network. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(10):1401–1411, 2022.

[19] R. Yasaei, S.-Y. Yu, and M. A. Al Faruque. Gnn4tj: Graph neural networks for hardware trojan detection at register transfer level. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1504–1509. IEEE, 2021.

[20] R. Yasaei, S.-Y. Yu, E. K. Naeini, and M. A. Al Faruque. Gnn4ip: Graph neural network for hardware intellectual property piracy detection. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 217–222. IEEE, 2021.

[21] S.-Y. Yu, Y. G. Achamyeleh, C. Wang, A. Kocheturov, P. Eisen, and M. A. A. Faruque. Cfg2vec: Hierarchical graph neural network for cross-architectural software reverse engineering. *arXiv preprint arXiv:2301.02723*, 2023.

[22] S.-Y. Yu, A. V. Malawade, D. Muthirayan, P. P. Khargonekar, and M. A. Al Faruque. Scene-graph augmented data-driven risk assessment of autonomous vehicle decisions. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):7941–7951, 2021.

[23] S.-Y. Yu, R. Yasaei, Q. Zhou, T. Nguyen, and M. A. Al Faruque. Hw2vec: A graph learning tool for automating hardware security. In *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 13–23. IEEE, 2021.