Autonomous Hierarchical Control and Coordination of Unmanned Aerial Vehicle Swarms

Yunpeng Yang

A dissertation submitted to the University of Bristol in accordance with the requirements for award of the degree of Master of Science in Computer Science (Conversion) in the Faculty of Engineering.

School of Computer Science

August 2024

Word count: ∼15000

Abstract

In recent years, the application of unmanned aerial vehicle swarms in civilian and military scenarios has become increasingly promising. There has been a surge of research interest in swarm algorithms. This thesis proposes a hierarchical swarm algorithm, where the unmanned aerial vehicles dynamically organise themselves into a tree structure, and coordinate with each other to carry out assigned tasks. The algorithm enables large swarms to deal with complex tasks robustly. The main contributions of this thesis are as follows.

- An application scenario of autonomous swarms is conceived (see chapter 3). The scenario is inspired by drone shows. The task of a swarm is to form designated shapes.
- Based on the scenario, a new algorithm is designed for the hierarchical control and coordination of swarms (see chapter 4).
- The algorithm is implemented in the Rust programming language with more than 2000 lines of source code (see chapter 5).
- Simulations are carried out to demonstrate the design and implementation of the algorithm (see chapter 6). The results show that the algorithm is suitable for large swarm size and complex task coordination.

Keywords: UAV swarm, hierarchical swarm, swarm control, swarm coordination.

GitHub repository: https://github.com/NGC0001/AutoSwarm

Acknowledgements

This thesis is accomplished under the guidance of my supervisor, Professor Ruzanna Chitchyan. She was also one of the lecturers of the course Software Engineering. I would like to thank her for her support and advice. I would also like to thank all the people who have helped me through out the year of study in the University of Bristol.

Author's Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

Yunpeng Yang

31 August 2024

Contents

List of Tables

List of Figures

1. Aims and Objectives

Unmanned aerial vehicles (UAVs) have seen more and more application in personal life, commercial business, industries, and military affairs. Recently with the advance of technology, scenarios emerge where a group of UAVs carry out an assigned task in a coordinated way [14], [21], [31]. Such UAV groups are called UAV swarms. Swarms are much more complex than individual UAVs, and application of swarms is still in its infancy.

To fully explore and utilise UAV swarms, a lot of researchers have investigated algorithms related to the communication, control, and task management of swarms [20], [53]. Such algorithms can be roughly divided into two categories, *centralised algorithms* where a leader UAV is in charge of all the other UAVs, and *decentralised algorithms* where all UAVs are treated equally. Decentralised algorithms can further be divided into two types, *negotiation-based algorithms* where each UAV negotiates with other UAVs to make decisions, and *reaction-based algorithms* where UAVs achieve collective behaviour through observing other UAVs and reacting according to preset rules.

Centralised algorithms are simple, but are prone to single points of failure induced by the leader. Reaction-based algorithms are robust, but are not suitable for complex tasks where elaborate coordination among UAVs is necessary. Negotiation-based algorithms are powerful, but entails a large amount of computation and all-to-all communication during negotiation, effectively limiting the size of the swarm.

The **aim** of this thesis is to *find a robust algorithm suitable for large swarms and*

complex tasks. The proposed **solution** is a *hierarchical swarm* where UAVs dynamically organise themselves into a tree structure. Instead of all-to-all communication, messages are mainly sent between UAVs whose corresponding tree nodes are directly connected. Tasks are managed in a divide-and-conquer way layer by layer down the tree.

To achieve this aim, the thesis below presents the following contributions:

1. Firstly a *swarm application scenario* is conceived (see chapter 3).
2. Then based on the scenario, the *algorithm is proposed* (see section 4.1), and the *software architecture* that embodies this algorithm is *designed* (see section 4.3).
3. Thirdly, a detailed *implementation is built* to demonstrate the viability of the proposed solution (see chapter 5).
4. Lastly, *simulation is carried out to evaluate the algorithm* (see chapter 6). The results verified that the algorithm is suitable for large swarms and complex tasks.

2. Background and Context

UAVs have been widely used for personal, commercial, industrial, academic, and military purposes. In many situations, they provide flexible and cost-effective solutions. For example, first-person view (FPV) drones are widely used by video bloggers, and surveillance drones greatly promote intelligence capabilities of an army.

However, the use of a single UAV is limited by its size, sensors, computing power, and other hardware restrictions. Besides, once a single UAV fails, the task being executed by it will also fail. In recent years, with the advancement of technology, it has become possible to group multiple UAVs into a swarm to perform complex tasks. Compared to single UAVs, swarms feature the advantages of high efficiency and robustness, and significantly expand the ways in which UAVs are used. Many researchers have investigated the potential use cases of swarms [14], [20], [31], [48], [53]. In civilian fields, swarms can contribute to large-scale environmental monitoring, efficient post-disaster search and rescue, forest fire fighting, logistics, etc. In military fields, swarms may be used to enhance reconnaissance, or even aerial warfare.

It should be noted that, a "swarm" in this thesis refers to a group of UAVs which autonomously coordinate with each other in real-time to complete tasks. If the UAVs in a group just fly and operate according to preprogrammed data, of if the UAVs are fully controlled by a remote control centre, e.g., a GCS, then the UAV group is not considered a swarm.

Generally, for a single UAV, the onboard hardware and software may include the

following functionalities.

- Communication: A UAV needs to exchange data with other UAVs or with a ground control station (GCS).
- Environment perception: A UAV needs to detect obstacles and to track targets. It also needs positioning information, which may come from a GPS receiver.
- Path planning: The flight path shall meet the requirement of a certain task. Collision with obstacles or other UAVs must be avoided.
- Flight control: With planned flight path, the actuators and engines shall be controlled according to aerodynamic principles, so as to acquire desired attitude and velocity.
- Task planning: When assigned a task, e.g., delivering a parcel, a UAV needs to figure out how to accomplish it. This is especially important for an autonomous UAV.

For a UAV swarm, except for all the above technical difficulties for single UAVs, there are also additional challenges including but not limited to [6], [21], [53]:

- Network topology [8], [9], [12], [20], [21]: In some cases, the communication among UAVs is limited by a lot of factors, such as communication range, bandwidth, obstacles and terrain. An all-to-all communication network may not be feasible. Therefore, a good network topology design is needed to ensure efficient data transfer.
- Formation control [21], [22], [28], [31], [41], [49]: UAVs need to stay together to form a swarm. In some cases, the UAVs also need to maintain their relative positions inside a swarm.

- Path planning [14], [21], [35]: For swarms, the fight path needs to be planned with a view of the entire swarm. Besides, flight path of each individual UAV is closely coupled with the formation control and network topology of the whole swarm.
- Task planning and allocation [10], [21], [25], [28], [33]: The task of the whole swarm must be planned and divided into sub-tasks for each individual UAV. For most UAV tasks, task management may overlap with path planning and formation maintaining to some extent. For instance, in aerial search tasks, flight path must ensure coverage of target areas. While in drone shows, the task itself is to form certain formations.

To fully explore and utilise UAV swarms, a lot of algorithms have been developed regarding different aspects of swarms. This thesis will particularly focus on formation control and task allocation.

2.1. Formation Control

For a swarm to function normally, UAVs need to stay close to each other, in order to communicate and cooperate. For certain tasks, UAVs are even required to keep rigid relative positions. Since swarm formation is ultimately achieved by the movement of every individual UAV, formation control is therefore closely coupled with path planning. In this section, some popular algorithms related to formation control and path planning are reviewed.

2.1.1. Leader-follower Method

In the leader-follower method [22], [31], [49], one UAV is chosen as the leader, while all other UAVs are followers. The leader is responsible for calculating the flight path. The followers just track and follow the leader. This method is simple to implement. Especially

for the followers, the formation problem is converted into a trajectory tracking problem. However, swarms employing this method risk single points of failure. As the leader UAV acts as a central control node, once it breaks down, the whole swarm may malfunction. Moreover, the calculation of the swarm flight path is done by the leader alone, which may pose a huge burden for its computing resource.

2.1.2. Virtual Structure Method

The virtual structure method treats the desired formation as a rigid virtual structure [22], [31], [49]. Each UAV is expected to embody a respective virtual point on the virtual rigid body. It observes the positions and velocities of other UAVs, fits the position, orientation, and velocity of the virtual structure, and then calculates its appropriate position and velocity. There is no central UAV, but the virtual geometric centre can be viewed as the leader of the swarm, and all the UAVs follow this virtual leader, as depicted by figure 2.1. Therefore this method is also known as virtual leader method.

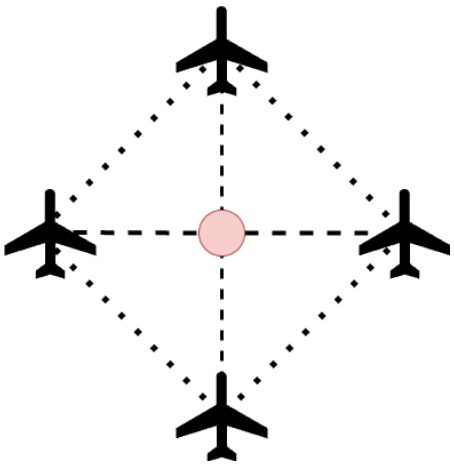

Figure 2.1.: In virtual structure method, the UAVs can be viewed as following the virtual geometric centre.

Due to the elimination of a central node, the virtual structure method is robust. But the UAVs are required to maintain a rigid formation, this compromises flexibility in some cases, e.g., when avoiding obstacles.

2.1.3. Behaviour-based Method

Inside a swarm, each UAV may have several prescribed basic behaviours. For eaxample, aggregation, separation, collision avoidance, and target tracking. Each behaviour calculates a desired velocity according to some rule. A coordinator is responsible for synthesising the final velocity out of these behavioural velocities, based on information from the environment and other UAVs.

One of the velocity synthesis methods is the null-space method [1], [2], [4], [17]. Each basic behaviour is assigned a priority. Velocity vector of lower-priority is projected onto the null space of the immediately higher-priority velocity, as shown in figure 2.2. In this way, lower-priority behaviour will not conflict with higher-priority behaviour.

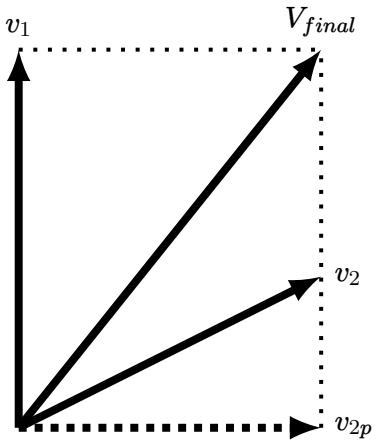

Figure 2.2.: In null-space method, the velocity v_2 from the lower-priority behaviour is projected onto the null space of the higher-priority v_1, resulting in v_{2p}, and the final velocity is $V_{final} = v_1 + v_{2p}$.

Behaviour-based method can be used not only in UAV swarm formation control and path planning, in fact, it can be viewed as a general framework for coordinating interacting behaviours [22]. The behaviour coordination mechanism is at the core of this method. For the case of formation control, behaviour coordination means velocity synthesis.

2.1.4. Bio-inspired Methods and Other Methods

Bio-inspired methods draw ideas from natural swarm behaviours, especially biological phenomena. For example, many bird species fly in flocks, so if UAVs imitate the behaviours of birds, UAV swarms may display similarities to bird flocks [28], [37]. Another interesting example is morphogenesis approach [28], [29]. Morphogenesis, i.e., the formation of the shape of organs, is connected with signal molecules that diffuse among cells, also known as morphogen. Cells adjust their behaviour according to the concentration of morphogen, and eventually the organ will develop a certain shape. Inside a swarm, messages can be propagated in certain pattern between neighbouring UAVs. If all UAVs act properly according to these messages, the swarm will be able to develop a global shape in a self-organised way.

There are also many other types of formation control methods, such as artificial intelligence (AI) methods [35], artificial potential fields [40], [41], [50], graph-based(consensus-based) algorithms [22], [31], and so on.

2.2. Task Allocation

However, for an autonomous swarm, it is equally crucial to optimally schedule tasks and allocate them to individual UAVs. Usually, for UAV, task management is highly related to path planning, after all, most tasks have restrictions on the position and velocity of a UAV. Moreover, in real scenarios, resources such as fuel or power are constrained for a task, making the problem even harder to tackle [21], [53].

One relatively easy way to manage swarm tasks is to take a centralised approach. That is, one UAV works as a central node, schedules tasks and allocates them to others. However, due to single points of failure, this method is not robust. Decentralised methods, where every UAV takes part in the task allocation process, are more resilient but also more complex. Many algorithms have been studied for their usage in swarm task allocation [33], [34], such as game theory, reinforcement learning, etc. In this section, some of them are reviewed.

2.2.1. Algorithms Based on Market Mechanisms

These algorithms imitate marketing processes. A typical example is the auction algorithm [23], [33], [42]. If a UAV has a task, but does not own the requisite resource, it can choose to auction the task. Other UAVs can bid, and the winner is assigned the task. Auction algorithm has clear rules and is easy to perform. It requires good communication between the auctioneer and the bidders.

2.2.2. Consensus-based Methods

In consensus methods, if a UAV wants to take any action, or if it wants the whole swarm to take any action, it firstly negotiates with other UAVs, and they need to reach an agreement on whether or not the action should be taken. If consensus is reached, the action will be taken, otherwise the action is abandoned. It can be seen that consensus algorithms require good communication inside the swarm. Many researchers have studied different types of consensus methods and their variants [19], [26], [32], [36]. The difficult part of a consensus algorithm is for a UAV to decide whether all others have been informed of and have agreed on its proposal, especially in real-world scenarios where network topology is constrained and communication condition is not ideal.

Consensus algorithms have a wide range of applications. They can be used not only

in swarm task allocation, but also in formation control, path planning, and many other types of distributed systems, e.g., a distributed software system [30].

2.2.3. Optimisation Algorithms

Since time and resources are limited for a swarm, optimisation algorithms come into play, so as to plan and allocate tasks in a most optimal way. Note that, optimisation is not used for reaching an agreement on which UAV takes which sub-task, but for finding out the best task division scheme.

A particular type of optimisation methods are swarm intelligence algorithms [3], [10], [21], [33], [44], [45], [53]. Swarm intelligence is not a strict term, but a loose concept that solves problems by modelling a population of agents which can self-organise and interact with each other. After a population is set to some initial state, each individual agent acts by certain principle, eventually, the whole population may achieve specific collective behaviour or reach desired global optimum state. A lot of swarm intelligence methods are inspired by biological swarm behaviours. Swarm intelligence can be used in many problems, and optimisation is just one of them. Some common swarm intelligence optimisation algorithms are genetic algorithm (GA), particle swarm optimisation (PSO) [5], [16], [38], ant colony optimization (ACO), wolf pack algorithm (WPA) [27], [52], and so on.

2.3. Hierarchical Swarm

Despite of lot of research effort put into UAV swarms, they are still not mature for real-world applications. In 2021, Lee, Kuo, Chen, and Chuang [24] reported an experiment where 3 UAVs planned their flight path to avoid no-fly zones and kept their formation with leader-follower method. In 2016, US department of defense conducted a micro-drone swarm test [47]. In the test, 103 small drones were launched from fighter jets.

The drones demonstrated collective decision making, adaptive formation flying, and other swarm behaviours. However, technical details were not disclosed. In 2022, British Army carried out a demonstration where swarms comprised of 4 or 6 drones executed autonomous missions [7]. Again, no technical details. As can be seen from the above examples, current swarms are either quite small, or can only deal with simple tasks. This is not surprising since swarm algorithms are still in their early stage.

2.3.1. Drawbacks of Common Algorithms

Swarm control algorithms can be roughly divided into two categories, centralised ones and decentralised ones. Centralised algorithms usually have a leader UAV as a central node, whereas all other UAVs are managed by the leader. The leader-follower formation control method is an example of centralised algorithm. Such algorithms are simple and efficient, but are vulnerable to single points of failure. Besides, as most computation happens on the central node, it usually needs much more communication and computing power than others. Therefore the swarm size may be limited by the resource on the central node. In contrast, decentralised algorithms treat every UAV equally, and the collective behaviour of the whole swarm is coordinated by all the UAVs involved. Such algorithms are more robust.

In this thesis, decentralised algorithms are further divided into two types. The first type is negotiation-based ones, such as auction algorithms and consensus algorithms. UAVs communicate with each other about what they plan to do next, and make decisions together. Consensus algorithms are powerful, because each UAV knows what others will do, and they have a fine control on task management. However, they impose a high communication and computation burden on swarms, since every UAV cares for the whole swarm. As swarm population increases, the complexity of communication and control goes up sharply. Thus, the swarm size is limited.

Another type of decentralised algorithms is reaction-based ones, such as virtual structure formation control, and most bio-inspired algorithms. In these algorithms, UAVs do not need to reach agreement. They just gather information about the environment and other UAVs, then they react according to preset rules. Such algorithms are highly robust, and do not require much communication or computation resource. They are highly distributed and suitable for large swarms. However, they lack fine and flexible control over the swarm, hence are not versatile and are only adequate for simple tasks. Table 2.1 shows a comparison between different types of algorithms.

Table 2.1.: Comparison of different types of swarm algorithms.

Algorithm Type	Centralised	Negotiation-based	Reaction-based
Robustness	not robust	robust	robust
Algorithm Complexity	simple	complex	intermediate
Performance Overhead	high (central node)	high (all nodes)	low
Supported Tasks	complex	complex	simple
Supported Swarm Size	small	small	large

2.3.2. A Hierarchical Method

This thesis aims to find an algorithm that is robust, suitable for complex tasks, adequate for large swarms, and low in performance overhead. The solution is a hierarchical swarm. The swarm is organised into a tree structure. Each UAV corresponds to a node on the tree structure, and it is in direct control of the UAVs which correspond to its child nodes. Layer by layer, the root node will be controlling the whole swarm in an indirect way, quite similar to the hierarchy in modern companies, governments, or armies. Figure 2.3 shows such an swarm structure. Communication mainly happens between each parent-child node pairs. For most of the time, there is no need for many-to-many communication,

reducing the communication overhead. Task is also solved by hierarchical coordination. A node receives task from its parent, then divides the task, and assigns the sub-tasks to itself and its children. The children then repeat the process. This divide-and-conquer way of task coordination potentially reduces computing overhead. As the tree grows in depth, the number of nodes grows exponentially, therefore large swarm sizes are supported.

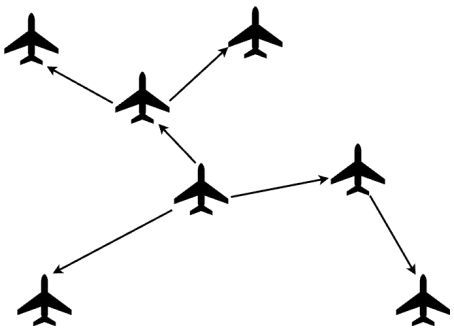

Figure 2.3.: A swarm organised hierarchically into a three-layer tree. Arrows point from parent node to child node.

Actually, researchers have already investigated the usage of hierarchical structure in swarm organisation [11], [43], [51]. In those researches, the hierarchy, which is not necessarily a tree, is predefined and fixed before flight. This leads to reduced flexibility and robustness. In fact, a tree is just a multi-layered leader-follower structure, which is centralised. To mitigate the disadvantages of leader-follower approach induced by centralisation, some researchers have proposed dynamic leader election algorithms [8], [18]. Similarly, to make the swarm tree structure flexible and robust, it needs to be organised dynamically. That is, the UAVs themselves build up the tree, and they are able to repair the tree structure in case any of them fails.

In the next chapters, the algorithm for dynamic hierarchical swarm organisation will be developed and implemented. The task for the swarm is to form designated geometric

shapes. This application scenario is inspired by drone light shows [46]. Currently, drone light shows do not employ autonomous UAV swarms. But this thesis shows the possibility for their usage in the future.

3. Problem Formulation

To develop and demonstrate a swarm algorithm, an application scenario is needed. Inspired by drone shows, where drones form various interesting shapes, in this thesis, the task for a swarm is to form a designated geometric shape. Since the purpose is focused on swarm control and coordination, many other aspects, such as aerodynamics, obstacle avoidance, and power or fuel management, are ignored or simplified. The problem is formulated below.

3.1. UAV Properties

At time point $t = 0$, a set of homogeneous UAVs $W = \{U_i | i = 0, 1, \ldots, N_u - 1; N_u > 0\}$ are located at initial positions $\boldsymbol{P}_i(t = 0) = (x_{i0}, y_{i0}, z_{i0})$. They are already airborne. The taking-off process is not considered.

Each UAV U_i has a unique ID id_i. The ID set has total order, which means IDs are comparable.

GPS receivers are installed, so UAVs can read their positions $\boldsymbol{P}_i(t) = (x_i, y_i, z_i)$. The distance between two UAVs U_i and U_j is

$$d_{ij}(t) = d_{ji}(t) = ||\boldsymbol{P}_i(t) - \boldsymbol{P}_j(t)||. \tag{3.1}$$

They will clash if $d_{ij} \leqslant 2r_{rad}$, where r_{rad} is the radius of a UAV.

UAVs are able to move at arbitrary velocity $\boldsymbol{V}_i(t) = (vx_i, vy_i, vz_i)$, as long as

$$v_i(t) = ||\boldsymbol{V}_i(t)|| \leqslant v_{max}, \tag{3.2}$$

where v_{max} is the given maximum speed of the UAVs. Hovering, i.e., $\boldsymbol{V} = \boldsymbol{0}$, is allowed. In real cases, a UAV can only control its engines or rotors. The actual acceleration and velocity are determined by engine power and aerodynamics. In this thesis, the flight control is greatly simplified, and the UAVs are assumed to be able to change their velocity directly. Besides, it's assumed that the UAVs fly in the same manner in all directions, without the need to care about attitude, which is of course not true for any real UAVs.

UAVs do not known the existence of each other at $t = 0$. UAV U_i can receive all the data sent by UAV U_j if the distance between them is within the given communication range r_{comm}, i.e., $d_{ij} \leqslant r_{comm}$. Latency and bandwidth problems are not considered. At $t = 0$, the initial positions of the UAVs ensure that, any two UAVs are within communication range.

3.2. Representation of Geometric Shapes

A "shape" here is made up of one or multiple lines. While a "line" here is made up of one or multiple connected line segments. A line segment is defined by two points, so a line can be defined by a sequence of points.

Let L represent a sequence of N_p different points $\boldsymbol{P}_0, \boldsymbol{P}_1, \ldots, \boldsymbol{P}_{N_p-1}$, where $\boldsymbol{P}_k = (x_k, y_k, z_k), 0 \leqslant k \leqslant N_p - 1$, and $N_p \geqslant 2$. L defines $N_p - 1$ connected line segments, which form a "line". A point $\boldsymbol{P} = (x, y, z)$ is said to be on line L if it falls onto any of the line segments, that is, there exist an integer $i, 0 \leqslant i < N_p - 1$, and a real number

$c, 0 \leqslant c \leqslant 1$, such that

$$(x - x_i) = c(x_{i+1} - x_i), \tag{3.3}$$

$$(y - y_i) = c(y_{i+1} - y_i), \tag{3.4}$$

$$(z - z_i) = c(z_{i+1} - z_i). \tag{3.5}$$

A shape S is a set of N_l lines $\{L_m | m = 0, 1, \ldots, N_l - 1; N_l > 0\}$. $\boldsymbol{P}$ is said to be on S if it is on any of the lines.

Lines defined above can not represent curves. However, a curve can be approximated by a large amount of short line segments, in the commonly used way how a circle is approximated by a polygon.

3.3. Swarm Tasks

A task TSK contains a unique task ID tid, a shape S and a time period Δt.

Let W be a set of UAVs. At some time point t', task TSK is sent by GCS to an arbitrary member of W. The mission for W is, all the UAVs move onto the shape and stay for Δt. That is, the task is said to be successfully executed if there exists a time point $T, t' \leqslant T < +\infty$, such that for any time point $t \in [T, T + \Delta t]$ and for any UAV $U \in W$, U is on S.

Collisions shall be avoided. That is, at any time point t, for any two UAVs U_i and U_j, the distance between them shall satisfy $d_{ij} > 2r_{rad}$.

Additionally, UAVs should be distributed as evenly as possible along the lines. This means to minimise the mean square root of UAV intervals along the lines. This is not the primary goal of the task, so it is not formally stated here. But later in section 5.3, an algorithm will be implemented to address this goal.

4. Algorithm and Software Design

In this chapter, the swarm algorithm is shown step by step to solve the problem stated in chapter 3. The module-level architecture of the onboard software based on this algorithm is also designed and detailed.

4.1. Swarm Algorithm Design

4.1.1. Dynamic Organisation of Tree Structure

Before a group of UAVs is able to handle any task, the UAVs should firstly organise themselves into a tree structure. Since each UAV can be viewed as a single-node tree, the problem can be viewed as building a single large tree from all existing small trees. Before continuing, some terms, assumptions and key designs must be clarified.

Remark 4.1. *A swarm is defined as all the UAVs on the same tree. A sub-swarm is defined as all the UAVs on a sub-tree.*

In the context of this thesis, the term "tree" is equivalent to the term "swarm", and they will be used interchangeably. Similarly, term "node" is sometimes used to refer to a UAV inside a swarm. Building a single tree from small trees means building a single swarm from existing small swarms.

Remark 4.2. *Inside a tree, each child-parent node pair forms a connection. The child shall ensure that its distance from the parent is within communication range.*

A node may be in connection with multiple child nodes, but with at most one parent node. For each child-parent pair, the child needs to track the parent since it is responsible for keeping the communication range.

Remark 4.3. *Child node and parent node send connection messages to each other periodically, in order to exchange necessary data.*

To distinguish from the root UAV of the whole tree, the root node of a sub-tree will be called the top node of that sub-tree.

Remark 4.4. *Inside a tree, for each child-parent node pair, the sub-tree topped by the child node is called a "child sub-tree" of the parent node.*

Remark 4.5. *The size of a tree is the number of its nodes, i.e., the population of the swarm.*

Remark 4.6. *A set of existing trees can be strictly ordered by tree size and root node UAV ID.*

Trees of different sizes can be ordered by size. Since UAV IDs are unique and comparable, two trees of the same size can be distinguished and ordered by root node UAV ID.

Remark 4.7. *For a node on a tree, the ID sequence of all its ancestor nodes and itself is called the NID (meaning "node ID") of the node.*

If the root node of a tree is UAV U_i, U_i has child U_j, and U_j has child U_k, then the NID of U_k is $[id_i, id_j, id_k]$. An NID always starts with the ID of the root node of the tree. The root node always knows its NID. If a parent node knows its NID and sends its NID to its children, the children will also known their NIDs.

Remark 4.8. *If every parent node sends its NID to its child nodes, recursively all nodes will known their correct NIDs.*

For a leaf node, i.e., node that has no children, its sub-tree size is one, since the only member of the sub-swarm is itself. For a parent node, if all its children know their respective sub-tree sizes and send the data to the parent node, the parent node can then calculate its own sub-tree size, which is the sum of child sub-tree sizes plus one.

Remark 4.9. *If every child node sends its sub-tree size to its parent node, recursively all nodes will known their correct sub-tree sizes.*

For the root node, its sub-tree size is also the size of the whole tree.

Remark 4.10. *If every parent node sends tree-size data to its child nodes, recursively all nodes will known the correct tree size.*

The ultimate purpose of a swarm is to carry out tasks assigned by GCS. For any swarm member, it may be handling some task, or it may be in free state.

Remark 4.11. *The "current task ID" of a UAV is denoted by the* ***tid*** *of the task, or is* ***None*** *if the UAV is free.*

In order for the UAVs to discover each other and to obtain the status of each other, they need to broadcast their information to nearby UAVs.

Remark 4.12. *A UAV broadcasts its NID, position, velocity, swarm size and current task ID periodically.*

Now the basic characteristics of a swarm are established, and a UAV knows well about other UAVs in its neighbourhood. Initially, each UAV is a single-node tree. To build a single tree from multiple existing trees, two ideas are feasible: swarm-merging and swarm-switching.

- Swarm-merging. For a swarm, if there are bigger other swarms nearby, the root node, leading the whole current swarm, joins the biggest swarm.

- Swarm-switching. For a UAV in a swarm, if there are bigger other swarms nearby, the UAV, leading its sub-swarm, leaves the current swarm and joins the biggest swarm.

In swarm-merging method, a smaller tree joins a bigger tree as a whole. Trees are stable. Existing trees will not collapse during merging process unless failure occurs. In swarm-switching method, trees are not stable since nodes may leave at any time. However, it's hard for swarm-merging method to handle the case where the nearby biggest swarm is found by a non-root node, but is out of the communication range of the root node. Whereas for tree-switching method, the node which finds the biggest swarm just joins the new swarm, and other nodes of the old swarm will almost certainly find the new swarm since they are at least in the communication range of that switching node.

The swarm-switching method is adopted. The pseudocode is shown in algorithm 4.1. The variable $\boldsymbol{self}$ in the algorithm refers to the UAV that runs the algorithm. The algorithm is only run when a UAV is free, i.e., when its current task ID is $\boldsymbol{None}$. The root node ID is used to tell which tree a node is currently on, it is contained in the NID of a node. When comparing two swarms, the swarm of bigger size is ordered before the one of smaller size. If two swarms are of the same size, the swarm with smaller root ID is ordered before the one with bigger root ID.

In algorithm 4.1, after a UAV U_i sets another UAV U_j as its parent node, U_i should send a request message to U_j. U_j can accept the request, or if U_j is in inappropriate state, it can reject the request by sending U_i a reject message. If U_i receives a reject message, it separates from U_j and the sub-tree led by it becomes a new independent tree.

If all free UAVs within communication range keep running algorithm 4.1, they will eventually form a single tree. Due to the latency of data flow inside a tree, information

Algorithm 4.1 Swarm-switching algorithm.

Require: *self* is free

1: **function** SwitchTree(*self*)
2: *root_self* ← *self*.get_root_id()
3: *candidates* ← all nearby UAVs
4: *candidates*.filter_by(*candidate* is free)
5: *candidates*.filter_by(*candidate*.get_root_id() ≠ *root_self*)
6: *candidates*.filter_by(swarm of *candidate* ordered before swarm of *self*)
7: **if** *candidates* is not empty **then**
8: *candidates*.sort_by(ordering of their respective swarms)
9: *new_swarm_root* ← *candidates*[0].get_root_id()
10: *candidates*.filter_by(*candidate*.get_root_id() = *new_swarm_root*)
11: *candidates*.sort_ascending_by(distance(*candidate*, *self*))
12: *self*.set_parent(*candidates*[0])
13: **end if**
14: **end function**

stored by a node, such as swarm size, may not be updated timely. Theoretically, this may cause nodes to join a tree which is not the biggest one. However, information will eventually propagate throughout a tree, and the algorithm is expected to converge.

4.1.2. Task Coordination

Tasks are handled hierarchically. The root node divides the shape of a task into parts, one for itself, and the others sent to its children. Its children then repeat this process, until every node gets a portion of the shape. Execution of a task means adjusting the velocity appropriately, so the UAV flies onto its part of the shape, and stays there for a required time duration.

In this thesis, based on the context, the word "task" may refer to three cases: the task sent by the GCS to a swarm; a portion of the task that is allocated to a sub-swarm; a portion of the task that is allocated to a node. The word "sub-task" is also used to refer to the latter two cases.

When the UAVs are in free state, the swarm is not stable. Nodes may join or leave a swarm. It is hard for an unstable swarm to handle tasks properly. So after a swarm receives a task, it needs to firstly fix itself before it starts to handle the task. An inner state is needed for every UAV to represent whether it has a task.

Remark 4.13. *The current task ID of a UAV is none* $\iff$ *the UAV is in state* ***Free****; the current task ID of a UAV is some* ***tid*** $\iff$ *the UAV is in state* ***InTask****.*

Remark 4.14. *If a UAV is in state* ***InTask****, it shall not leave the swarm; it shall also reject any request that sets it as parent.*

It can be seen that if all members of a swarm is in ***InTask*** state, the swarm is stable. So after a task is received, an approach is needed to transform the whole swarm into ***InTask*** state.

Remark 4.15. *Based on the execution state of the task,* $\boldsymbol{InTask}$ *state has three sub-states:* $\boldsymbol{InTask}(\boldsymbol{InProgess})$, $\boldsymbol{InTask}(\boldsymbol{Success})$ *and* $\boldsymbol{InTask}(\boldsymbol{Failure})$.

Remark 4.16. *If a non-root UAV receives a task message sent by GCS, it relays the GCS task to its parent node. If root UAV receives a GCS task, it turns from* $\boldsymbol{Free}$ *state into* $\boldsymbol{InTask}(\boldsymbol{InProgress})$ *state.*

A child node can receive the current task ID of its parent node. It changes its state according to this information.

Remark 4.17. *A child node turns into* $\boldsymbol{Free}$ *state if the current task ID of its parent is* $\boldsymbol{None}$*; and turns into* $\boldsymbol{InTask}(\boldsymbol{InProgress})$ *state if the current task ID of its parent is some* $\boldsymbol{tid}$.

All UAVs in a swarm will recursively align their $\boldsymbol{Free}/\boldsymbol{InTask}$ states with the root UAV. After the root node receives a GCS task, the whole swarm will gradually transform into $\boldsymbol{InTask}(\boldsymbol{InProgress})$ state. Before this alignment is complete, the tree structure may change, besides, the swarm size and sub-swarm size data stored by a UAV may be incorrect, so task division is not possible. Note that, for non-root nodes, $\boldsymbol{InTask}$ means it has received the task ID, but may or may not have actually received a sub-task.

After the root node turns into $\boldsymbol{InTask}(\boldsymbol{InProgress})$, it needs to know whether all the other UAVs have finished aligning their states, in order to start to divide the task and allocate sub-tasks. In addition, once a task is fully divided and allocated to every node, the root node needs to know the sub-task results of all other UAVs, in order to decide the final result of the task.

Remark 4.18. *A node in* $\boldsymbol{InTask}(\boldsymbol{InProgress})$ *turns into* $\boldsymbol{InTask}(\boldsymbol{Success})$ *if all nodes in its sub-tree succeeded in executing the task; A node in* $\boldsymbol{InTask}(\boldsymbol{InProgress})$ *turns into* $\boldsymbol{InTask}(\boldsymbol{Failure})$ *if any node in its sub-tree failed in executing the task.*

Again, those information can be collected in a recursive way. For this purpose, besides the state of a node itself, the overall state of the sub-tree topped by the node is also needed.

Remark 4.19. *At any time point, a sub-swarm is in one of the six task states:* ***None****,* ***Recv****,* ***Algn****,* ***Allc****,* ***Succ****, and* ***Fail****.*

The meaning of the six task states are explained in table 4.1. The top node of a sub-tree is responsible for calculating the sub-swarm task state and sending it to its parent node. The calculation process is shown in algorithm 4.2. It is seen that the top node needs data from its children to do the calculation, which means recursion. A leaf node has no children. Once it receives a task ID and turns into ***InTask***(***InProgress***), its sub-tree, which contains only itself, is in task state ***Algn***. The node then reports ***Algn*** to its parent. If all children of the parent report ***Algn***, the parent knows that all its child sub-trees have been aligned to ***InTask***, and the parent's sub-tree is in task state ***Algn***. Recursively, the root node will eventually know that the whole swarm has been aligned. Sub-task results are propagated up the tree in a similar way.

With the sub-swarm task states defined above and algorithm 4.2, it is easy to figure out the complete state transition conditions of a node and what a node should do in each of the states.

Remark 4.20. *If all children of the root node report sub-tree task state* ***Algn****, then all nodes are in* ***InTask****, and the task can be divided and allocated layer by layer.*

When a node with N_c children divides a received task, the task is divided into $N_c + 1$ sub-tasks, one for itself and N_c for its child sub-trees. The weight of each sub-task shall be proportional to the number of UAVs executing it.

Remark 4.21. *If all children of a node report sub-tree task state* ***Allc****, it means the sub-task allocation process of that node is finished.*

Algorithm 4.2 The process of the top node calculating the sub-swarm task state.

```
1: function CalculateSubswarmTaskState(self)
2:     state_self ← self.get_node_state()
3:     if state_self = Free then
4:         return None
5:     else if state_self = InTask(Success) then
6:         return Succ
7:     else if state_self = InTask(Failure) then
8:         return Fail
9:     else                                   ▷ state_self = InTask(InProgress)
10:        if any child sub-swarm in None or Recv then
11:            return Recv
12:        else          ▷ No children, or all child sub-swarms have been aligned
13:            if self has received a sub-task then
14:                return Allc
15:            else
16:                return Algn
17:            end if
18:        end if
19:    end if
20: end function
```

Table 4.1.: The meaning of the six task states of a sub-swarm.

None	The top node is in *Free* state.
Recv	The top node has received the task ID and is in *InTask*(*InProgress*), but has not confirmed that all the other nodes of the sub-tree are in *InTask*.
Algn	The top node has confirmed that all nodes of the sub-tree are in *InTask*, and it has not received a sub-task from its parent.
Allc	The top node has confirmed that all nodes of the sub-tree are in *InTask*, and it has received a sub-task from its parent.
Succ	The top node finds that all nodes of the sub-tree have succeeded.
Fail	The top node finds that at least one node of the sub-tree has failed.

Remark 4.22. *If a node fails a task, or if it loses connection with a child which is in* ***InTask*** *state, or if any of its children reports sub-tree task state* ***Fail****, the node turns into* ***InTask***(***Failure***) *state.*

Remark 4.23. *If a node succeeds in a task, i.e., has stayed long enough on some shape, and all its children report sub-tree task state* ***Succ****, the node turns into* ***InTask***(***Success***) *state.*

If the root node turns into ***InTask***(***Failure***) or ***InTask***(***Success***) state, the whole GCS task is failed or successful, and the root node will turn into ***Free*** to wait for the next GCS task.

Figure 4.1 shows the state transition conditions of a node. Table 4.2 summarises what a node shall do in each of the states. Figure 4.2 gives an example of how node state changes in a swarm during a task.

It is worth noticing that, for a task with shape S and time duration Δt, the root

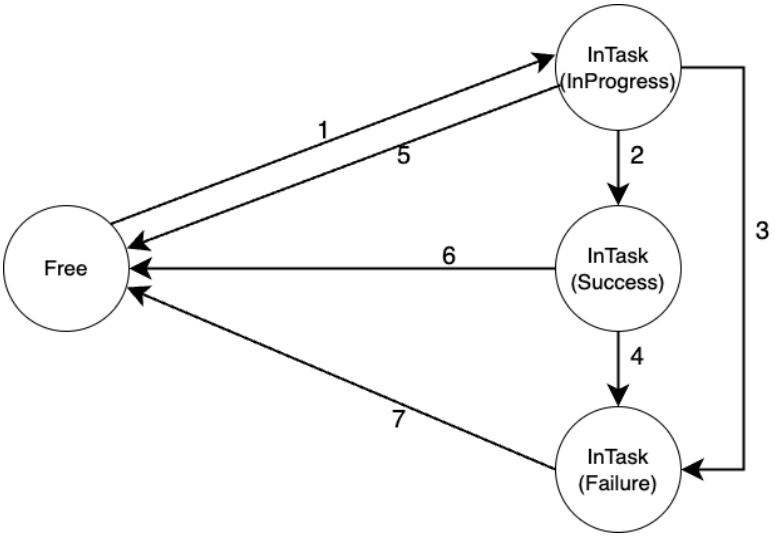

Figure 4.1.: State transition conditions for a node. [1] The root node receives a GCS task; or a non-root node receives the current task ID ***tid*** from its parent. [2] A node has succeeded and all its children report sub-swarm task state ***Succ***. [3, 4] A node has failed, or it loses connection with a child which is in ***InTask*** state, or any of its children reports sub-swarm task state ***Fail***. [5] A non-root node receives the current task ID ***None*** from its parent. [6] A non-root node receives the current task ID ***None*** from its parent; or the root node decides that the whole task has succeeded. [7] A non-root node receives the current task ID ***None*** from its parent; or the root node decides that the whole task has failed.

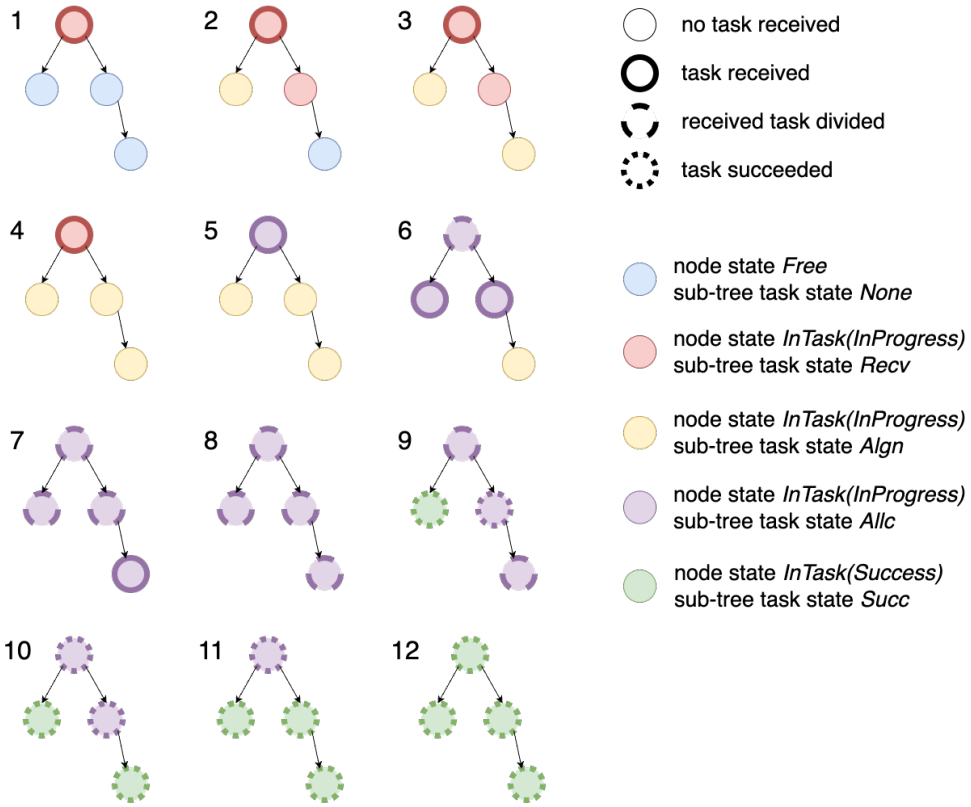

Figure 4.2.: An example of state transitions of a 4-UAV swarm during a task.

Image 1 shows the root have received a GCS task.

Image 1-3 depict how the nodes align their states with the root.

Image 3-5 depict how the information of alignment completion propagates up the tree.

Image 6-8 depict how each node divides and allocates tasks down the tree.

Image 9 and 10 show nodes finish their tasks.

Image 10-12 depict how the root knows that all nodes have succeeded.

Table 4.2.: What a node does in different states.

$\boldsymbol{Free}$	A node tries to join a bigger swarm.
$\boldsymbol{InProgress}(\boldsymbol{InProgress})$	A node waits until all its children are in $\boldsymbol{InTask}$ state (i.e., all children report sub-tree task state $\boldsymbol{Algn}$) and a task has been allocated to it; then it divides the task, allocates sub-tasks, and executes its own sub-task.
$\boldsymbol{InProgress}(\boldsymbol{Success})$	The root node switches to $\boldsymbol{Free}$ state; a non-root node waits for its parent to change state.
$\boldsymbol{InProgress}(\boldsymbol{Failure})$	The root node switches to $\boldsymbol{Free}$ state; a non-root node waits for its parent to change state.

node in $\boldsymbol{InTask}(\boldsymbol{Success})$ state only means that all nodes have stayed on $\boldsymbol{S}$ for at least Δt, however, the overlapping part of their stay duration may not be long enough. This problem is ignored, for it deviates from the main focus of this thesis and requires complex algorithm to solve.

4.2. Velocity Control

Velocity control is related to formation control and path planning. To maintain the connection with the parent node, a UAV also needs to track its parent. To execute a task, a UAV needs appropriate velocity to move around or to stay at some position. Besides, collision with other UAVs shall be avoided. All these requirements contribute to the complexity of the calculation of velocity. A behavioural approach is adopted. The structural design of velocity control is illustrated in figure 4.3.

If a UAV has a task, the task velocity is needed by task execution. For non-root

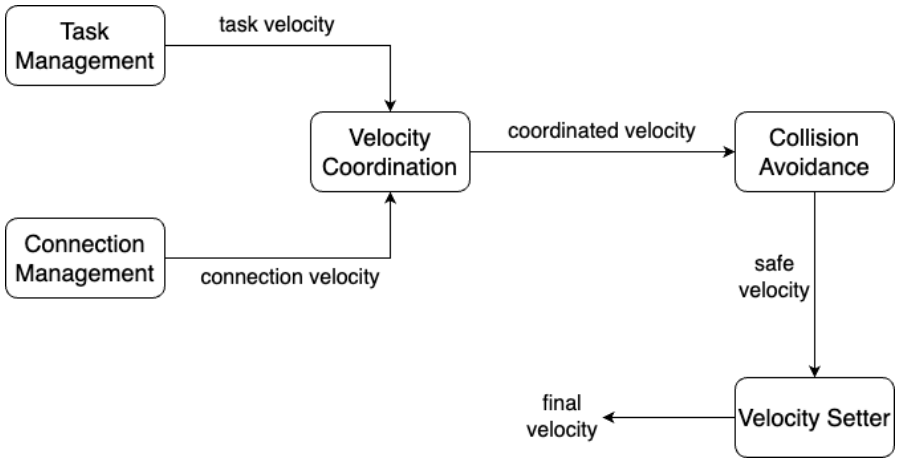

Figure 4.3.: The structure of velocity control.

nodes, the connection velocity constrains the UAV from flying too far away from its parent, so as to maintain the connection with its parent. These two velocities goes into a coordinator, which generates a coordinated velocity. The basic idea of the velocity coordination is, task velocity dominates if the parent is within a safe distance, and connection velocity dominates if the parent is on the verge of losing connection. The collision avoidance module takes the coordinated velocity as input applies modifications to prevent collisions with nearby UAVs, and outputs a safe velocity. The velocity setter corresponds to the flight control module on a real UAV. It takes the safe velocity, clamps it if it exceeds the maximum UVA speed v_{max}, and sets it as the final desired velocity.

Figure 4.3 shows only the idea and structure of velocity control. The exact algorithms of each part are up to the implementation.

4.3. Software Architecture Design

To run and test the designed swarm algorithm on a UAV, a piece of onboard software which implements the algorithm is needed. In this section, a top-down modular archi-

tecture of the software is designed.

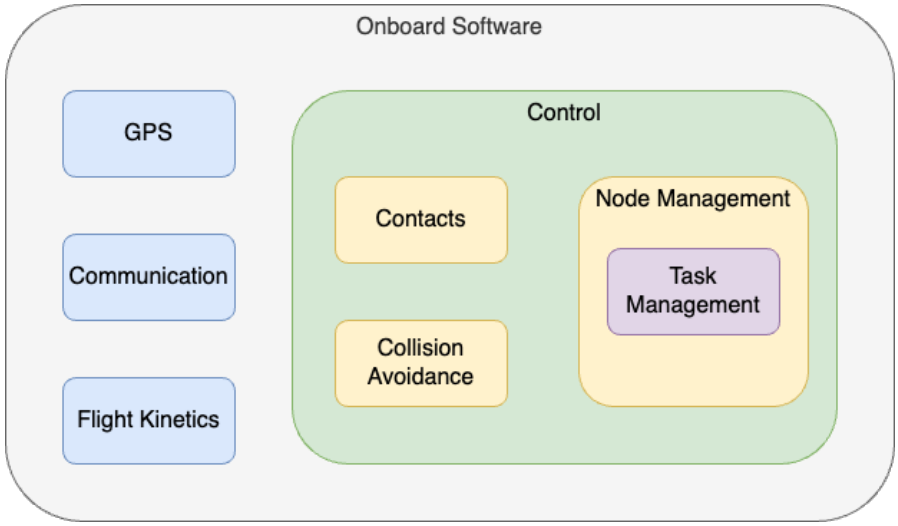

Figure 4.4.: The architecture of the onboard software.

As shown in figure 4.4, the top-level modules are GPS module, Communication module, Flight Kinetics module, and Control module. Among them, the Control module implements the swarm algorithm, while all the other three are hardware interfaces that interact with sensors, radio frequency (RF) devices, or actuators. Since this thesis runs simulations, there is no hardware, so the three modules actually interact with simulation infrastructure.

The Control module is where the designed swarm algorithm is implemented. It contains three child modules: Contacts module, Collision Avoidance module, and Node Management module. The Contacts module monitors nearby UAVs that are in communication range. The Collision Avoidance module prevents collision with other UAVs by adjusting velocity. The Node Management module handles connections with parent and children, and manages node state transitions. It further contains the Task Management module, which divides and executes tasks.

Table 4.3.: The functionality of each module.

GPS	Interface to the GPS sensor. Provides position information to other modules.
Communication	Interface to the RF device. Provides received messages from nearby UAVs to other modules, and sends messages generated by other modules to nearby UAVs.
Flight Kinetics	Interface to the flight control system. Sets the UAV velocity to the value required by other modules. Corresponds to the "Velocity Setter" in figure 4.3.
Control	Implements the swarm algorithm.
Contacts	Monitors all the other UAVs within communication range. Provides notice when a UAV loses contact.
Node Management	Manages the connections with parent and children, and manages the transitions of node states (figure 4.1). This is the core module of the swarm algorithm. Responsible for generating connection velocity and coordinated velocity (figure 4.3).
Task Management	Manages a received task. Responsible for dividing a task and generating task velocity (figure 4.3).
Collision Avoidance	Prevents the UAV from colliding with neighbour UAVs (figure 4.3).

Table 4.3 lists the functionality of each module. Figure 4.5 shows the sequence diagram of the onboard software. It is a single-thread design. The main part of the software is an infinite loop. Inside the body of the main loop, the Control module is responsible for updating the inner status of the UAV and controlling the UAV. The detailed sequence diagram of the Control module is shown in figure 4.6. The detailed sequence diagram of the Node Management module is shown in figure 4.7. These sequence diagrams show how the modules carry out the designed algorithm.

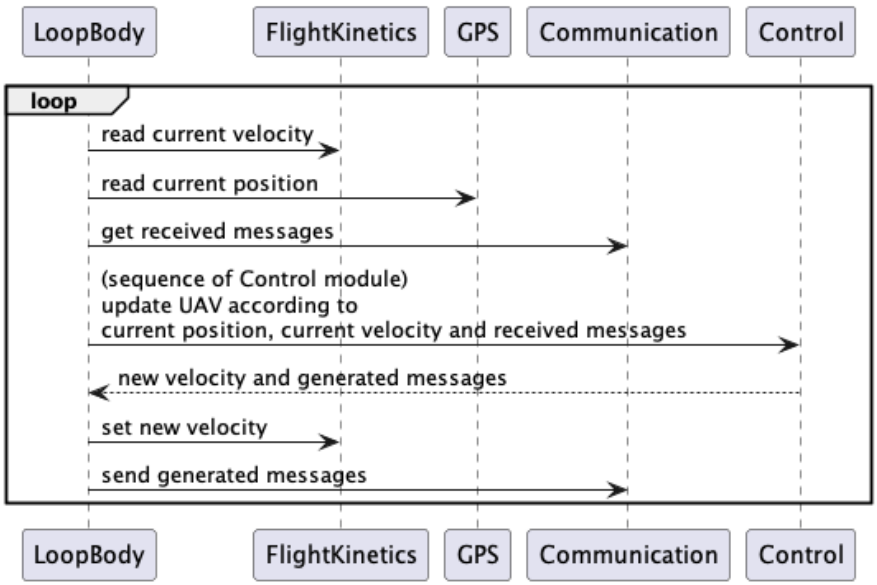

Figure 4.5.: Sequence diagram of the onboard software.

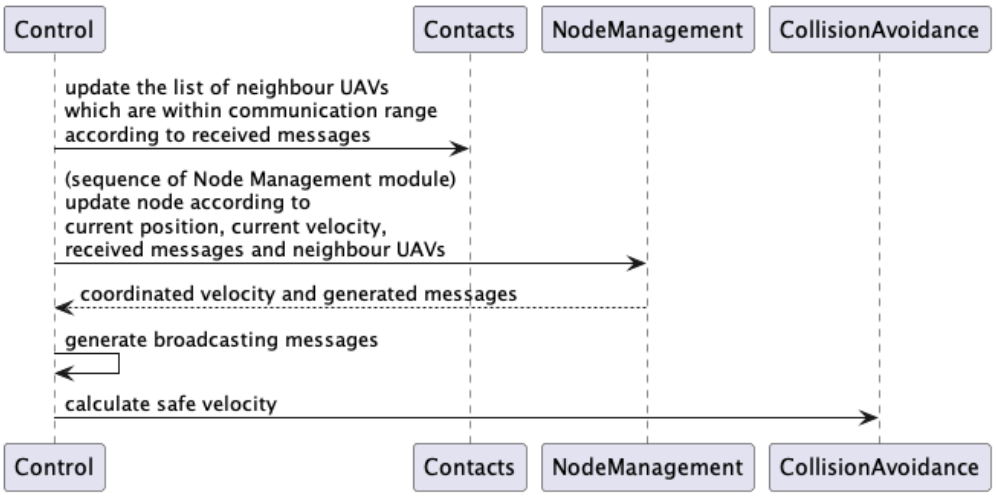

Figure 4.6.: Sequence diagram of the Control module.

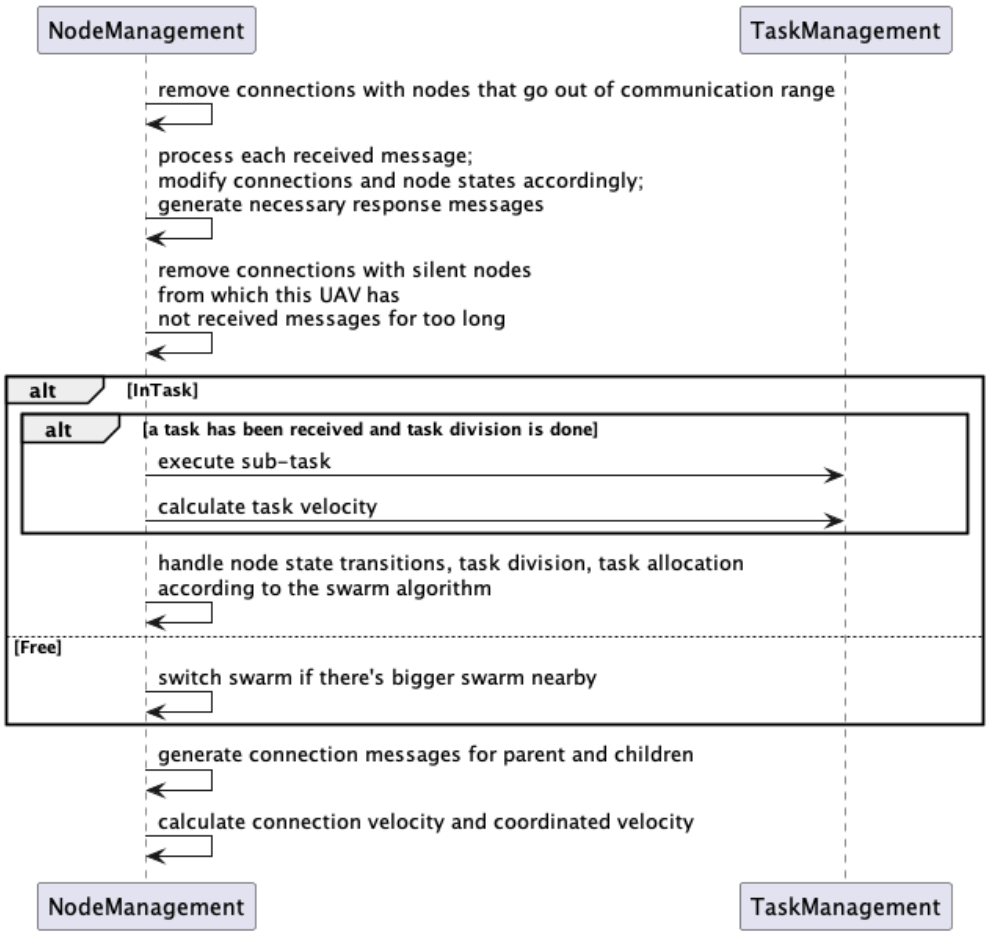

Figure 4.7.: Sequence diagram of the Node Management module.

5. Implementation Details

With the design in chapter 4, it is not hard to develop an implementation of the swarm control and task coordination algorithm. Specific details of different implementations may vary, but as long as they conform to the design, they should be able to solve the problem raised in chapter 3, and demonstrate the feasibility of the design of a dynamic hierarchical swarm.

In this thesis, the design is implemented in the Rust programming language [39]. Rust is a relatively young programming language. The first stable version of Rust was released in 2015, less than 10 years ago. However, it soon became popular due to its safety, performance, modern syntax, and powerful compiler. In 2022, Rust became the third language for Linux kernel development. Rust excels in system programming and embedded programming, so it is very suitable for writing code which runs on a UAV controller.

The GitHub link of the project is `https://github.com/NGC0001/AutoSwarm`. The project has more than 2000 lines of source code. Appendix A gives an overview of the code. Some of the interesting implementation details are introduced in this chapter.

5.1. Messages

Messages are sent by a UAV to other UAVs, or by the GCS to a UAV. A message is an object of struct type Msg. Msg has 3 fields as below.

- An object of struct type *NodeDesc* which contains the basic information of the sender. *NodeDesc* further has the following member fields.
  - NID of the sender.
  - Position of the sender.
  - Velocity of the sender.
  - The size of the swarm it is in.
  - Its current task ID. The type of this field is *Option*<*u*32>, with *None* meaning free and *Some*(*tid*) meaning some task of ID *tid*.
- A list of UAV IDs designating the receivers of the message.
- The message body of enum type *MsgBody*.

Different variants of *MsgBody* represent different purposes of the message, and contain different information. Some important variants are as follows.

- *Empty*. The message contains no extra data. This variant is used for broadcasting basic status of the sender. The information contained in the *NodeDesc* is sufficient. The receiver list is empty.
- *Connection*. This is a connection message. It is used for periodic exchange of information between a child-parent pair. The information sent by a child to its parent includes sub-swarm size and sub-swarm task state.
- *Join*. The sender requests to set the receiver as its parent node.
- *Accept*. Response to *Join*. The sender accepts the receiver's request.
- *Reject*. Response to *Join*. The sender rejects the receiver's request.

- ***Leave***. The sender informs the receiver, which is the current parent of the sender, that it is leaving the current swarm.

- ***Task***. This message contains a task sent by the GCS.

- ***Subtask***. The sender allocates a sub-task to the receiver, which is a child of the sender.

5.2. Limit on Child-Adding Rate

Initially at time $t = 0$, all UAVs are single-node trees. After they are aware of the existence of each other, they start to execute the swarm-switching algorithm. According to algorithm 4.1, since all existing trees are of size 1, the UAV with the smallest ID becomes the switching target of all other UAVs. Hence, it is quite likely that the resulting swarm tree is a very flat one, where the UAV with the smallest ID is the root, and it has many child nodes. If a node has too many children, the communication and computation overhead of the parent node will be high.

A simple solution to this problem is to set limit on the rate of adding child. Consider three UAVs U_i, U_j, and U_k. U_i is closer to U_j than to U_k. U_i and U_j both send ***Join*** messages to U_k. U_k accepts U_j, but rejects U_i due to the limit on child-adding rate. Then the next time U_i tries to join the tree of U_k, it finds that U_j is also on the tree. Since U_j is closer, according to algorithm 4.1, U_j is a better choice than U_k, so it sends a ***Join*** message to U_j, instead of to U_k. In this way, the probability is greatly reduced for U_k to accumulate too many children.

At time point t, suppose UAV U_i added child at time point $\{t_l | l = 1, \ldots, L; 0 < t_l < t\}$. The child-adding rate for U_i is

$$A_i(t) = \sum_{l=1}^{L} e^{\frac{t_l - t}{t_s}}, \tag{5.1}$$

where t_s is a configurable time scale factor. If $A_i(t)$ is greater than the preset limit A_L, U_i rejects any *Join* message at time t.

In the software, $A_i(t)$ is easier to calculate than it seems to be. During the time period from $t1$ to $t2$, if no child is added, $A_i(t)$ has the following property

$$A_i(t') = A_i(t1)e^{\frac{t1-t'}{t_s}}, t1 < t' < t2. \tag{5.2}$$

With this property, the calculation and utilisation of child-adding rate is shown in algorithm 5.1.

5.3. Task Division

For general swarm tasks, a good task allocation algorithm ensures that work load is reasonably balanced across the swarm. Besides, the algorithm shall also be fast and efficient, to save computing power and increase swarm reaction speed. However, task allocation algorithms usually require sophisticated optimisation methods [25], [33].

For a task defined in this thesis, a shape is divided, and different parts are allocated to different UAVs. A good shape division algorithm ensures that UAVs are distributed evenly on a shape, that the flight path for each UAV is optimal, and that each child is within the communication range of its parent. However, the main goal of the implementation is to demonstrate that hierarchical task coordination is viable, not to optimise the process of coordination. Therefore, this implementation only focuses on the even distribution problem, with all other problems ignored.

The shape division works in two steps. First, calculate how many UAVs are needed by each line of the shape. Next, decide which child sub-swarm goes to which lines according to the calculation result.

The first step works in a way similar to algorithm 5.2. In the algorithm, the line_length_of function calculates the length of a line, which is the sum of the lengths of individual line

Algorithm 5.1 Limit on child-adding rate.

```
1: interval ← time interval of software main loop
2: tscale ← time scale factor
3: limit ← preset child-adding rate limit
4: rate ← 0
5: t ← current system time
6: while true do                          ▷ The main loop of the onboard software
7:     t_prev ← t
8:     t ← current system time
9:     rate ← rate × exp((t_prev − t)/tscale)
10:    for msg in received Join messages do
11:        if rate < limit then
12:            accept(msg)
13:            rate ← rate + 1
14:        else
15:            reject(msg)
16:        end if
17:    end for
18:    sleep_for(interval)
19: end while
```

segments. The returned *dist_uavs* is an array containing the number of UAVs needed by each line. This heuristic algorithm does not ensure that the UAVs are distributed in the most even way, but it is simple and straightforward.

Algorithm 5.2 Calculation of UAV distribution among the lines.

Require: number of UAVs no less than number of lines

```
1:  function CalculateUavDistribution
2:      arr_lines ← the array of lines of the shape
3:      num_uavs ← number of UAVs in the sub-swarm
4:      num_lines ← number of lines in arr_lines
5:      dist_uavs ← an empty array
6:      for u in range [0, num_lines) do
7:          dist_uavs.push_back(1)                ▷ Each line needs at least 1 UAV
8:      end for
9:      for u in range [num_lines, num_uavs) do
10:         i ← argmax_{j∈[0,num_lines)} line_length_of(arr_lines[j])/dist_uavs[j]
11:         dist_uavs[i] ← dist_uavs[i] + 1
12:     end for
13: end function
14: return dist_uavs
```

The second step is to allocate the top node and its child sub-swarms onto the lines according to the numbers of UAVs needed by the lines. No optimisation is performed for this step. The lines and the child sub-swarms are trivially matched one by one. A line may be split where needed. Figure 5.1 illustrates how this step works.

After the shape division process has been performed layer by layer down the tree, each node gets its own line. It can choose a suitable point on the line as its target point, e.g., the middle point of the line.

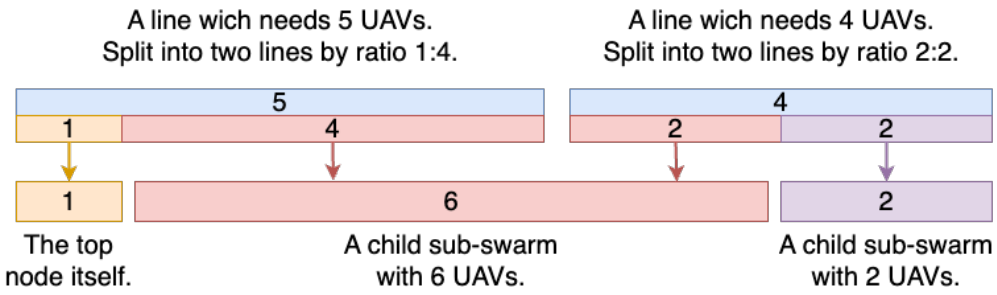

Figure 5.1.: A shape with two lines is divided into 3 parts, one for the top node itself, the other two for the two child sub-swarms.

5.4. Testing

Testing is an important part of software development. However due to a tight schedule, this implementation has not been systematically tested. The overall correctness is ensured by the following two aspects.

1. Black box testing. Simulations are carried out in chapter 6. The results turn out as expected.

2. The safety features of Rust and the powerful static analysis of Rust compiler. As long as unsafe blocks are not used, Rust ensures that the code is memory safe. Whereas in programming languages such as C and C++, memory safety is the main source of bugs.

Nevertheless, small bugs may still exist in the code. So systematic testing, especially unit testing, should be carried out in the future.

6. Simulation and Evaluation

In this chapter, simulations are carried out to verify the swarm algorithm design and implementation. Firstly, the simulation bed is introduced, which is a simulation platform developed as a part of this project. Then, the results of two simulation cases are presented and analysed. Lastly, the swarm algorithm and the implementation are evaluated based on the simulations.

6.1. The Simulation Bed

There are already existing simulation platforms for testing UAV swarms [13]. However, most of these platforms are too complex and not suitable for the specific problem stated in chapter 3. Hence, a simple but sufficient simulation bed is developed.

During a simulation of an N-sized swarm, there will be $N + 1$ processes, as shown in figure 6.1. N processes are for the N UAVs, and the remaining one is for the simulation bed. Currently, all the processes are single-threaded. Each UAV process executes a copy of the onboard software implemented in chapter 5, with its own UAV-specific arguments such as its unique UAV ID. The simulation bed process communicates with the UAV processes through non-blocking Unix sockets. The main functionalities of the simulation bed are as follows.

- Receiving the velocity values set by a UAV, carrying out kinematic integration, and sending the calculated positions back to the UAV.

- Collecting the messages sent by the UAVs and dispatching the messages to their recipient UAVs.

- Detecting collisions and removing crashed UAVs.

- Sending GCS tasks to the swarm.

- Dumping the states of the UAVs periodically to an output file.

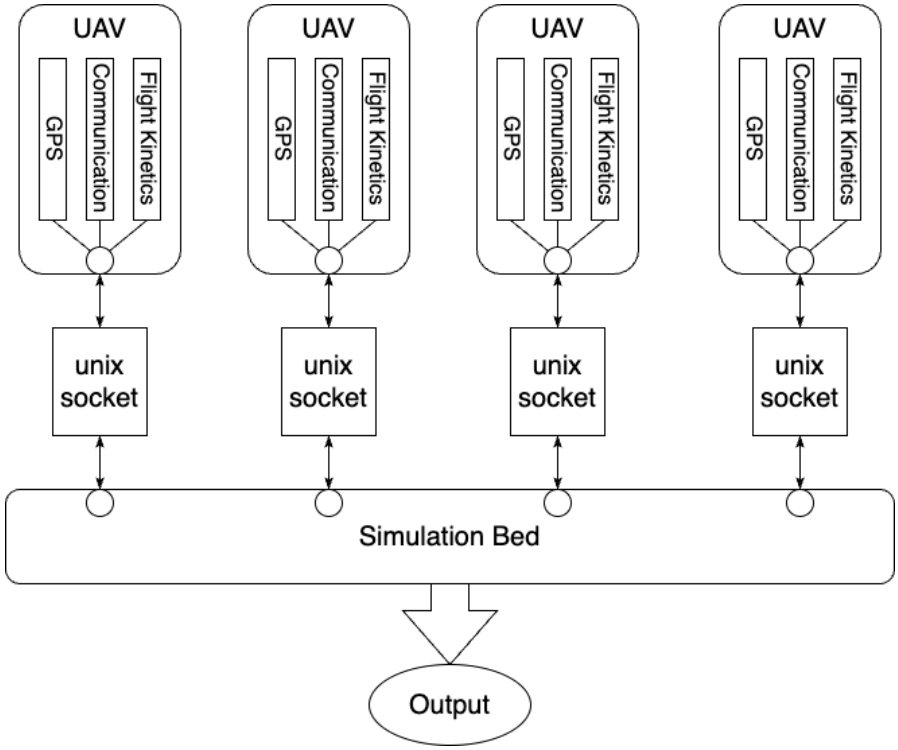

Figure 6.1.: UAV processes and the simulation bed process.

6.2. Simulation Results

In the simulations in this section, the radius of a UAV is $0.1m$, which is similar to the size of a DJI Mini 2 [15] drone. The maximum speed of a UAV is $4m/s$. The communication range of a UAV is $30m$. In each simulation case, 10 seconds after the start of the simulation, a task will be sent by the simulation bed to 1 or 2 of the UAVs.

6.2.1. Simple Line Task

In this case, 10 UAVs are commanded to form a straight line from point $(0m, 10m, 10m)$ to point $(0m, 20m, 10m)$. Figure 6.2 to 6.7 show the swarm organisation process and the task execution process.

In figure 6.2, at simulation time $0.1s$, 10 UAVs are at their initial positions, each of them being a separate single-node swarm. Figure 6.3 and figure 6.4 show the UAVs organise themselves into a single swarm after some attempts. The tree structure plots are based on the NIDs of the UAVs. During the swarm organisation process, tree-switching algorithm is executed. The NIDs of UAVs may change constantly, until stable connections are established between all child-parent pairs.

Figure 6.5 to 6.7 show the UAVs fly onto the designated line after receiving the task. Each UAV occupies a different point of the line. UAVs coordinate with each other to form the whole line.

This simulation case demonstrates that the swarm algorithm and the implementation perform as designed. Both swarm control and task coordination are accomplished successfully.

6.2.2. Letter Task

In this case, 50 UAVs are commanded to form the letters “LOVE”.

Figure 6.8 to 6.11 show the swarm organisation process. Compared to the simple line

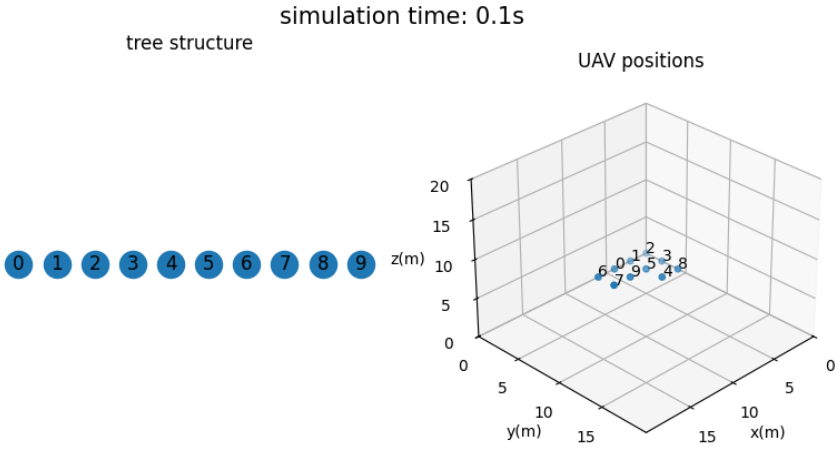

Figure 6.2.: Line task at 0.1s.

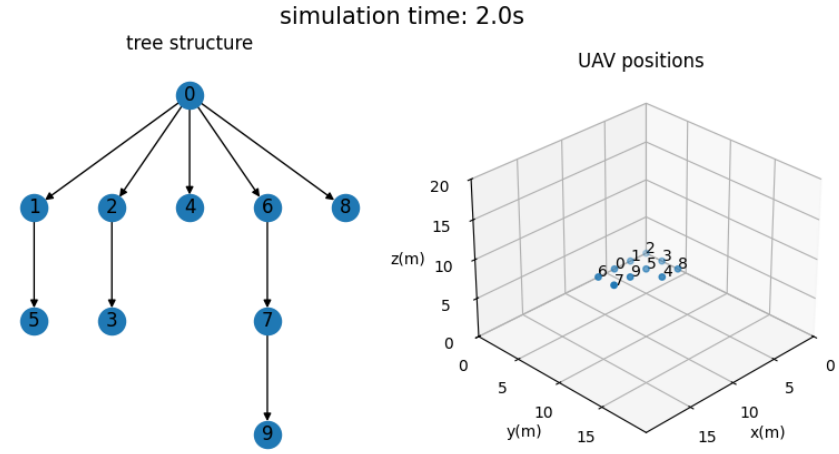

Figure 6.3.: Line task at 2.0s.

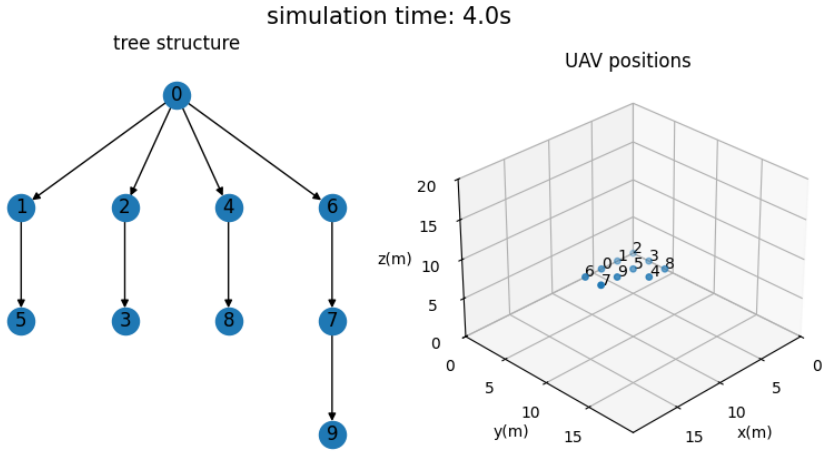

Figure 6.4.: Line task at 4.0s.

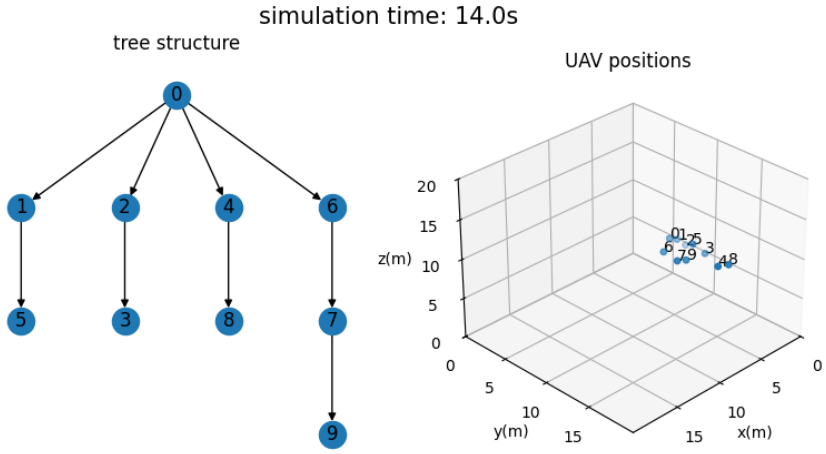

Figure 6.5.: Line task at 14.0s.

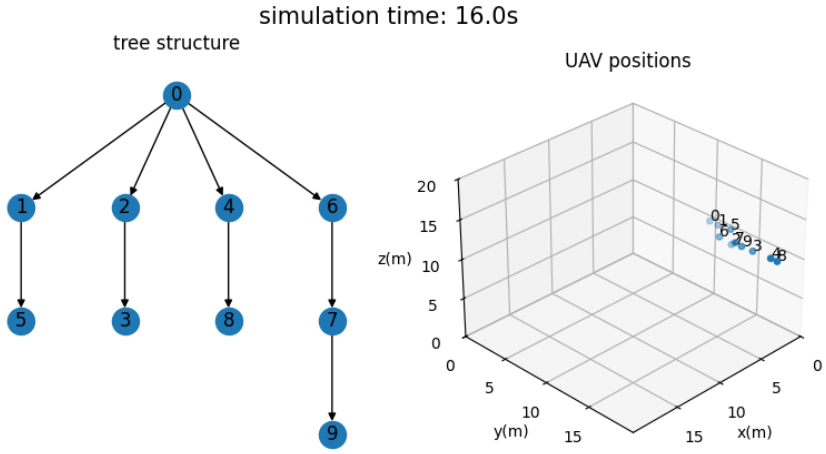

Figure 6.6.: Line task at 16.0s.

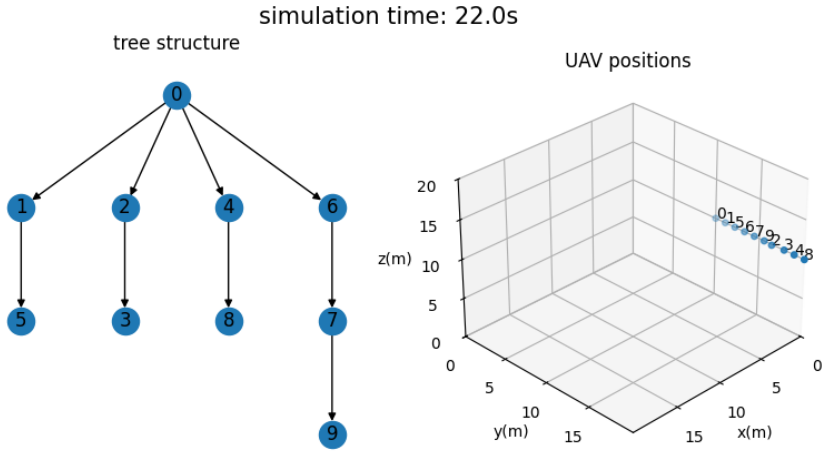

Figure 6.7.: Line task at 22.0s.

task, due to a much large swarm size, it takes the UAVs more time to establish a stable tree structure. The resulted tree is also more complicated.

Figure 6.12 to 6.15 show the task execution process. This letter task is comprised of multiple lines. Moreover, the letter 'O' is a curve. The success of this task proves the effectiveness of the task division algorithm.

This simulation case demonstrates that the swarm algorithm is suitable for a large swarm to handle complex tasks.

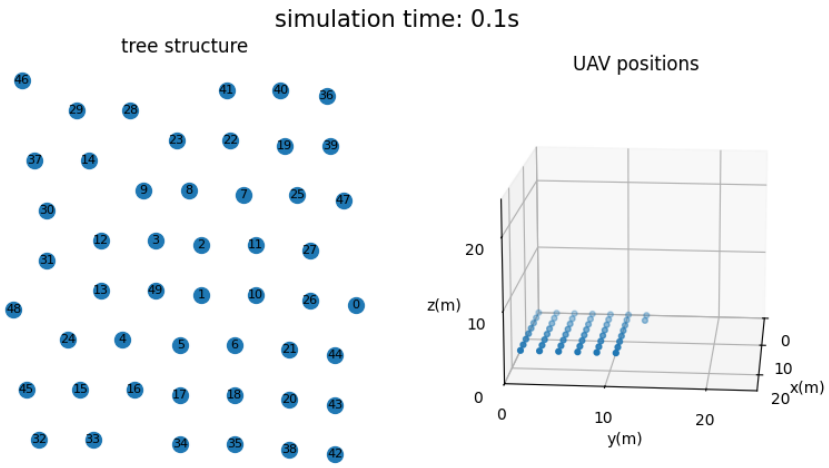

Figure 6.8.: Letter task at 0.1s.

6.3. Evaluation

The simulation results in section 6.2 show that the problem in chapter 3 is successfully solved. The design of the swarm algorithm in chapter 4 is demonstrated, and the implementation in chapter 5 is validated. It can be seen that a group of UAVs can dynamically organise themselves into a tree structure. UAVs in such hierarchical swarm can coordi-

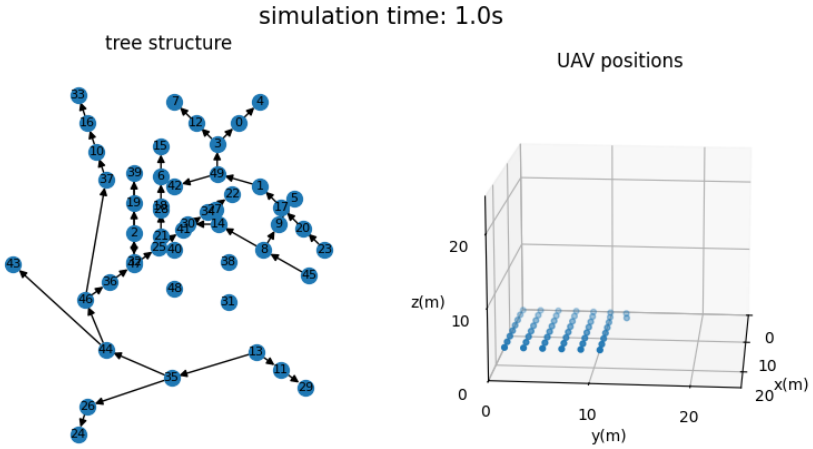

Figure 6.9.: Letter task at 1.0s.

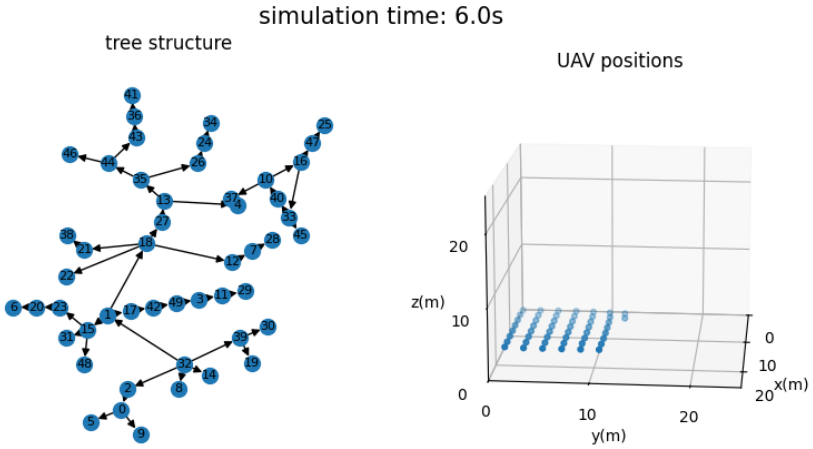

Figure 6.10.: Letter task at 6.0s.

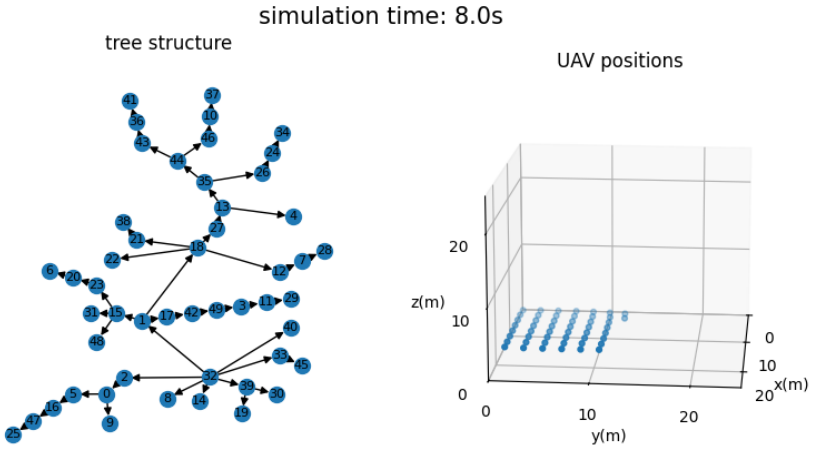

Figure 6.11.: Letter task at 8.0s.

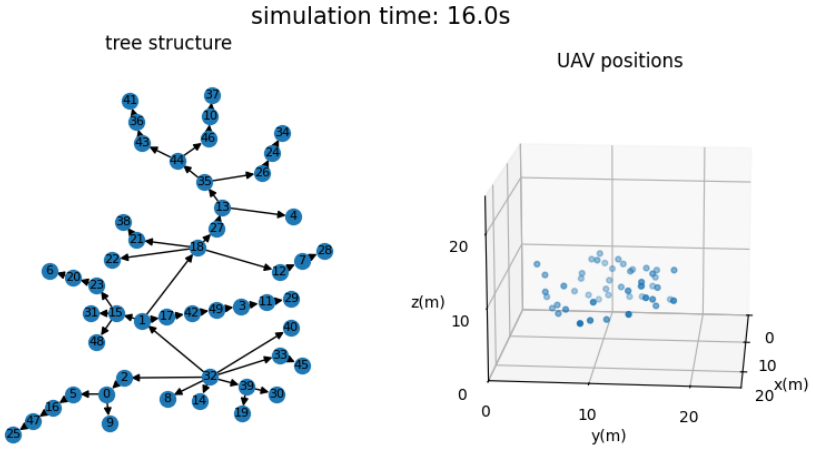

Figure 6.12.: Letter task at 16.0s.

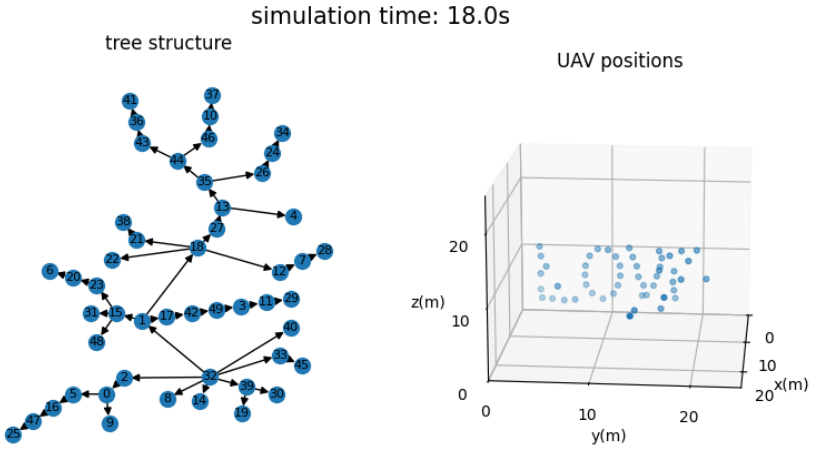

Figure 6.13.: Letter task at 18.0s.

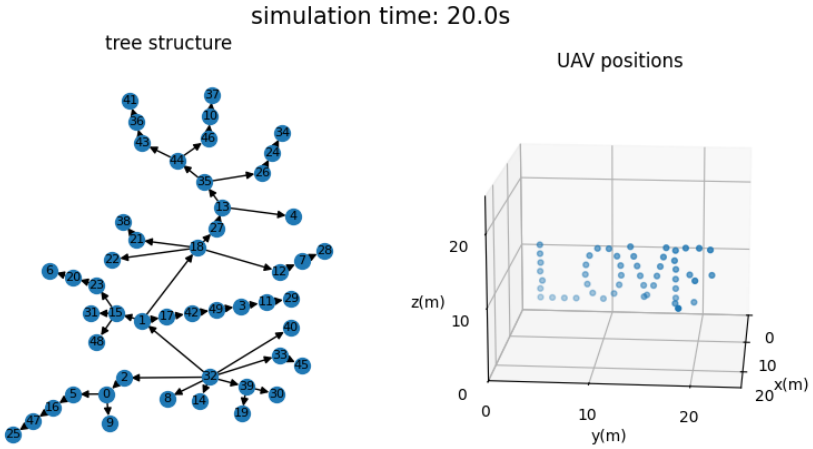

Figure 6.14.: Letter task at 20.0s.

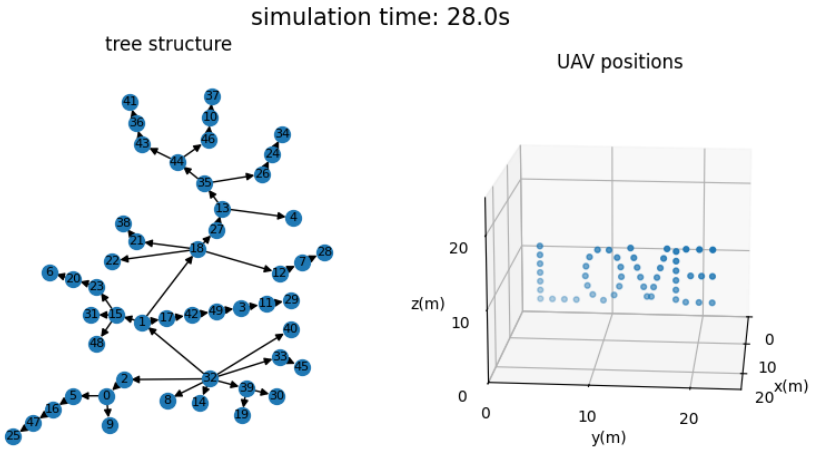

Figure 6.15.: Letter task at 28.0s.

nate with each other to carry out tasks. A task is divided layer by layer down the tree, with each UAV responsible for its own part of the task.

This thesis is inspired by drone shows. The overall goal of this thesis is successfully achieved, and the results indicate the potential usage of autonomous swarms in future drone shows. However, there are still a lot of problems to be addressed.

- Support for very large swarm sizes (>100) needs to be verified. One of the reasons of the hierarchical design is to support extremely large swarm sizes. However, the simulation bed in this thesis involves a lot of inter-process communication, which is time-consuming and limits the maximum simulated swarm size.

- Support for limited communication range needs to be tested. In the simulations in section 6.2, the communication range is large enough so that two arbitrary UAVs can communicate with each other. But in the design, a UAV does not have to be able to communicate with all the other UAVs. Simulation cases are needed where

the communication range does not cover the whole swarm. The shape division algorithm in section 5.3 also needs to be adapted if the communication range does not cover the whole shape.

- The tree structure of a swarm is not optimised. The current design only ensures the swarm is organised into a tree, but does not ensure the tree structure is optimal. To improve the communication and coordination inside the swarm, the tree should be as balanced as possible. It should not be too deep. Nor should parent nodes have too many children. So some kind of optimisation algorithm is needed.

- There is no fault tolerance and fault recovery for task execution. The dynamic tree organisation design solves the fault tolerance problem when the swarm is in free state. But during task execution, in the current implementation, if any single UAV fails, the whole task fails. This compromises the robustness of the swarm.

Apart from the above problems, the following aspects which are ignored in this thesis also restrain the application of the swarm algorithm in any real drone shows.

- Lack of obstacle detection. UAVs are not able to detect obstacles and other UAVs. They acquire the positions of other UAVs through receiving broadcasting messages. Broadcasting frequency must be high to reduce the possibility of collisions. But frequent broadcasting leads to high performance overhead.

- Lack of the ability to deal with complex or animated patterns. In real drone shows, the patterns are more than just simple shapes. Patterns may also transform smoothly from one into another.

- Other problems such as no battery management, no LED lights control, unrealistic flight kinetics, etc.

7. Conclusion and Future Work

7.1. Conclusion

In this thesis, a hierarchical UAV swarm algorithm is proposed and demonstrated. The algorithm enables large autonomous swarms to carry out complex tasks effectively and robustly. The key idea of the algorithm is to dynamically organise a group of UAVs into a tree structure. Tasks are divided and executed layer by layer down the tree. Since messages are mainly sent between each child-parent node pair, there is no need for many-to-many communication, improving the flexibility of the swarm and reducing the communication overhead. Tasks are handled in a divide-and-conquer way, so the computational overhead is reduced compared to fully distributed algorithms. When the depth of the tree grows, the number of nodes grows exponentially, thus large swarm sizes are supported. The tree structure of the swarm is dynamically organised, which means the structure can be repaired if any UAV fails, therefore the structure is robust.

To design and implement the algorithm in detail, an application scenario is conceived in chapter 3, where the task for a swarm is to form designated shapes. Based on the scenario, the swarm algorithm is developed step by step in chapter 4. Then the algorithm is implemented in Rust in chapter 5. A simulation bed is built to carry out simulations in chapter 6.

Two simulation cases are run, the simple line task and the letter task. The results show that the algorithm performs as expected. In both cases, the tree structure is organised

and the designated shapes are formed. Therefore, the feasibility of the algorithm is verified. In the letter task, the swarm comprises of 50 UAVs, showing that the algorithm is suitable for large swarm sizes.

7.2. Future Work

Although the main goal of this thesis is accomplished, there is still a significant need for further improvement.

- Solve the problems listed in section 6.3, so as to make the algorithm more applicable to real drone shows.
- Carry out simulations to compare the developed algorithm with other types of swarm algorithms, and verify the advantages and disadvantages of the developed algorithm.
- Adapt the software, deploy it to real drones, and perform real experiments.

Besides, based on the results of this thesis, there are some interesting directions for future research.

- Complex hierarchical swarm structure. A hierarchical swarm structure does not have to be a simple tree. For example, UAVs can be divided into small groups, and these groups are organised into a tree. Inside each group, UAVs are equal and fully distributed algorithms are run. While between the groups, hierarchical algorithms are run. This tree-of-group structure may be more robust than a simple tree.
- Tree adaption according to task requirements. A swarm will be more powerful if it is able to adjust its tree structure for each specific task. For example, if a node with 10 descendant nodes has two sub-tasks, which require 4 UAVs and 6 UAVs

respectively, then the node can arrange its descendant nodes into two sub-swarms of size 4 and size 6.

- Node management with machine learning methods. In this thesis, a state machine is used to manage node status and task status. However, for more complex tasks and more complex swarm structures, the number of states grows quickly and becomes unmanageable. Machine learning methods may help in these situations.
- Heterogeneous hierarchical swarm. UAVs may be heterogeneous. For instance, they may have different communication ranges. Organisation and coordination of such swarms are much more complicated.

A. Code Overview

The code is organised into two Rust packages, *astro* and *quantity*.

A.1. Package *astro*

astro stands for "Autonomous Swarm Tasking, Routing, and Organisation". It is the main package. The Rust modules contained in this package are listed below.

- *astro*. Defines struct *Astro*, which has member fields of type *Comm*, *Control*, *Gps*, and *Kinetics*. The software main loop mentioned in section 4.3 is implemented in this Rust module.
- *astroconf*. Contains configurations such as the radius of a UAV.
- *comm*. Defines struct *Comm*. This Rust module implements the functional module "Communication" in section 4.3.
- *control*. Defines struct *Control*, which has member fields of type *Contacts*, *NodeManager*, and *ColliVoid*. Implements the functional module "Control" in section 4.3. It has the following submodules.
  - *collivoid*. Defines struct *ColliVoid*. Implements the functional module "Collision Avoidance" in section 4.3.
  - *contacts*. Defines struct *Contacts*. Implements the functional module "Contacts" in section 4.3.

  - *msg*. Defines struct *Msg* and some other related types. Objects of *Msg* type need to be transferred among UAVs as messages.
  - *nm*. Defines struct *NodeManager*. Implements the functional module "Node Management" in section 4.3. Rust enum *NodeState* is defined to represent the node states.
  - *tm*. Defines struct *TaskManager*. Implements the functional module "Task Management" in section 4.3. It also defines structs *TaskDivider* and *TaskExecutor* which are used to divide and execute a task.

- *gps*. Defines struct *Gps*. Implements the functional module "GPS" in section 4.3.
- *kinetics*. Defines struct *Kinetics*. Implements the functional module "Flight Kinetics" in section 4.3. Some kinetic physical quantities are also defined here, e.g., *PosVec* for position vector.
- *transceiver*. As there is no real hardware, *comm*, *gps*, and *kinetics* actually interact with simulation infrastructure. The *transceiver* module is an adapter between the three modules and the simulation infrastructure. It uses socket with non-blocking API as an inter-process communication method.

Serialisation and deserialisation are necessary for any software that transfers internal data through network. In package *astro*, Rust crate *serde* is employed for this purpose. *serde* provides macro *Serialize* and macro *Deserialize*, which enable a type to serialise into and deserialise from a byte sequence. All the structs defined in module *msg* derive *Serialize* and *Deserialize*, since they need to be transferred as byte sequence through network.

The 32-bit unsigned integer type *u*32 is chosen as the type of UAV ID. *Vec*<*u*32> is chosen as the type of NID. Task ID is also of type *u*32.

A.2. Package *quantity*

quantity is an auxiliary package. It defines $VectorF32$, which is a Rust macro. After deriving $VectorF32$, structs which are composed of $f32$ fields can participate in some linear algebraic manipulations. The struct $PosVec$ for position vector, and the struct $Velocity$ for velocity vector, which are both 3D kinematic quantities defined in ***kinetics*** module, derive this macro.

Bibliography

[1] G. Antonelli, F. Arrichiello, and S. Chiaverini, "The null-space-based behavioral control for mobile robots," in *2005 International Symposium on Computational Intelligence in Robotics and Automation*, IEEE, 2005, pp. 15–20, ISBN: 0-7803-9355-4. DOI: 10.1109/CIRA.2005.1554248 (cit. on p. 7).

[2] F. Arrichiello, S. Chiaverini, G. Indiveri, and P. Pedone, "The null-space-based behavioral control for mobile robots with velocity actuator saturations," *The International Journal of Robotics Research*, vol. 29, pp. 1317–1337, 10 2010, ISSN: 0278-3649. DOI: 10.1177/0278364909358788 (cit. on p. 7).

[3] G. Beni and J. Wang, "Swarm intelligence in cellular robotic systems," in Springer Berlin Heidelberg, 1993, pp. 703–712. DOI: 10.1007/978-3-642-58069-7_38 (cit. on p. 10).

[4] B. Bishop and D. Stilwell, "On the application of redundant manipulator techniques to the control of platoons of autonomous vehicles," in *Proceedings of the 2001 IEEE International Conference on Control Applications (CCA'01) (Cat. No.01CH37204)*, IEEE, 2001, pp. 823–828, ISBN: 0-7803-6733-2. DOI: 10.1109/CCA.2001.973971 (cit. on p. 7).

[5] M. R. Bonyadi and Z. Michalewicz, "Particle swarm optimization for single objective continuous space problems: A review," *Evolutionary Computation*, vol. 25, pp. 1–54, 1 2017, ISSN: 1063-6560. DOI: 10.1162/EVCO_r_00180 (cit. on p. 10).

[6] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: A review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, pp. 1–41, 1 2013, ISSN: 1935-3812. DOI: 10.1007/s11721-012-0075-2 (cit. on p. 4).

[7] British Army. "British army carries out successful swarming drone capability." (2022), [Online]. Available: https://www.army.mod.uk/news-and-events/news/2022/09/british-army-carries-out-successful-swarming-drone-capability (cit. on p. 11).

[8] M. R. Brust and B. M. Strimbu, "A networked swarm model for uav deployment in the assessment of forest environments," in *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, IEEE, 2015, pp. 1–6, ISBN: 978-1-4799-8055-0. DOI: 10.1109/ISSNIP.2015.7106967 (cit. on pp. 4, 13).

[9] M. Campion, P. Ranganathan, and S. Faruque, "Uav swarm communication and control architectures: A review," *Journal of Unmanned Vehicle Systems*, vol. 7, pp. 93–106, 2 2019, ISSN: 2291-3467. DOI: 10.1139/juvs-2018-0009 (cit. on p. 4).

[10] P. Cao, L. Lei, S. Cai, G. Shen, X. Liu, X. Wang, L. Zhang, L. Zhou, and M. Guizani, "Computational intelligence algorithms for uav swarm networking and collaboration: A comprehensive survey and future directions," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2024, ISSN: 1553-877X. DOI: 10.1109/COMST.2024.3395358 (cit. on pp. 5, 10).

[11] H. CHEN, X. WANG, L. SHEN, and Y. CONG, "Formation flight of fixed-wing uav swarms: A group-based hierarchical approach," *Chinese Journal of Aeronautics*, vol. 34, pp. 504–515, 2 2021, ISSN: 10009361. DOI: 10.1016/j.cja.2020.03.006 (cit. on p. 13).

[12] X. Chen, J. Tang, and S. Lao, "Review of unmanned aerial vehicle swarm communication architectures and routing protocols," *Applied Sciences*, vol. 10, p. 3661, 10 2020, ISSN: 2076-3417. DOI: 10.3390/app10103661 (cit. on p. 4).

[13] Z. Chen, J. Yan, B. Ma, K. Shi, Q. Yu, and W. Yuan, "A survey on open-source simulation platforms for multi-copter uav swarms," *Robotics*, vol. 12, p. 53, 2 2023, ISSN: 2218-6581. DOI: 10.3390/robotics12020053 (cit. on p. 47).

[14] A. Dimakos, D. Woodhall, and S. Asif, "A study on centralised and decentralised swarm robotics architecture for part delivery system," 2024. DOI: 10.31031/AES.2021.2.000540 (cit. on pp. 1, 3, 5).

[15] DJI. "Support for dji mini 2." (2024), [Online]. Available: https://www.dji.com/uk/support/product/mini-2 (cit. on p. 49).

[16] H. Duan, Q. Luo, Y. Shi, and G. Ma, "?hybrid particle swarm optimization and genetic algorithm for multi-uav formation reconfiguration," *IEEE Computational Intelligence Magazine*, vol. 8, pp. 16–27, 3 2013, ISSN: 1556-603X. DOI: 10.1109/MCI.2013.2264577 (cit. on p. 10).

[17] R. Falconi and C. Melchiorri, "A decentralized control algorithm for swarm behavior and obstacle avoidance in unknown environments," *IFAC Proceedings Volumes*, vol. 41, pp. 44–49, 1 2008, ISSN: 14746670. DOI: 10.3182/20080408-3-IE-4914.00009 (cit. on p. 7).

[18] R. Ganesan, X. M. Raajini, A. Nayyar, P. Sanjeevikumar, E. Hossain, and A. H. Ertas, "Bold: Bio-inspired optimized leader election for multiple drones," *Sensors*, vol. 20, p. 3134, 11 2020, ISSN: 1424-8220. DOI: 10.3390/s20113134 (cit. on p. 13).

[19] V. Grishchenko, M. Patrakeev, and S. Q. Locke, "Swarm consensus," 2021. [Online]. Available: http://arxiv.org/abs/2112.07065 (cit. on p. 9).

[20] S. Javaid, N. Saeed, Z. Qadir, H. Fahim, B. He, H. Song, and M. Bilal, "Communication and control in collaborative uavs: Recent advances and future trends," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, pp. 5719–5739, 6 2023, ISSN: 1524-9050. DOI: 10.1109/TITS.2023.3248841 (cit. on pp. 1, 3, 4).

[21] S. Javed, A. Hassan, R. Ahmad, W. Ahmed, R. Ahmed, A. Saadat, and M. Guizani, "State-of-the-art and future research challenges in uav swarms," *IEEE Internet of Things Journal*, vol. 11, pp. 19 023–19 045, 11 2024, ISSN: 2327-4662. DOI: 10.1109/JIOT.2024.3364230 (cit. on pp. 1, 4, 5, 8, 10).

[22] M. A. Kamel, X. Yu, and Y. Zhang, "Formation control and coordination of multiple unmanned ground vehicles in normal and faulty situations: A review," *Annual Reviews in Control*, vol. 49, pp. 128–144, 2020, ISSN: 13675788. DOI: 10.1016/j.arcontrol.2020.02.001 (cit. on pp. 4–6, 8).

[23] K.-S. Kim, H.-Y. Kim, and H.-L. Choi, "A bid-based grouping method for communication-efficient decentralized multi-uav task allocation," *International Journal of Aeronautical and Space Sciences*, vol. 21, pp. 290–302, 1 2020, ISSN: 2093-274X. DOI: 10.1007/s42405-019-00205-1 (cit. on p. 9).

[24] M.-T. Lee, S.-T. Kuo, Y.-R. Chen, and M.-L. Chuang, "Uav swarm real-time rerouting by edge computing under a changing environment," in *2021 IEEE 3rd Eurasia Conference on IOT, Communication and Engineering (ECICE)*, IEEE, 2021, pp. 6–9, ISBN: 978-1-6654-4516-0. DOI: 10.1109/ECICE52819.2021.9645660 (cit. on p. 10).

[25] Q. Li, H. Xiong, Y. Ding, J. Song, J. Liu, and Y. Chen, "A review of unmanned aerial vehicle swarm task assignment," in *Advances in Guidance, Navigation and Control*, Springer Nature Singapore, 2023, pp. 6469–6479, ISBN: 978-981-19-6613-2. DOI: 10.1007/978-981-19-6613-2_624 (cit. on pp. 5, 42).

[26] Y. Li and C. Tan, "A survey of the consensus for multi-agent systems," *Systems Science & Control Engineering*, vol. 7, pp. 468–482, 1 2019, ISSN: 2164-2583. DOI: 10.1080/21642583.2019.1695689 (cit. on p. 9).

[27] Y. Lu, Y. Ma, J. Wang, and L. Han, "Task assignment of uav swarm based on wolf pack algorithm," *Applied Sciences*, vol. 10, p. 8335, 23 2020, ISSN: 2076-3417. DOI: 10.3390/app10238335 (cit. on p. 10).

[28] L. Ma, B. Lin, W. Zhang, J. Tao, X. Zhu, and H. Chen, "A survey of research on the distributed cooperation method of the uav swarm based on swarm intelligence," in *2022 IEEE 13th International Conference on Software Engineering and Service Science (ICSESS)*, IEEE, 2022, pp. 305–309, ISBN: 978-1-6654-1031-1. DOI: 10.1109/ICSESS54813.2022.9930182 (cit. on pp. 4, 5, 8).

[29] M. MAMEI, M. VASIRANI, and F. ZAMBONELLI, "Experiments of morphogenesis in swarms of simple mobile robots," *Applied Artificial Intelligence*, vol. 18, pp. 903–919, 9-10 2004, ISSN: 0883-9514. DOI: 10.1080/08839510490509081 (cit. on p. 8).

[30] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX annual technical conference (USENIX ATC 14)*, 2014, pp. 305–319. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro (cit. on p. 10).

[31] Q. Ouyang, Z. Wu, Y. Cong, and Z. Wang, "Formation control of unmanned aerial vehicle swarms: A comprehensive review," *Asian Journal of Control*, vol. 25, pp. 570–593, 1 2023, ISSN: 1561-8625. DOI: 10.1002/asjc.2806 (cit. on pp. 1, 3–6, 8).

[32] P. Pasek and P. Kaniewski, "A review of consensus algorithms used in distributed state estimation for uav swarms," in *2022 IEEE 16th International Conference*

on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), IEEE, 2022, pp. 472–477, ISBN: 978-1-6654-6861-9. DOI: 10.1109/TCSET55632.2022.9766903 (cit. on p. 9).

[33] Q. Peng, H. Wu, and R. Xue, "Review of dynamic task allocation methods for uav swarms oriented to ground targets," *Complex System Modeling and Simulation*, vol. 1, pp. 163–175, 3 2021, ISSN: 2096-9929. DOI: 10.23919/CSMS.2021.0022 (cit. on pp. 5, 9, 10, 42).

[34] S. J. Plathottam and P. Ranganathan, "Next generation distributed and networked autonomous vehicles: Review," in *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, IEEE, 2018, pp. 577–582, ISBN: 978-1-5386-1182-1. DOI: 10.1109/COMSNETS.2018.8328277 (cit. on p. 9).

[35] A. Puente-Castro, D. Rivero, A. Pazos, and E. Fernandez-Blanco, "A review of artificial intelligence applied to path planning in uav swarms," *Neural Computing and Applications*, vol. 34, pp. 153–170, 1 2022, ISSN: 0941-0643. DOI: 10.1007/s00521-021-06569-4 (cit. on pp. 5, 8).

[36] S. Ranganathan, M. Mariappan, and K. Muthukaruppan, "Efficient distributed consensus algorithm for swarm robotic," in *2022 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, IEEE, 2022, pp. 1–6, ISBN: 978-1-6654-6837-4. DOI: 10.1109/IICAIET55139.2022.9936787 (cit. on p. 9).

[37] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH Computer Graphics*, vol. 21, pp. 25–34, 4 1987, ISSN: 0097-8930. DOI: 10.1145/37402.37406 (cit. on p. 8).

[38] V. Roberge, M. Tarbouchi, and G. Labonte, "Comparison of parallel genetic algorithm and particle swarm optimization for real-time uav path planning," *IEEE*

Transactions on Industrial Informatics, vol. 9, pp. 132–141, 1 2013, ISSN: 1551-3203. DOI: 10.1109/TII.2012.2198665 (cit. on p. 10).

[39] Rust Team. "Rust programming language." (no date), [Online]. Available: https://www.rust-lang.org/ (cit. on p. 39).

[40] F. Schneider and D. Wildermuth, "A potential field based approach to multi robot formation navigation," in *IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, 2003. Proceedings. 2003*, vol. 1, IEEE, 2003, pp. 680–685, ISBN: 0-7803-7925-X. DOI: 10.1109/RISSP.2003.1285656 (cit. on p. 8).

[41] M. M. Shahzad, M. H. Asad, M. Haris, H. Munawar, and M. H. Yousaf, "Formation control and sub-swarm generation of multirotor uavs," in *2023 International Conference on Robotics and Automation in Industry (ICRAI)*, IEEE, 2023, pp. 1–7, ISBN: 978-1-6654-6472-7. DOI: 10.1109/ICRAI57502.2023.10089546 (cit. on pp. 4, 8).

[42] P. Sujit and R. Beard, "Distributed sequential auctions for multiple uav task allocation," in *2007 American Control Conference*, IEEE, 2007, pp. 3955–3960, ISBN: 1-4244-0988-8. DOI: 10.1109/ACC.2007.4282558 (cit. on p. 9).

[43] A. Tahir, J. M. Boling, M.-H. Haghbayan, and J. Plosila, "Comparison of linear and nonlinear methods for distributed control of a hierarchical formation of uavs," *IEEE Access*, vol. 8, pp. 95 667–95 680, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2988773 (cit. on p. 13).

[44] Y. Tan and Z.-y. Zheng, "Research advance in swarm robotics," *Defence Technology*, vol. 9, pp. 18–39, 1 2013, ISSN: 22149147. DOI: 10.1016/j.dt.2013.03.001 (cit. on p. 10).

[45] J. Tang, H. Duan, and S. Lao, "Swarm intelligence algorithms for multiple unmanned aerial vehicles collaboration: A comprehensive review," *Artificial Intelligence Review*, vol. 56, pp. 4295–4327, 5 2023, ISSN: 0269-2821. DOI: 10.1007/s10462-022-10281-7 (cit. on p. 10).

[46] N. Thorjussen and T. Samaritano. "Everything you ever wanted to know about drone light shows." (no date), [Online]. Available: https://www.verge.aero/everything-about-drone-light-shows (cit. on p. 14).

[47] U.S. Department of Defense. "Department of defense announces successful micro-drone demonstration." (2017), [Online]. Available: https://www.defense.gov/News/Releases/Release/Article/1044811/department-of-defense-announces-successful-micro-drone-demonstration/ (cit. on p. 10).

[48] U.S. Government Accountability Office. "Science & tech spotlight: Drone swarm technologies." (2023), [Online]. Available: https://www.gao.gov/products/gao-23-106930 (cit. on p. 3).

[49] Y. Wan, J. Tang, Z. Zhao, X. Chen, and J. Zhan, "Systematic review of formation control for multiple unmanned aerial vehicles," in *2023 9th International Conference on Big Data and Information Analytics (BigDIA)*, IEEE, 2023, pp. 169–176. DOI: 10.1109/BigDIA60676.2023.10429313 (cit. on pp. 4–6).

[50] J. Wang, X.-b. Wu, and Z.-l. Xu, "Decentralized formation control and obstacles avoidance based on potential field method," in *2006 International Conference on Machine Learning and Cybernetics*, IEEE, 2006, pp. 803–808, ISBN: 1-4244-0061-9. DOI: 10.1109/ICMLC.2006.258457 (cit. on p. 8).

[51] R. Wang, J. Du, Z. Xiong, X. Chen, and J. Liu, "Hierarchical collaborative navigation method for uav swarm," *Journal of Aerospace Engineering*, vol. 34, 1 2021, ISSN: 0893-1321. DOI: 10.1061/(ASCE)AS.1943-5525.0001216 (cit. on p. 13).

[52] S. Xu, L. Li, Z. Zhou, Y. Mao, and J. Huang, "A task allocation strategy of the uav swarm based on multi-discrete wolf pack algorithm," *Applied Sciences*, vol. 12, p. 1331, 3 2022, ISSN: 2076-3417. DOI: 10.3390/app12031331 (cit. on p. 10).

[53] Y. Zhou, B. Rao, and W. Wang, "Uav swarm intelligence: Recent advances and future trends," *IEEE Access*, vol. 8, pp. 183 856–183 878, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3028865 (cit. on pp. 1, 3, 4, 8, 10).