UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Universitat Politècnica de València

Departamento de Informática de Sistemas y Computadores

A solution for the efficient takeoff and flight coordination of UAV swarms

Master's thesis
Master's Degree in Computer and Network Engineering

Author
Jamie Wubben

Advisors
Dr. Carlos Tavares Calafate
Dr. Juan-Carlos Cano

July 2021

Acknowledgements

First of all, I would like to thank both of my advisors, Carlos Tavares Calafate and Juan-Carlos Cano, for their support and excellent guidance. Thanks to them, I am able to achieve this goal, and many more in the future. In the same way, I would like to express my gratitude for the entire team from the Computer Networks Research Group (GRC). They, too, have contributed with their knowledge, but most of all by creating a wholesome workplace where I am working for many hours, but never with reluctance. Finally, to my parents who, although with difficulties, allowed me to continue my studies abroad and provide continuous support.

Jamie Wubben
Valencia, July 4, 2021

Abstract

In the last decade, we have seen a great increase in the use of Unmanned Aerial Vehicles (UAVs). This is mainly due to advances in technology and materials. Nowadays, UAVs are no longer only toys for entertainment, but also important assets for many enterprises. UAVs are versatile, and thus many diverse applications exist: search and rescue missions, border surveillance, thermal pipeline inspection, cinematography, and precision agriculture, just to name a few. Now that the industry is incorporating UAVs based solutions, it is crucial that research advances. The most prominent change (with respect to UAVs) that we will witness in this decade, is the deployment of groups of UAVs working collaboratively to fulfill a higher goal. Those groups, also called swarms, allow us to perform more complex tasks, more efficiently, or with more redundancy. However, there are inherent challenges while operating a swarm of UAVs: there must be a good communication channel between the UAVs, collisions must be avoided, and the individual UAVs should be used intelligently in order to increase the overall efficiency.

In this master thesis, a solution is given for some of the main problems concerning Unmanned Aerial Vehicle (UAV) swarms. First, we lay out various useful swarm formation patterns. Then we incorporate those formations in two takeoff procedures - an heuristic and an existing algorithm (Kuhn-Munkres algorithm (KMA)) - which are extensively tested to decide which one is the most appropriate for the takeoff of a swarm of UAVs in the most efficient manner. Once we are able to take off an entire swarm, we continue our research by providing a solution to keep that swarm organized and stable during a pre-planned mission. Such solution incorporates mechanisms to provide resilience to the swarm in such a manner that any number of UAVs can be removed from the swarm (mid-flight) without disturbing the others in their mission.

Resumen

En la última década, hemos asistido a un gran aumento del uso de los VANTs, debido principalmente a los avances en tecnología y materiales. Hoy en día, los VANTs ya no son sólo juguetes para el entretenimiento, sino también importantes activos para muchas empresas. Los VANTs son muy versátiles y, por ello, existen muchas y variadas aplicaciones: misiones de búsqueda y rescate, vigilancia de fronteras, inspección térmica de tuberías, cinematografía y agricultura de precisión, sólo por nombrar algunas. En estos momentos en que las industrias están incorporando soluciones basadas en VANTs, es crucial que la investigación avance. El cambio más destacado (con respecto a los VANTs) que presenciaremos en esta década, es el despliegue de grupos de VANTs trabajando en colaboración para cumplir un objetivo superior. Estos grupos, también llamados enjambres de drones, permiten realizar tareas más complejas, de forma más eficiente, o con mayor redundancia. Sin embargo, existen retos inherentes al funcionamiento de un enjambre de VANTs. Debe existir una buena comunicación entre los VANTs, deben evitarse las colisiones y los VANTs individuales deben utilizarse de forma inteligente para aumentar la eficiencia global.

En este trabajo fin de máster se da solución a algunos de los principales problemas relativos a los enjambres de vehículos aéreos no tripulados. En primer lugar, diseñamos varios patrones de formación de enjambres útiles. A continuación, incorporamos esas formaciones en dos procedimientos de despegue - una heurística y un algoritmo ya existente (KMA) - los cuales se prueban ampliamente para decidir cuál es el más adecuado para despegar un enjambre de VANTs de la manera más eficiente. Una vez que somos capaces de despegar de forma sincronizada y segura un enjambre completo, continuamos nuestra investigación proporcionando una solución para mantener ese enjambre organizado, y estable durante una misión pre-planificada. Nuestra solución incorpora mecanismos para proporcionar resiliencia al enjambre, de tal manera que todos y cada uno de los VANTs pueden abandonar el enjambre (en pleno vuelo), sin perturbar a los demás en su misión.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

UAVs, colloquially known as drones, are used more and more by the general public. Popular applications are: aerial photography and video, topography, entertainment, etc. [1]. Furthermore, UAVs are increasingly used for specialized, business-oriented applications such as precision agriculture, border surveillance, parcel delivery, thermal inspections, people monitoring under the COVID-19 pandemic, and many more [2, 3]. Nowadays, UAVs are also starting to be used to assist in emergency situations such as search and rescue, or disaster scenarios [4, 5], where they can act as supporting nodes for communications being deployed on demand, and offering a wider communications range and better line-of-sight (LOS) features than ground infrastructures. This is all to say that UAV applications are versatile and can be used to solve real problems (instead of just using them for entertainment).

However, in all the examples above only one UAV is used and managed. Since the battery lifetime and the lift power of an UAV are limited, interest is growing towards solutions that use multiple UAVs at the same time. This combination of multiple UAVs working collaboratively is also known as a UAV swarm. Such a swarm can provide many benefits to existing solutions. In general, tasks can be performed more efficiently and/or with more redundancy. It also gives rise to new applications such as carrying heavy loads. However, deploying a swarm of UAVs is not an easy task, and many challenges remain unsolved. Such challenges include: (i) swarm formation definition, (ii) takeoff assignment and procedure, (iii) in-flight coordination, (iv) handling the loss of swarm elements, (v) swarm layout reconfiguration, (vi) communications and data relaying optimization, and (vii) controlled landing.

1.2 Objectives

The main objective of this thesis is to design, implement, and verify solutions for the first four challenges listed above, i.e. (i) swarm formation definition, (ii) takeoff assignment and procedure, (iii) in-flight coordination, (iv) handling the loss of swarm elements. To achieve this, ArduSim [6](a flight simulation platform for UAVs) will be used. In order to achieve those global goals, it becomes necessary to accomplish several specific objectives.

Swarm formation definition:

- Designing and implementing widely used formation patterns such as: a linear, a circle, and a matrix formation.

Takeoff assignment:

- Developing an efficient heuristic to assign UAVs on the ground a position in the air based on the desired formation (the assignment problem).
- Implementing the KMA to solve (the same) assignment problem.
- Comparing the two different algorithms in terms of: total distance travelled, number of possible collisions, etc.
- Designing a comprehensible flowchart to decide which algorithm should be used depending on the swarm configuration.

Resilient in-flight coordination:

- Developing an algorithm that coordinates the UAV swarm, while they are following a pre-planned mission, so that the swarm stays coherent.
- Extending that algorithm so that, during the flight, any (and multiple) UAVs can be removed without disrupting the rest of the swarm in their mission.
- Testing that algorithm in various scenarios.

1.3 Structure of the Thesis

This Master thesis is organized in 8 chapters. Below, we briefly describe the contents of each part:

- **Chapter 2. Related works.** we provide an overview of various state-of-the-art publications related to UAVs and UAV swarms.

- **Chapter 3. Unmanned aerial vehicles: an overview.** We provide an overview of the different types of UAVs that exist, and the general aspects related to the UAVs' architecture.

- **Chapter 4. ArduSim: a multi-UAV simulation platform.** We detail the inner workings of our own open-source multi-UAV flight simulator. We also detail how to deploy the simulated protocols on real UAVs for real experiments.

- **Chapter 5. Swarm formations.** We design, and implement various swarm formations that will be used for the other protocols.

- **Chapter 6. Efficient Takeoff for a swarm of UAVs.** We propose two algorithms to solve the assignment problem: a heuristic, and the KMA. Both algorithms are then implemented and verified; during the experiments we measure the effectiveness of the algorithms in terms of: calculation time, total distance travelled, number of possible collisions, and the total takeoff time.

- **Chapter 7. MUSCOP: a resilient coordination protocol.** We present an existing protocol that is able to maintain a stable flight formation for a UAV swarm that is following a global planned mission. This protocol is improved by additional mechanisms that allow for the failure of any number of UAVs. The protocol is verified by a series of experiments, where we test different scenarios, and measure the time overhead introduced by our protocol.

- **Chapter 8. Conclusions, publications and future work.** We conclude this master thesis, present the related publications, and include some ideas for future work.

Chapter 2

Related works

2.1 UAV swarm protocols

As stated before, more and more people are getting interested in UAV-based applications. A great survey on the core characteristics of drone swarms was published by Tahir et al. [7]. Together with their survey they also created a questionnaire (with an academic audience) to review the public awareness of the swarm technology. In total, they asked 187 participants questions, about application areas, flying mechanisms, autonomous swarms, etc. Their results show that there is moderate knowledge about a drone and a swarm of drones. Despite many participants acknowledged the benefits of using UAVs, not many would rely on drones in his/her activities/business. This is probably because over 60% of the participants have security concerns.

A survey about those security and privacy issues is given in [8]. They analyse various threads from sensors and communication media in the context of UAVs. Since nearly all UAVs are equipped with a camera, the photos taken by a UAV can invade privacy in two ways. First, the pilot can fly over residential areas and (illegally) take pictures; on the other hand, even when the pictures are taken legally, they are usually stored as a JPEG file. JPEG files contain metadata, which often include the shooting time and location. This data can be misused if these pictures are uploaded to social media platforms (as often occurs). Furthermore, UAVs can also be hacked; most of the UAVs depend heavily on the GPS signal, which is vulnerable to jamming and spoofing attacks. Nowadays, drones are often controlled by a smartphone via Wi-Fi instead of a remote controller. This communication can also be hacked and, if successful, the hacker can take control over the UAV.

The work of Mirzaeinia et al. [9] addressed the problem of the increasing

number of drones in smart cities. This makes it more difficult to find the optimal station for each drone after it has completed its mission. When a drone finishes its mission, it is assigned a destination landing station to be recharged. Adverse weather situations can result in having the drone assigned to a station to take shelter. Their proposal uses the KMA to match drone charging stations. In their study, three different scenarios were investigated. The first scenario used drones with the same energy level. The second scenario, used drones with a different energy level, and finally the third scenario where drones and stations had different energy levels. The results showed that a reduction in energy consumption of 30-90% can be achieved by applying this algorithm in drone station pairing compared to an algorithm using a randomly preassigned station.

Pestana et al. [10] presented a modular multi-robot swarm architecture. Their approach was implemented in the Robot Operating System (ROS) software framework. Since ROS heavily relies on reusable modules, their solution is available for a great audience. They also used the readily available AR Drone 2.0. In their approach, the only information shared among swarm agents is the position of each robot, and they rely on a visual-based solution for localization based on ArUco markers, which are used to sense and map obstacles. In addition, they rely on an Extended Kalman Filter localization and mapping method. However, their approach heavily depends on the Wi-Fi links between the UAV and the laptop, which caused some problems according to the authors.

Cho and Kim [11] attempted to minimize consumption in the deployment of low power drones, arguing that it is an NP-hard problem in the context of air-to-ground models in a three-dimensional area. To approach this problem, they studied the established solutions based on heuristic methods, comparing different heuristic algorithms for optimal UAV deployment. In particular, they compared the performance of the heuristic algorithms for each scenario based on the number of UAVs, and presented the speed of convergence to achieve near optimal solutions.

2.2 The takeoff problem

Above we have provided few examples of works that consider UAV swarms in general; now we proceed to present publications that specifically consider a safe takeoff. Unfortunately, only a few works address this topic, of which many are our own. Three simple takeoff options were explored by N. Dousse et al. [12]: manual, sequential, and simultaneous. However, especially when the swarm becomes larger, these simple takeoff procedures are not feasible

as they would take too much time (manual, sequential) or be prone to cause collisions between the UAVs (simultaneous).

For the above-mentioned reasons, there are only a few examples of large-scale UAV swarms, and most of them were performed by private companies. Intel was a pioneer in this area by being the first to create a UAV-based light show. This light show was held in 2015 in Germany, and 100 drones were used[1]. Since then some other companies such as EHANG[2] and Bilibili[3] have done the same. The records for the most unmanned aerial vehicles in simultaneous flight are changing rapidly, and include already more than 3000 UAVs. It must be said that the light shows are just marketing stunts, and do not add true value. Nevertheless, they are highly entertaining, and show the world what UAV swarms are capable of.

2.3 Swarm resilience

Lastly, we present works that consider swarm resilience. When operating a swarm of UAVs there are basically two approaches: flocking and master-slave models. Flocking is a behaviour that is common in nature, for instance in a group of fish, birds or insects. It consists of a few basic rules that are applied to each entity of the group. When those rules are respected, the group will stay united without collisions between the group elements. There are various methods to achieve a flocking behaviour for a group of UAVs, as discussed below.

Victor Casas et al. [13] developed a flocking model without the use of a master-slave model. The UAVs in the swarm regularly broadcast and receive movement information. That information is then used to calculate two forces: a *flock goal force*, which guides the flock towards the target location and aligns the swarm members, and a *flock members force*, which provides cohesion and separation to the flock. Those two forces are used to update a direction vector which points towards the target location, while at the same time avoids collisions. Their model is tested in simulation and in real experiments, which show that a collision-free flight is ensured. They tested the model under various speeds, although all of them were rather slow (a maximum of 3 m/s). Results also showed that, during real experiments, the minimum distance between UAVs is decreased; according to the authors, this is due to GPS inaccuracy.

[1]https://www.intel.com/content/www/us/en/technology-innovation/article/coachella-drone-light-show.html

[2]https://www.popsci.com/china-drone-swarms/

[3]https://dronedj.com/2021/04/21/drone-qr-code-marketing/

Yazhe Tang et al. [14] presented a swarm flocking scheme that was able to work in a radio silent environment. In contrast to many other works, their approach was not based on sending (GPS) information between the swarm elements. They used two types of vision sensors (standard and thermal cameras) to track their leader, and a LiDAR sensor to sense the surrounding environment for navigation and obstacle avoidance. Because they used various high-end sensors, their flocking mechanism can be used both during the day and during the night. Indoor and outdoor experiments performed in obstacle-rich environments have proven the effectiveness of the proposed method. Furthermore, their software is implemented in the robot operating system (ROS) [15], which promotes reusability through its modular design.

While flocking mechanisms are great to keep a swarm of UAVs organized, they do not provide the flexibility to completely define and change the formation itself. In many applications, it is useful to change the formation (for instance, from a line to a circle); however, it is difficult to encapsulate such behaviour using flocking mechanisms. Therefore, in our work, we specifically focus on changing between different flight formations. Hence, instead of using a flocking mechanism, we use a master-slave model where the master instructs the slaves how to safely accomplish the reconfiguration.

Other works address swarm resilience with other solutions. For instance, Mulgaonkar et al. [16] tested the performance of micro quadcopter swarms in tight/dense formations. By adding a protective case around the drone, the micro UAVs could withstand collisions at velocities of 4 m/s. We believe that micro UAVs are certainly useful. However, due to their low mass, they cannot be used effectively in an outdoor environment. This makes micro UAVs only useful for indoor applications, which is not in the scope of this thesis.

Finally, Chen et al. published a paper [17] that is focused on effectively reorganizing the surviving UAVs in a severely damaged UAV swarm. They start by analysing the damage-resilience problem of unified UAV Swarm Networks (USNETs). The goal of their work was to design a damage-resilience mechanism, which is usually divided into multiple disjoint subnets of isolated nodes. Three challenges are investigated: first, the network will be divided into several disjoint subnets or isolated nodes; secondly, they work on restoring the network connectivity; finally, they explain how to reduce the computational and communication overhead.

Chapter 3

Unmanned aerial vehicles: an overview

The topic of UAVs is broad an interesting; one could go into depth about the flight dynamics[18], propeller types[19], the electronics of the Electronic Speed Controller (ESC)[20], motor types, legislation[21], and so on. In this chapter we provide an overview; to put into context the drones we are targeting, detailing the advantages and disadvantages they have with respect to other types of drone. After that, an overview of the main parts and their function is given.

3.1 Types of UAVs

Nowadays, we can find a large amount of UAVs of various types. Colloquially they are often referred to as drones. However, this is a common misconception; as shown in Figure 3.1, the concept of drones is broader. A drone can be any type of unmanned robot, and thus it also includes robots that operate on land (Unmanned Ground Vehicles (UGV)) or on water (Unmanned Surface Vehicles (USV) and Unmanned Underwater Vehicles (UUV)). As mentioned throughout this document, autonomous robots operating in the air are called UAVs. The UAVs can be further subdivided into fixed wing and copters. The copters have the unique ability of taking off vertically, and so in the literature the term Vertical take off and landing (VTOL) is often used to refer to this ability. Here, we further subdivide the copter category in two: multicopters and helicopters. As the name suggests, multicopters have multiple propellers. The minimum number of propellers is three, although this model is rare. Much more common are the quadcopters, but also hexacopters and octacopters exists; they are typically capable of carrying a heavier load, and

can withstand the failure of one (or multiple) engines. The protocols in this master thesis are designed and tested for multicopters. However, they can be directly used for all VTOL UAVs, and some ideas are even transferable to fixed wing UAVs.

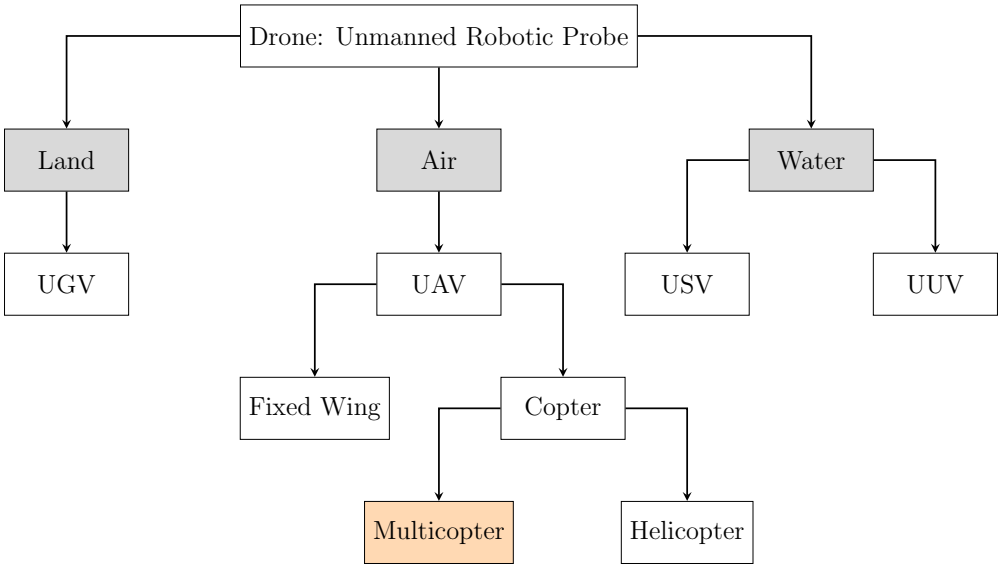


Figure 3.1: Classification of different drones.

As mentioned, in this work we have specifically chosen to work with multicopters. The reason for this is that multicopters have many advantages. First of all, compared to other types of UAVs, multicopters are the cheapest. When working with a swarm, this is a particularly important factor. The price of a multicopter can vary greatly, from a couple of hundred euros for the cheapest models, up to several thousand for more advanced models. The main differences between them are: load capacity, battery life, maximum speed, build quality, camera quality (if included), and proprietary software. A second advantage is that multicopters have a high level of mobility. Due to how they are built, a multicopter can quickly rotate in three dimensions (see Figure 3.2): pitch (forward and backwards), yaw (turning around its axis), and roll (left and right). Furthermore, they are able to stay still at a fixed aerial position; this is not possible with a fixed-wing aircraft, but it is a crucial ability for many applications (e.g. bridge inspection). Furthermore, they can be equipped with different types of sensing devices. The most common is a camera; but also lidar, radar and air pollution sensors are often used. Finally, multicopters are relatively easy to manage.

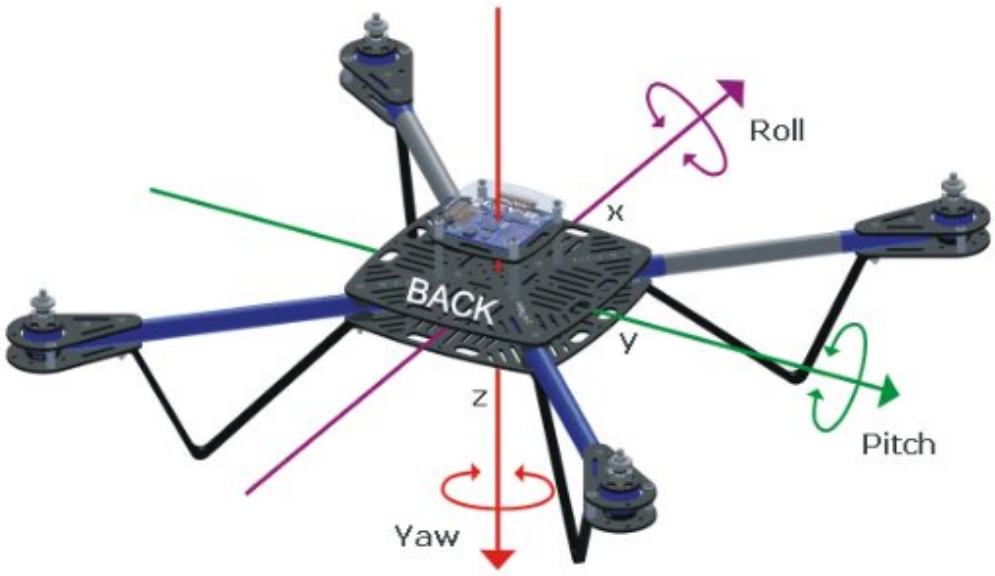


Figure 3.2: The three dimensions in which a UAV can move: pitch, yaw, and roll

3.2 The hardware components of a multicopter

Although there are many differences between multicopters in terms of materials, layout, capabilities, price, etc., there are some necessary components that every multicopter must have. Those components are schematically depicted in Figure 3.3. The flightcontroller (in the center) is the brain of the multicopter, and all the other components are connected and controlled by it. Although there are many flight controllers in the market, the Pixhawk [22] open source flight controller is probably the most used one. Essentially, it is a small lightweight embedded computer with build-in sensors and various connectors. The Pixhawk 4, for instance, is equipped with an 32-Bit Arm Cortex-M7, two accel/gyro meters, a magnetometer, a barometer, and a build-in GPS module. Furthermore, it has various interfaces such as: PWM, PPM, serial, I2C, SPI, CANbuses, analog inputs, etc.

The multicopter is obviously also equipped with motors (minimum of 3) and propellers. In most cases these motors are brushless, and they need to be controlled by an Electronic Speed Controller (ESC). The ESC converts the low current signal from the flight controller into a higher current signal that controls the speed of the motor. Thus, to prevent failure of the ESC, it is very important that an adequate ESC (one that can provide the current

drawn by motors) is chosen.

The motors, and the other components to a lesser extent, use energy. In most cases, this energy is provided by Li-Po batteries. Yet, in the case of very large multicopters, energy can also be provided by combustion engines. For the majority of multicopters, Li-Po batteries are preferred because they have a high power to weight/volume ratio (w.r.t other battery types). Li-Po batteries typically have a number of cells connected in series. Each cell has a voltage potential between 3v and 4.2v, depending on the depletion level. Thus, the number of cells (typically 3 or 4) in one battery determine the overall battery voltage. Each battery has also a certain capacity, which can be extended by placing cells or batteries in parallel. For a typically multicopter, the battery capacity is 3000 mAh. The battery capacity influences the maximum flight time. However, since the flight time also depends on the weight, motors, propellers, and many other factors (including weather conditions), a simple mapping between battery capacity and flight time cannot be made. Nevertheless, we could say that, in general, a battery with a higher capacity will increase the flight time. The energy from the battery cannot be used directly as the various components operate at various voltage levels; therefore, a power module is used. This power module will distribute the energy to all other components, except for the ESCs.

In theory, when the above-mentioned components are mounted on a frame, the multicopter can fly. However, in almost all cases, additional components are used. Probably the most important ones are the transmitter and receiver. With the transmitter, which operates in the 433 MHz band in Europe, or in the 915 MHz band in the USA, the multicopter can be controlled by a pilot. A receiver is connected to the Pixhawk to receive the instructions send by the pilot. For the communication between the transmitter and receiver a protocol is used. The *de facto* standard is the Micro Air Vehicle Link (MAVLink) protocol. Furthermore, usually a switch is added to the frame to prevent an unwanted takeoff of the drone, as well as a buzzer to notify the pilot about the states of the multicopter, and a telemetry kit to send and receive more information. Finally, although not required, the GPS and compass can be improved by connecting a dedicated sensor.

3.3 Software

As mentioned above, the flight controller is a lightweight embedded computer. The flight controller uses the information it receives from various sensors (and the pilot) to control the drone. This behaviour is of course implemented in code. Many different firmwares exist, and it is important

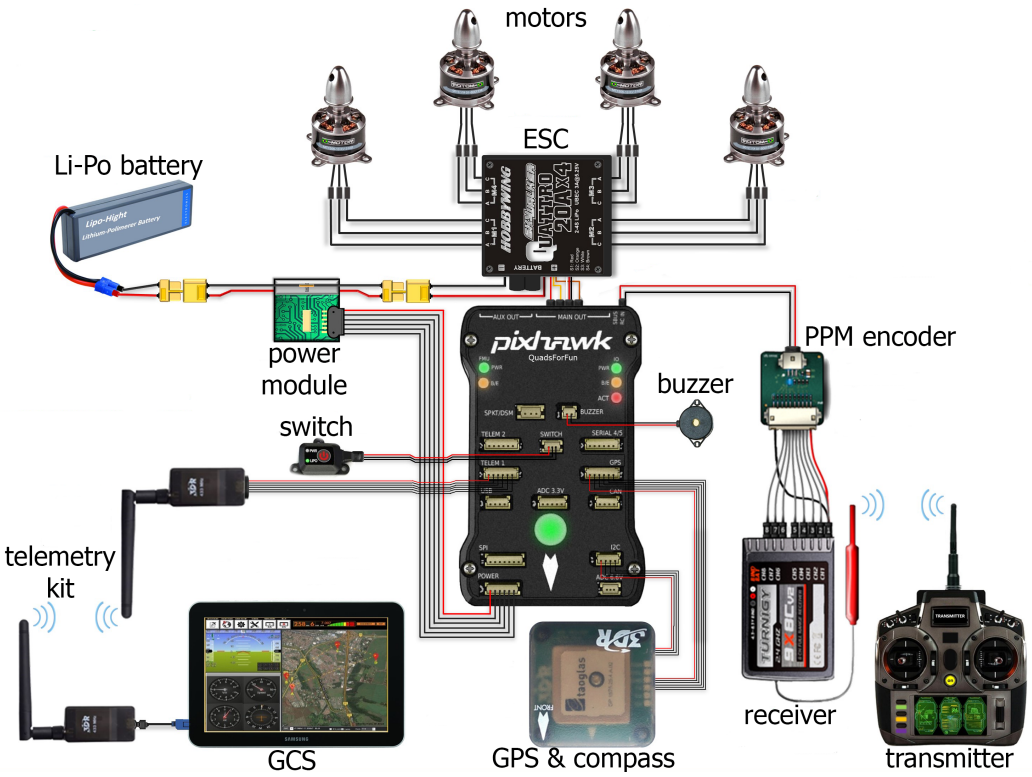


Figure 3.3: Basic components, and wiring of an open source multicopter.

that the firmware is compatible with the specific flight controller and drone model. A very famous one, which is compatible with the Pixhawk, is called ArduPilot. It is famous because it is very advanced, fully open-source, and it supports many vehicle types such as: multi-copters, traditional helicopters, fixed wing aircraft, boats, submarines, rovers and more. The specific project that contains the code to controls UAVs is called Arducopter [23]. The project is enormous and Arducopter has many features, the most important one with respect to our works are the following:

- **No vendor lock-in:** ArduPilot is fully open source, and free to use.
- **Autonomous missions:** Arducopter allows to plan flights based on GPS waypoints, and perform the mission fully automatically. Including taking off, returning home, landing and disarming the UAV.
- **Simulator:** The ArduPilot project also includes a simulator, which allows us to try out our protocols quickly and safely before deploying them on real UAVs. Unfortunate, this simulator is only capable of

simulating one UAV. Therefore, we have developed our own simulator (explained in Chapter 4) which internally uses the ArduPilot simulator.

- **Flight modes:** ArduCopter also includes many flight modes such as: altitude hold, loiter, fully automatic, etc. This allows us to develop protocols on a higher abstraction level, without being concerned of maintaining the UAV stable in the air.

Furthermore, the ArduPilot can also receive commands from a radio controller. Our protocols are usually always fully automatic. However, internally (as explained in chapter 4) we pretend to use a radio controller to control the drone, while in fact the controls come from a Raspberry Pi. The communication protocol between the UAV and radio controller is called MAVLink[24]. MAVLink is a well-established lightweight message serialization protocol, and it is released under the LGPL licence. Typically, the messages are small and therefore, it can be reliably be transmitted over different wireless mediums with low data rates. Reliability is ensured a double checksum verification in its packet header. All these (and others) features make the MAVLink protocol one of the most popular. The MAVLink protocol already exists of two versions. The newer version (2.0) released in early 2017 is backward compatible but includes several improvements such as a signature field of 6 bytes.

Chapter 4

ArduSim: a multi-UAV simulation platform

Performing real experiments with a swarm of UAVs is time-consuming, dangerous, and costly. Therefore, it is very important that we are certain that our protocols work before deploying them on real UAVs. Therefore, we use ArduSim [6], a simulation platform that can simulate many UAVs at the same time and with great accuracy. Any protocol in ArduSim is directly portable to real multicopters; thus once protocols are thoroughly tested, deployment is somewhat trivial. Furthermore, ArduSim is open source, well documented, and available online [25] under the Apache License 2.0. It is developed in Java, and has modular structure; which makes it possible to implement new protocols without being concerned about the underlying implementations of communication protocols and drone control. The simulator has many features, which are explained in great detail in [6]. In this work, we will highlight the key characteristics, and provide an overview of its various user interfaces. some of the features of ArduSim include:

- **Seamless deployment on real UAVs:** As mentioned in the previous chapter, the MAVLink protocol is one of the most frequently used protocols to control UAVs. ArduSim has incorporated this protocol into its core, and it uses the MAVLink protocol to fully control the UAV while it is flying. In order to deploy a protocol developed in ArduSim on real UAVs, the UAV should have a device capable of running Java (e.g. Raspberry Pi) attach to its flight controller telemetry port. The device should also be equipped with a WiFi adapter (with 5Ghz connectivity), so that it can communicate with a Ground Control Station (GCS) and other UAVs.

- **Complete Application Programming Interface (API):** ArduSim

was designed to abstract the UAV control, and communication between the UAVs. Developers that want to create a new protocol use an API. This API includes a complete set of functions to perform the most common manoeuvres such as: take-off, start a mission, pause a mission, move to a GPS location, land, communicate with other UAVs, create a formation, etc. This API allows the developer to ignore the underlying details and develop new protocols at a fast pace.

- **Comprehensive experiment data logging:** At the end of an experiment (both in simulations and in real experiments), ArduSim stores a fast amount of information about the flight. Including, among others, the path followed, heading, speed, acceleration, distance to origin, etc. The same path is also stored in Google Earth, ns-2, and OMNeT++ formats. This extensive data logging helps the developer in retrieving results and/or solving bugs.

- **UAV-to-UAV communication simulation:** When a protocol is deployed in real UAVs, ArduSim uses a WiFi adapter to connected to an Ad-hoc network with UDP messages. When ArduSim is used as a simulator this communication is replicated with the use of virtual links. The links are based on 802.11a technology, and models obtained by real experiments are used in order to resemble the realty as good as possible.

- **High scalability:** ArduSim was designed to be a multi-UAV simulator. With a high-end computer (Intel Core i7-7700, 32 GB RAM, SSD hard drive), ArduSim can easily run up to 100 UAVs in near real-time, and up to 500 UAVs in soft real-time. Performance can be improved by using a more performant computer, or/and the command line interface.

- **Various interfaces:** ArduSim can be used in three modes: (i) protocol testing on simulation, (ii) protocol deployment in a real multicopter, and also (iii) as a PC companion that aids real multicopters during their execution. With the PC companion, a user has the ability to start and stop protocols. It also allows recovering manual control in case the protocol does not behave as expected, which might avoid a crash. The simulation mode has two interfaces: graphical and command line. The graphical interfaces help the developer to design a protocol and verify its function. Later, the command line interface can be used for automatic testing. While using the command line interface parameters are set using parameter files (simple text files, with a specific format). When the parameter files are changed, no recompilation is required.

This allows the developer to write a simple script that changes the parameter file and executes ArduSim automatically. In this way, many experiments can be performed rapidly, without the need of intervention of the developer.

Out of the various interfaces, the graphical simulator is used the most. When ArduSim is executed in its graphical interface, a menu (shown in Figure 4.1) is opened. With this interface, the user can check all the general parameters and alter them where needed. The general parameters, are those that apply to all protocols in ArduSim; the most important general parameters are: the number of UAVs to simulate; the protocol to execute; the path towards the file with specific parameters for a given protocol; and the wireless communication model. However, as shown in Figure 4.1, many other parameters exist. For the convenience of the user, a *save* button is implemented which stores all the general parameters, and recalls them the next time ArduSim is executed. Depending on the protocol used, a new interface might appear. In that interface, parameters for specific protocols are shown, and can be changed as well. Since those parameters are protocol specific, it is the job of the developer to create that interface.

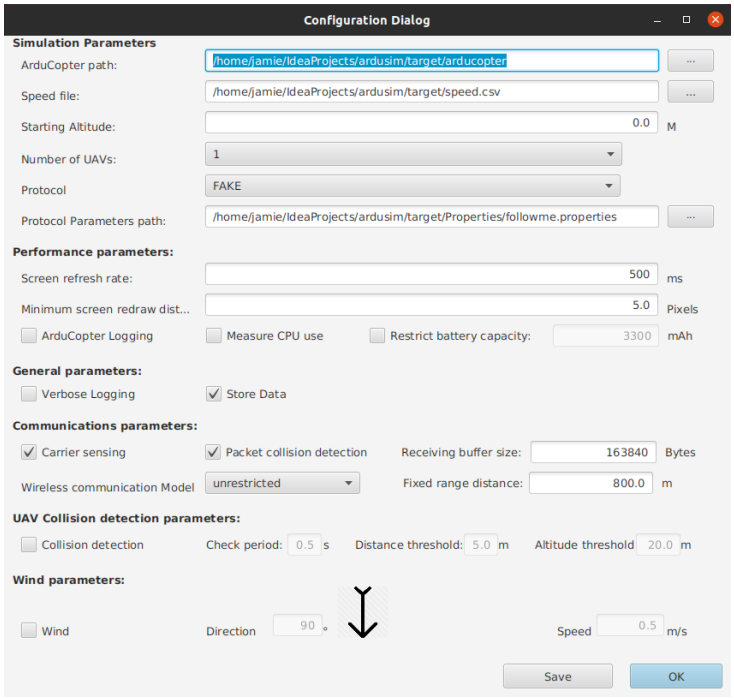


Figure 4.1: Menu of ArduSim: selecting general parameters and starting a specific protocol.

Once all the parameters are set, ArduSim is loaded and its main window (shown in Figure 4.2) appears. The largest part of the screen (rectangle 1) is filled with a map (either from Bing [1] or openstreetmap [2]). On that map, the UAVs are shown, and a thick coloured line is drawn to indicate where the UAV has flown to. If the UAVs are following a mission, a dotted line is also drawn to indicate where the UAV will go. Furthermore, to keep the rendering calculation to a minimum, ArduSim only shows the UAVs in 2D. Therefore, next to the UAV, the altitude is shown in text. More information about the UAVs, the state of the protocol, etc. is shown in the upper left corner (rectangle 3). Right next to it (rectangle 2), some buttons to control ArduSim are placed.

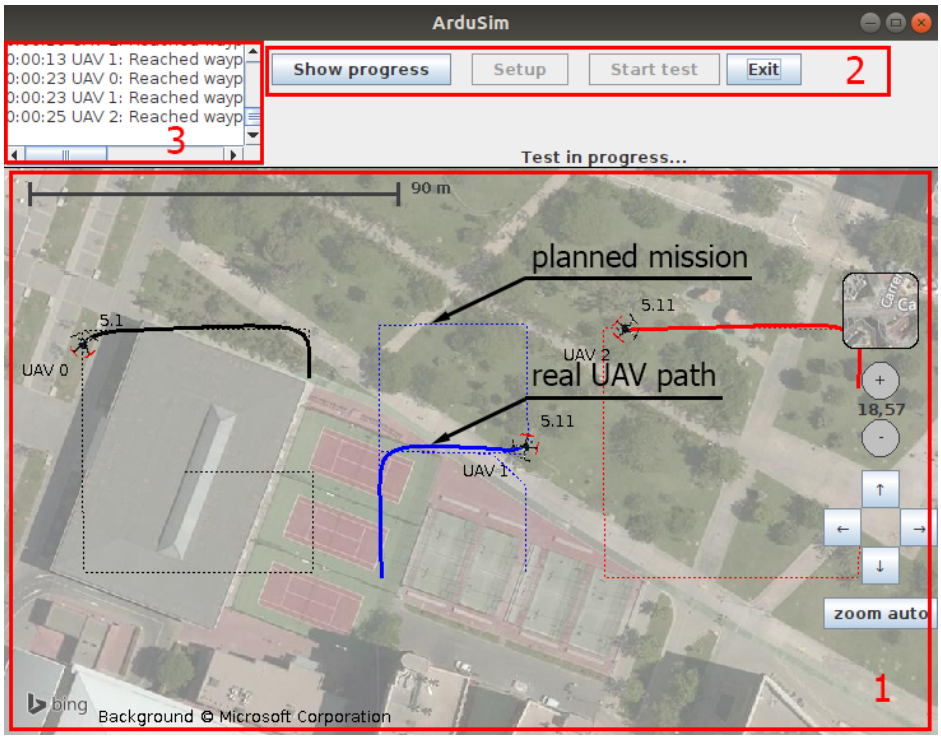


Figure 4.2: The main window of ArduSim.

[1] https://www.bing.com/maps
[2] www.openstreetmap.org

Chapter 5

Swarm formations

In general, there are two approaches to manage a swarm of UAVs; on the one hand, one can deterministically design a formation for a swarm and maintain this formation throughout a mission. On the other hand, one can use an approach that is called flocking; a behaviour that is common in nature, for instance in a group of fish, birds or insects. It consists of a few basic rules that are applied to each entity of the group. When those rules are respected, the group will stay united without collisions between the group elements. Both approaches are valid, although one approach can be more suitable for a specific situation. For this master thesis we have chosen to work with the first option because it suits better with the applications that we plan to develop in the future (such as search and rescue missions). Choosing this option implies that we have to determine the place of each UAV in the swarm, and maintain this place throughout the mission with methods such as the ones described in Chapter 7. For that reason, before detailing how those methods work, we will first explain how we designed the UAVs formations.

Currently, we have implemented four formations in ArduSim: linear, circular, matrix, and random. The first three formations are regular patterns that are useful for real applications. For instance, at the start of a search and rescue mission, the UAVs do not know exactly where the target is. At that moment it may be useful to spread out the UAVs using a linear formation. That way, each UAV can use a camera (or another sensor) to search for the target. Once, a UAV has indicated the target of interest, it is important to follow the target closely. At that moment a circular formation around the target might be more useful. Instead, the matrix formation can be used to cover some area uniformly while providing, e.g., network services. The random formation does not directly have a useful application, but it is nevertheless implemented to also validate our algorithms using irregular patterns.

5.1 Linear

The linear formation is the easiest formation, but nevertheless a very useful one. Basically, all the drones are on a line with some minimum spacing between them (see Figure 5.1). This makes its implementation (see Algorithm 1) very straightforward. The linear formation is used the most when a large area needs to be covered. This includes applications such as search and rescue [26], precision agriculture [27], and border surveillance [28].

Figure 5.1: Linear formation.

Algorithm 1 Linear formation(int numUAVs, double minDistance)

1: let *positions* be a list of FormationPoints of $numUAVs$ elements
2: $centerUAVIndex = \frac{numUAVs}{2}$
3: $i = 0$
4: **for** $i < numUAVs$ **do**
5: $\quad x = (i - centerUAVIndex) \times minDistance$
6: $\quad$ positions[i] = new FormationPoint(index=i,x offset=x,y offset=0)
7: **end for**
8: **return** positions

5.2 Circular

The second formation we implemented is a circular formation. As shown in figure 5.2, there is one UAV in the middle (the master) and all others are place along the rim. Although its implementation is equally straightforward, there is a small caveat. We have to take into account that, when the number of UAVs grows, the radius of the circle must grow as well. Otherwise, the minimal distance between the UAVs defined by the user cannot be guaranteed. The circle formation is especially useful for applications which closely track objects, as it becomes quite difficult for the target to escape from inside the circle; also, compared to the matrix formation, fewer UAVs are necessary to cover the same area.

5.3 Matrix

The matrix formation is the most complicated formation that we present in this work. At first glance, the problem looks easy, just fill the row and the

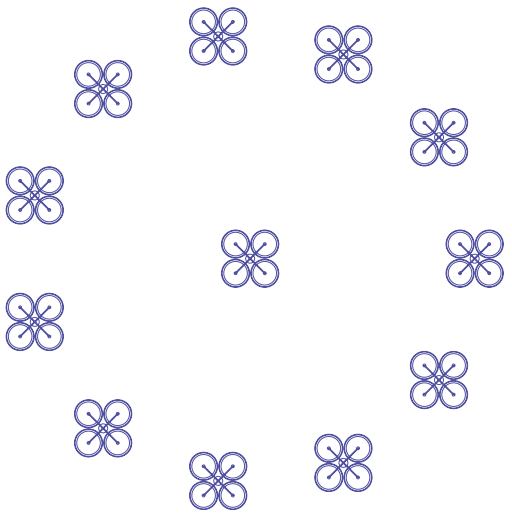


Figure 5.2: Circle formation.

columns. However, we want to be able to use this formation for any n positive integer number. This means that many times the grid will not be fully filled. In particular, when we have a large swarm, this can lead to an asymmetric and unbalanced formation. Furthermore, we also want to make sure that the UAVs are placed closed to the centre, to improve network connectivity. An example is given in Figure 5.4; as shown, the outer UAVs are placed: east, north, and then west. This allows us to maintain symmetry the best we can (if the swarm had one UAV more, it would have been placed south).

The matrix formation is basically created in two phases: first, a list of angles is created; second, the UAVs are placed, rotated around the centre, and shifted outwards. Let us first consider the rotation around the centre part. Each UAV is placed on location x=1;y=0, and is then rotated (around x=0;y=0) with a specific angle. This angle is calculated in the function *getStartAngles*, and it basically returns the following list: $0, \frac{\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{16}, \frac{7*\pi}{16}, ...$ Each angle in this list is used four times (but each time 90 degrees is added to it). With this mechanism, UAVs are first placed on the principal axis (east, north, west, south), and then on the diagonals, etc. Of course, when the matrix is full, we need to translate the UAV outwards. The smallest symmetrical matrix that we can make has 9 UAVs (3^2), the next 25 (5^2), 49 (7^2), 81 (9^2), and so on. In other words, we can check whether we have to increase the translation distances based on the parity of the base number. To finally put the UAVs at their correct positions, we have to round the x and y positions that we have calculated before (otherwise, they are placed more or less on a circle). For that, we have created a small function *ceilAbs(double)*,

Algorithm 2 Circle formation(int numUAVs, double minDistance)

let *positions* be a list of FormationPoints of $numUAVs$ elements
if numUAVs ≤ 7 **then**
    radius = minDistance
else
    radius = $\frac{minDistance}{2 \times sin(\frac{\pi}{numUAVs-1})}$
end if
positions[0] = new FormationPoint(index=0,x offset=0, y offset=0)
for $i < numUAVs$ **do**
    $x = radius \times cos(\frac{(i-1) \times 2 \times \pi}{numUAVs-1})$
    $y = radius \times sin(\frac{(i-1) \times 2 \times \pi}{numUAVs-1})$
    positions[i] = new FormationPoint(index=i,x offset=x, y offset=y)
end for
return positions

which manipulates the numbers as follows: if the value is close to zero, make it zero; if the value is positive, round it down; otherwise, round it up (always to the closest integer).

Figure 5.3 depicts this process for the 14th UAV in the matrix formation. First, the UAV is placed; since $14 > 3^2$ the UAV is placed on coordinate (x=2,y=0). Secondly, the appropriate angle out of the list (*getStartAngles*) is chosen. Each (start) angle in that list is used 4 times; therefore, the fourth element is selected ($14/4 = 3$ integer division, and the list is zero-based indexing). This angle corresponds to $\frac{\pi}{4}$. Afterwards (step 3), that angle is incremented with $2 \times \frac{\pi}{2}$ (because 14-12 = 2). Finally, to ensure that the the UAV is place on a matrix and not on a circle, its coordinate is rounded to the closest integer.

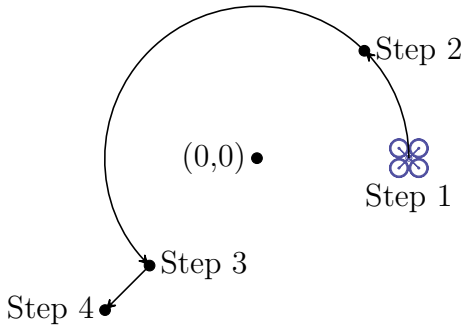


Figure 5.3: Example of the steps taken to calculate the position for the 14th UAV in a matrix formation.

Algorithm 3 Matrix formation(int numUAVs, double minDistance)

let *positions* be a list of FormationPoints of *numUAVs* elements
startAngles = getStartAngles(numUAVs)
distance = 0
i = 0
for $i < numUAVs$ **do**
    currentEven = $\sqrt{i}$ %2 == 0
    nextOdd = $\sqrt{i+1}$ %2 == 1
    if currentEven & nextOdd **then**
        distance +=1
    end if
    x=1,y=0
    theta = $i \times \frac{\pi}{2} + startAngles[i/4]$
    y = $x \times sin(theta)$
    x = $x \times cos(theta)$
    x = $ceilAbs(x \times distance \times minDistance)$
    y = $ceilAbs(y \times distance \times minDistance)$
    positions[i+1] = new FormationPoint(index=i+1,x offset=x, y offset=y)
end for
positions[0] = new FormationPoint(index=0,x offset=0, y offset=0)
return positions

Function getStartAngles(numUAVs)
    numUAVs = numUAVs -1
    let *angles* be a list
    if numUAVs $\leq$ 8 **then**
        angles.add(0.0)
        angles.add($\frac{\pi}{4}$)
    else
        while numUAVs > 0 **do**
            numUAVs = numUAVs-8
            angles.addAll(getStartAngles(numUAVs))
        end while
        length = angles.size()
        angles.add($\frac{\pi}{4 \times length}$
        angles.add(((4 $\times \frac{lenght}{2}$) $- 1) \times \frac{\pi}{4 \times length}$
    end if
End Function

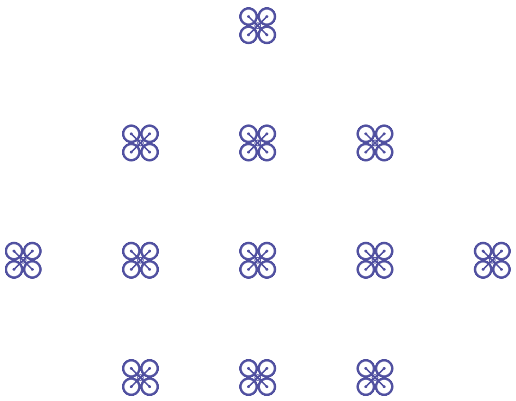


Figure 5.4: Matrix formation.

5.4 Random

The last formation that we implemented is a random formation. As stated above, it does not have a real application. There is no good reason to fly having the UAVs at random positions. However, the UAVs can be placed on the ground in a random position; or we might want to experiment how our algorithms perform with asymmetrical formations. We could place the UAVs in random positions, and relocate the ones that do not keep the minimum safety distance until they do. However, that approach does not work efficiently for a high number of UAVs, which can thus lead to very long calculation times. Therefore, we opted for a different approach. First, a (square) grid is created, of which its length depends on the number of UAVs, and an occupancy rate given by the user. Then the cell numbers (counting from left to right, and top to bottom) are placed in a list. For each UAV a random element of that list is picked (and removed once used). Then the UAV is placed in the cell with some additional random offset, in both x and y, called jitter. In this way, we can place all UAVs in a random position (efficiently $O(n)$), while maintaining the minimal distance.

Algorithm 4 Random formation(int numUAVs, double minDistance)

let *positions* be a list of FormationPoints of $numUAVs$ elements
occupancyRate = 0.5
maxJitter = 0.4
lengthSide = $ceil(\sqrt{\frac{numUAVs}{occupancyRate}})$
let *freeSquares* be a list of $lengthSide^2$ elements with value equal to the index
halfLength = $\frac{lengthSide}{2}$
positions[0] = new FormationPoint(0,0,0)
centerCellNr = $halfLength \times lengthSide + halfLength$
freeSquares.remove(centerCellNr)
i = 1
for $i < numUAVs$ **do**
    randomIndex = random.getInteger(max=freeSquares.size())
    cellNumber = freeSquares.get(randomIndex)
    freeSquares.remove(randomIndex)
    row = $\frac{cellNumber}{lengthSide} - halfLength$
    column = $(cellNumber\%lengthSide) - halfLength$
    jitterX = $random.getDouble() \times minDistance \times maxJitter$
    jitterY = $random.getDouble() \times minDistance \times maxJitter$
    x = $row \times (1 + maxJitter) \times minDistance + jitterX$
    y = $column \times (1 + maxJitter) \times mindistance + jitterY$
    positions[i] = new FormationPoint(index=i,x offset=x, y offset=y)
end for

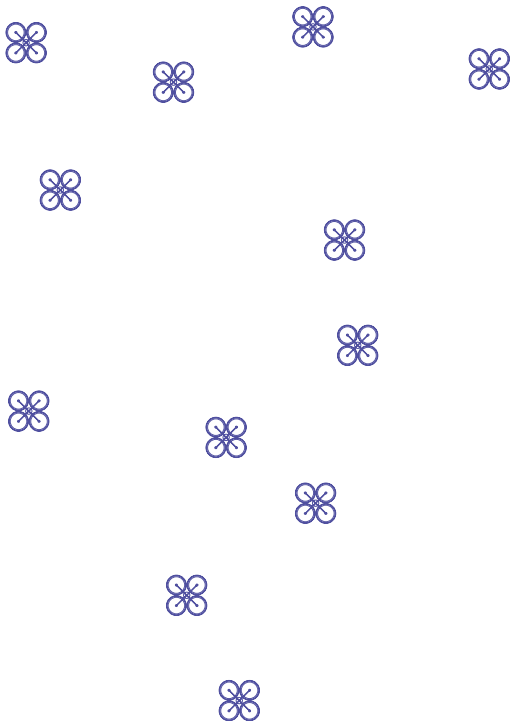


Figure 5.5: Random formation.

Chapter 6

Efficient UAV swarm takeoff

In the previous chapter we have described how UAVs can be placed in a specific formation. In this chapter, we will explain how we can take off a swarm of UAVs efficiently and safely using those formations. Before the start, the UAVs are placed on the ground in a specific formation, i.e. the ground formation. The ground formation is normally not important and is often random. The next step in any protocol is to take off the entire swarm of UAVs. Once they have taken off, they are supposed to be in a specific formation in the air, i.e. the flight formation. The flight formation can of course be chosen by the user, and it depends on the specific protocol or application. The problem that we solve in this chapter is: which UAV on the ground goes to which place in the flight formation (see Figure 6.1). From a user perspective, any assignment is acceptable (because all the UAVs have the same capabilities). Thus, we search for a solution that minimizes the total distance travelled by all UAVs, because this will reduce the flight time and battery usage as well. This problem is commonly referred to as the assignment problem.

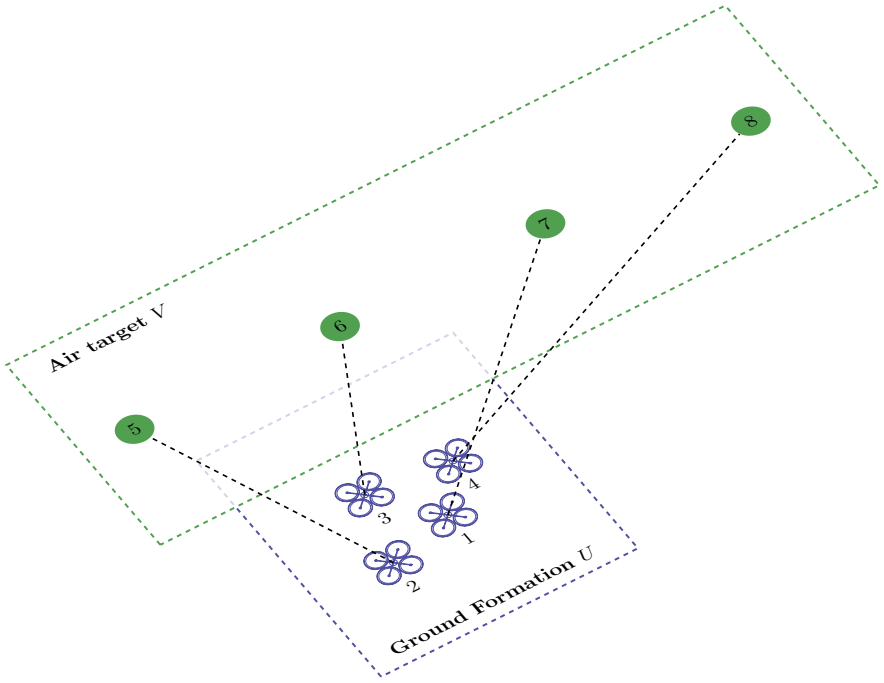


Figure 6.1: A graphic presentation of a possible UAV swarm flight assignment.

A first approach could be to just try all the possible arrangements, and pick the best one. However, the calculation time of such a brute force algorithm has a factorial growth, and therefore cannot be used in practice (even for a low number of UAVs (< 20)). For instance, as shown in Figure 6.2, there are already many solutions for just four UAVs. Therefore, in this chapter we develop, implement, and test two different algorithms: a heuristic which returns a sub-optimal solution very quickly ($O(n^2)$), and an approach using the KMA[29], which returns the optimal solution in $O(n^3)$.

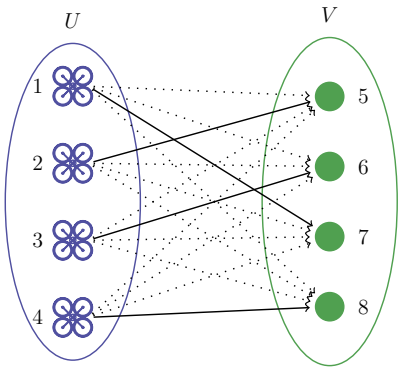


Figure 6.2: All possible solutions with only four UAVs.

6.1 Heuristic

The first algorithm that we propose to solve the assignment problem is a heuristic. This means that it will not provide the optimal solution, but one that comes closer, and most importantly it is very fast. Our heuristic starts from the brute force algorithm mentioned above, but to decrease the calculation time we no longer check every solution. In our heuristic, all the locations on the ground are reduced to one point, which lies in the middle. Then, the distances between the central position and the air locations is calculated and stored in a list, which is later sorted in descending order. Using this sorted list, the UAV closer to each of these positions is then assigned to it, as shown in Algorithm 5. Since most ground formations are more or less symmetrical, the impact of this estimation on the total distance travelled remains small (as shown later). However, it greatly reduces the calculation time to only $O(n^2)$.

6.2 The Kuhn-Munkres algorithm

After a more intensive literature review, we found an algorithm that was developed by James Munkres in 1957 [29]. It is called the Kuhn-Munkres algorithm (KMA), because his work was based on an earlier work of H.W. Kuhn [30]. This algorithm (sometimes referred to as the Hungarian algorithm) also gives an answer for the assignment problem. Due to clever matrix manipulation, it can obtain the optimal solution in only $O(n^3)$, which is much better than the $O(n! \cdot n^2)$ of the brute force algorithm. In the original problem, the authors were trying to match a set of n persons to a set of n jobs, where each person had a specific cost for a job. As can be seen, this problem is

Algorithm 5 HeuristicTakeOff(numUAVs, groundLocations, flightFormation)

Require: $groundLocations.size = numUAVs \wedge$
$flightFormation.size = numUAVs$

```
1: centerLocation = mean(groundLocations)
2: airLocations = f(centerLocation, flightFormation)
3: airList = ∅
4: for loc in airLocations do
5:     airList ← (loc, loc.distance(centerLocation))
6: end for
7: sort airList in descending distance order
8: fit = ∅
9: for aLocation in airList do
10:    bestError = MAX_VALUE
11:    for gLocation in groundLocations do
12:        error = gLocation.distance(aLocation)^2
13:        if error < bestError then
14:            bestError = error
15:            bestID = gLocation.ID
16:        end if
17:    end for
18:    fit ← (id, groundLocations[bestID], aLocation)
19:    groundLocations.remove(bestID)
20: end for
21: return fit
```

almost identical to ours. We only have to replace the persons with UAVs on the ground, the cost with the distance, and the job with locations in the air. Therefore, we have implemented the KMA in ArduSim as well in order to compare it against our own heuristic, which is faster but does not give the optimal solution.

Before we can execute the KMA we need to define a cost matrix. In our case the cost matrix is created based on the distance between the ground locations and the flight formation (as shown in Algorithm 6).

The KMA tries to create elements with the value zero, and distinguishes some zero elements by calling them starred and primed zeros. Furthermore, it also works with lines (vertically and horizontally across a row or column) which can be covered or non-covered. Therefore, any element of the matrix is said to be non-covered, once-covered, or twice-covered, depending on how many covered lines it lies.

In the beginning of the algorithm: no lines are covered, and no zeros are starred or primed. Then, for each row, the smallest element is found and subtracted from the other elements in that row. This will create at least one zero for each row. Afterwards, consider a zero Z in the matrix. If there is a non-starred zero in its row or column, star this Z, and repeat this for all zeros in the matrix. Finally, cover every column containing a starred zero.

After these preliminaries steps, the algorithm begins (**Step 1)** by choosing a non-covered zero Z and prime it. If there is no starred zero in its row, go to Step 2 immediately. Otherwise, cover this row and uncover the column of Z, repeating until all zeros are covered; then go to Step 3.

In **Step 2**, a sequence of alternating starred and primed zeros is constructed as follows: Denote the only uncovered 0' as Z_0. Then, denote the 0* in Z_0's column as Z_1 (if any). Continue by denoting Z_2 as the 0' in Z_1's row. Continue this sequence until there is no 0* in the column of Z_k. Next, unstar each starred zero in the sequence, and star each primed zero. Remove all primes, uncover each row, and cover every column containing a 0*. If all columns are covered, the starred zeros form the desired independent set (**end**). Otherwise, return to step 1.

Step 3 consists of getting the smallest non-covered element h of the matrix, adding h to every element in a covered row, and subtracting it from every element in an uncovered column. Then return to step 1 without altering star, prime or covered lines.

Algorithm 6 distanceMatrixCalc(numUAVs, groundLocations, flightFormation)

Require: $groundLocations\ of\ size\ numUAVs \wedge$
$flightFormation\ of\ size\ numUAVs \wedge$

1: Let costMatrix be a matrix of size numUAVs * numUAVs
2: **for** $i \in \{0, \ldots, numUAVs\}$ **do**
3: **for** $j \in \{0, \ldots, numUAVs\}$ **do**
4: $errorMatrix[i][j] \leftarrow d(groundLocations[i], flightFormation[j])^2$
5: **end for**
6: **end forreturn** costMatrix

6.3 Takeoff procedure

With the two above-mentioned assignment algorithms we can only determine which UAV has to go to which place. It does not give us any information about the order of the takeoff. The takeoff procedure should be as fast as possible, but also guarantee a collision free flight. Those two restrictions are somewhat contradictory. In order to take off fast, the takeoff should be simultaneous. However, this increases the chances of collision. A sequential takeoff procedure, on the other hand, guarantees a collision-free flight, but it is much slower.

To address the aforementioned challenge we have implemented a semi-sequential takeoff procedure, where a UAV takes of vertically up to a certain altitude Z_t, and then it moves diagonally towards its location, following a straight line. From the moment that the UAV starts moving diagonally, it will tell other swarm members that another UAV can take off. The UAV that has to fly the furthest will take off first. This way collisions are avoided, and the takeoff time is reduced.

6.4 Performance assessment

Above, we have proposed two algorithms to solve the assignment problem: our heuristic and the KMA. In this section, we will compare the two to decide which algorithm is the most appropriate to take off a swarm of UAVs in terms of: calculation time, the total travelled distance, and the number of possible

collisions. In all experiments, we have examined the influence of the number of UAVs, and also the type of flight-formation.

6.4.1 Computation time analysis

As expected, our heuristic ($O(n^2)$) returns an assignment faster than the KMA ($O(n^3)$). In Figure 6.3, we can observe that there is a difference of 100×. Furthermore, figures 6.4 and 6.5 show that the computation time is independent on the formation when the heuristic algorithm is used. This cannot be said for the KMA, which performs notably worse when the UAVs are spread out over one dimension (linear formation).

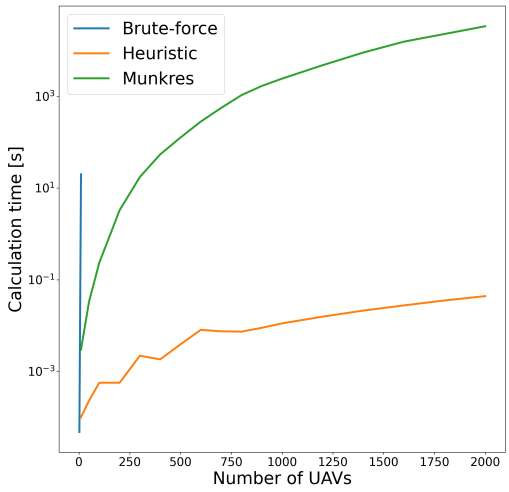


Figure 6.3: Average computation time for all UAV formations and assignment algorithms.

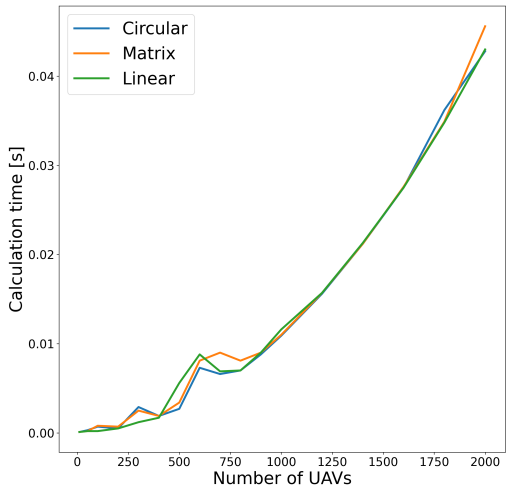


Figure 6.4: Computation time using the heuristic algorithm for various UAV formations.

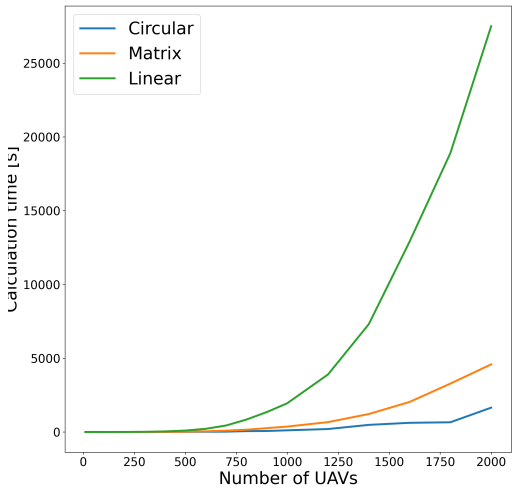


Figure 6.5: Computation time using the Kuhn-Munkres algorithm for various UAV formations.

6.4.2 Travel distance analysis

Obviously, our heuristic only returns an assignment faster because that assignment is not the optimal one. Therefore, we have also measured the total distance travelled by all the UAVs. As shown in Figures 6.6 and 6.7, our

heuristic imposes an additional distance, which grows in an almost linear fashion. In the case of a circular formation, the additional distance travelled grows very quickly. This is because our heuristic calculates a centre point on the ground, and then calculates the distance to all the air locations. Since those air locations are on a circle, the distances will be very similar, and thus our heuristic performs poorly.

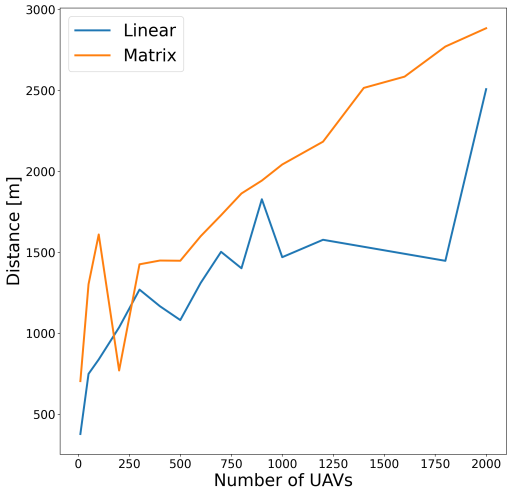


Figure 6.6: Total additional distance travelled by all UAVs when the heuristic algorithm is used for a Matrix and Linear formation.

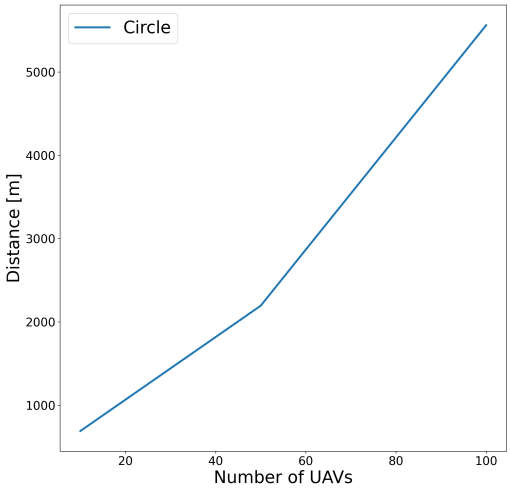


Figure 6.7: Total additional distance travelled by all UAVs when the heuristic algorithm is used for a Circle formation.

6.4.3 Number of collisions analysis

As we mentioned above, a simultaneous takeoff procedure is preferred because it reduces the takeoff time. However, since there is a chance of collisions (i.e. flight paths crossing), it is not used in practice. Therefore, we now use a semi-sequential takeoff procedure. In the future, we would like to improve this, so that it becomes semi-simultaneous. For such a procedure, it will be important that the assignment algorithm reduces the number of flight paths crossing. The lower the chances of collision, the easier it is to take off simultaneously.

Since the number of flight paths crossing each other will become an important metric in the future, we decided to measure it already for both assignment algorithms. We conducted an experiment where a number of UAVs took off simultaneously. Then, during the flight, we constantly measured the distance between all the UAVs. Whenever two UAVs came closer than 5 meters (a typical GPS error), a virtual collision is detected. After that collision, we allowed both UAVs to continue their flight, and thus they could collide with another UAVs later on.

From Figure 6.8, the KMA clearly performs better for the Matrix and Circle formations, where it avoids almost all collisions. However, as shown in Figure 6.9, the KMA does not fully guarantee a collision-free flight, but it merely lowers the number of possible collisions.

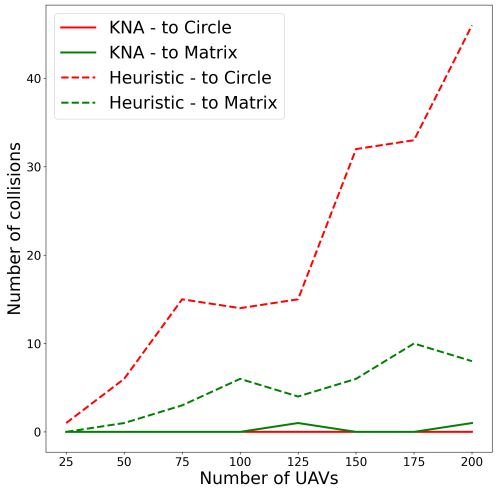


Figure 6.8: A comparison of the heuristic vs. the KMA algorithm in terms of potential collisions when varying the number of UAVs (Circle and Matrix formations).

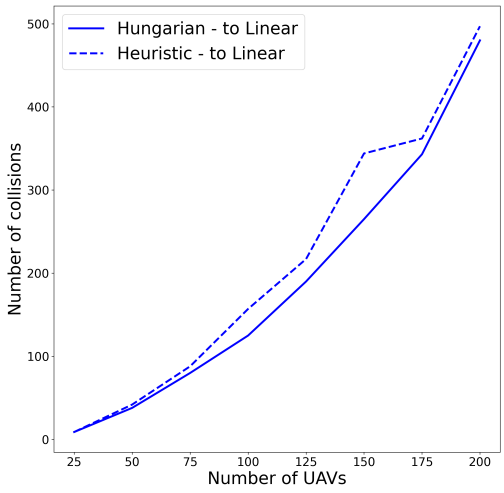


Figure 6.9: A comparison of the heuristic vs. the KMA algorithm in terms of potential collisions when varying the number of UAVs (Linear formation).

6.4.4 Takeoff time analysis

Finally, we measured the total takeoff time for the simultaneous and semi-sequential takeoff procedures. As shown in Figure 6.10, taking off all UAVs one-by-one takes a lot of time (so much that we stopped our tests after 200 UAVs). Therefore, there is a real need for a (semi)simultaneous takeoff. Those procedures allow to takeoff the UAVs in a nearly constant time. Such time is not perfectly constant because the total distance travelled will increase when the minimum distance between UAVs has to be maintained. This effect is mostly visible for the linear UAV formation.

6.5 Summary

In this chapter, we searched for a way to determine which UAV on the ground goes to which position in the aerial formation. This problem is called an assignment problem, and various algorithms to determine the assignment exists. The optimal assignment is, in our case, the one that minimizes the total distances travelled. Since a brute-force search for the optimal assignment is not doable within an acceptable time frame, and so we proposed and validated two algorithms: a very fast heuristic, which calculates a sub-optimal assignment, and the Kuhn-Munkres algorithm, which is slower but returns the optimal assignment. Experimental results have shown that our heuristic is faster, and that it only increases the total distance travelled by a small

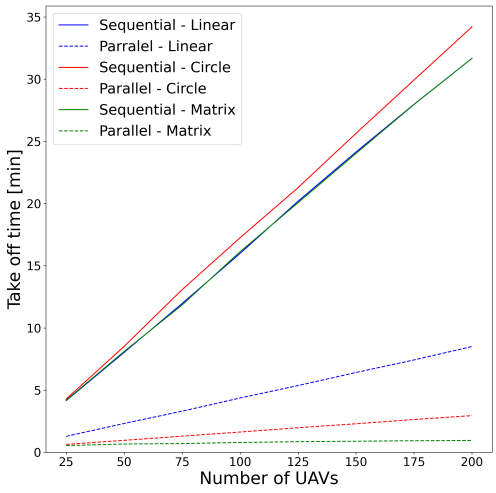


Figure 6.10: Total take-off time using the KMA with varying formations and take-off procedure

amount. Nevertheless, in most cases, the KMA is better suited than the heuristic approach. This occurs because it drastically reduces the number of collisions and, therefore, allows for more simultaneous takeoffs, which in turn decreases the overall takeoff time. However, when the size of the swarm grows, the time required to calculate the assignment with theKMA exceeds the acceptable limits.

Chapter 7

MUSCOP: a resilient coordination protocol

Now that we are able to take off a swarm of UAVs safely and efficiently, we want to move that swarm according to a pre-planned mission, while maintaining the swarm in a tight formation. In the past, our research group created Mission-based UAV Swarm Coordination Protocol (MUSCOP) [31], that achieves that very goal. However, it relies on a master-slave model, and in that (initial) version of MUSCOP no attention was given to the possible UAV failures during the flight. The result was that MUSCOP can maintain a desired flight formation with minimal time overhead (0.55s per waypoint), as long as no UAV fails. In this master thesis we will start from that initial version of MUSCOP(V1.0), and expand it so that it is able to continue the mission even if UAVs fail during the flight. First, we will briefly describe how MUSCOP V1.0 works, then we will explain how we improved it, and how we validated the new version (V2.0) through various experiments.

7.1 The MUSCOP protocol V1.0

As stated above, MUSCOP succeeds at keeping a stable flight formation while performing a planned mission (which consists of various GPS-waypoints). The protocol relies on a master-slave model, where the master UAV synchronizes all the slaves at every waypoint in the mission. MUSCOP is a complex protocol, which depends on many messages, states, transitions, etc. Since it is well explained in [31], we won't explain everything in this document. We will, however, give a general overview, with special enfasis on the things that will change in the new version.

Since MUSCOP relies on the master-slave model, the first step in the

protocol is to define who the master will be throughout the entire mission. In order to improve network connectivity, the master will be the UAV at the centre of the formation. Afterwards, the master will send information about the mission (GPS waypoints) to all the slaves. These waypoints are slightly adapted for each UAV (with an offset), so that it corresponds to their position in the formation. Once all the slaves have acknowledged that they have received their mission, the master will give the order to take off. At the moment that all the slaves have taken off, the master gives the order to go to the next waypoint. This order (message) is sent periodically (every 200ms). According to the authors, this redundant behaviour increases the reliability of the protocol, as the message sent among the UAVs could be lost due to distance or the presence of noise in the communications channel. The slaves, on the other hand, will broadcast periodically the last waypoint they have visited. The moment the master receives a message from each slave (with the correct waypoint), it will start broadcasting the order to move to the next waypoint. This behaviour repeats itself until the last waypoint is reached, and then the master will give the order to land.

The messages sent and received by the master and slaves determine when states have to be changed, and they will be used (in MUSCOP v2.0) to detect the failure of any UAV. Therefore, all messages with their format are shown in Figure 7.1. Each message starts with a *type* field, so that the receiver can identify the message.

At the beginning, the slaves send a *Hello* message (1). When the master receives this message, he will (i) add that UAV to its list of UAVs in the swarm, and (ii) he will calculate the specific mission (GPS waypoints) for that slave, based on the original mission and the location of that slave (included in the message).

Then the master sends a *Data* message (2), which includes the calculations made earlier, as well as additional information:

- *ID* - the ID of the target UAV;
- ID_c - the ID of the UAV that will be in the centre of the flight formation, i.e. the master UAV;
- *nUAVs* - the total number of UAVs in the swarm;
- *form* - the specific formation in which the UAVs will be flying;
- *pos* - the position of the target UAV in that formation;
- h - the heading of the swarm;

- z - the altitude for the takeoff phase;
- n - the number of waypoints;
- $(x, y, z)_i$ - the waypoints.

The master will broadcast these messages (all data messages, but specific for each slave), until all the slaves have sent back an acknowledgement (*DataAck* message 4).

Once all the master receives an acknowledgement for the data messages (*DataAck* message (4)), it will start broadcasting the third message: *ReadyToFly*. If the slaves receive this message they will (i) arm the engines, and (ii) send back an acknowledgement message (*ReadyToFlyAck* (4).

Once the master knows all the slaves are ready to fly, it will periodically (every 200ms) send the fifth message (*moveToWP*); this message includes the waypoint identifier and, since the slaves received the location of that waypoint (in the data message), they know where to fly to.

Upon arrival at the waypoint, each slave will send message $reachWPAck_i$. If the master received this message from every slave, it will increment the waypoint identifier in the *moveToWP* message (which is still periodically broadcasted). This process will be repeated until the last waypoint is reached. In that case, the master will stop the broadcasting of the *moveToWP* message, and it will start broadcasting the *land* message.

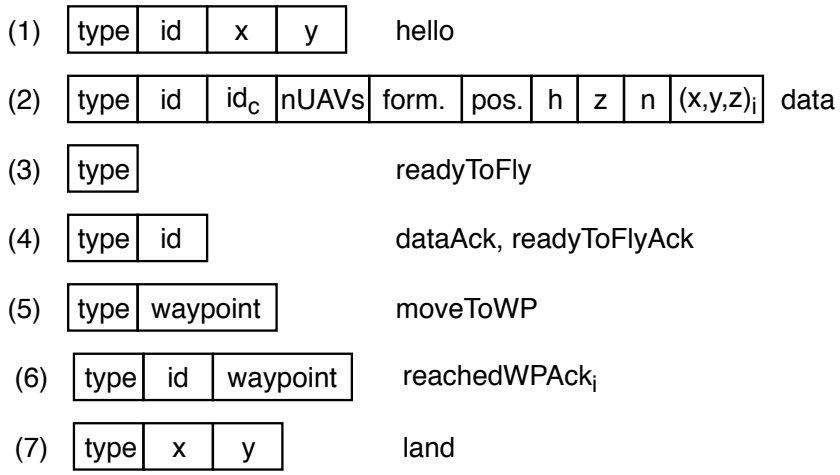


Figure 7.1: MUSCOP v1.0 messages and their format

With this functionality, the swarm will be synchronized at every waypoint, and therefore will not become unstable. Due to the synchronization of swarm elements, a small time overhead is introduced (just 0.55s per waypoint).

7.2 MUSCOP v2.0

As mentioned before, MUSCOP V1.0 works correctly if none of the UAVs fail. The failure of a UAV can occur for many reasons: a UAV could have collided with something (bird, building, other UAV, etc.), the batteries could be depleted, there could be an error in the code, mechanically something can come loose, or it could just be too far removed from the swarm that it is unable to communicate. Whatever the reason is, in MUSCOP V1.0 the entire swarm will be stopped at the next waypoint and unable to continue the mission. This is because, if the master fails, the slaves will never receive the message to go to the next waypoint; additionally, if any of the slaves fail, the master will wait for it before continuing. This results in a swarm that just keeps hovering at a waypoint, until the batteries are depleted. This is of course undesirable. To cope with this problem, we propose to use the messages that are broadcasted periodically. If those messages fail to arrive, we can assume that the corresponding UAV has failed, and we should act accordingly.

Basically, there are two cases to consider: the master can fail, or a slave can fail. In case that a master fails, a new master should be denoted, and it should take over the function of the previous one. In the case a slave fails, the master should not wait for messages of that failed UAV and just continue with the mission.

To replace the master in case it fails, a backup master should be denoted. To provide full resilience, we have opted to create a list of backup masters which includes all UAVs since, theoretically, all UAVs could fail, although such event is extremely rare in real experiments. This list of backup masters could be chosen randomly. However, as stated before, in order to maximize the network connectivity, it is in our best interest that the master is in the middle of the formation. In the chapter dedicated to the takeoff procedure, we have said that the UAV who has to fly the furthest should take off first. Since we have done the calculation there, we can just reuse that list. Hence, in order to get a list of the UAVs closest to the centre, we should just reverse that list. Now that we have a list of possible backup masters, we still have to decide when to switch to another master. For that, we propose to use timeouts. As shown in Algorithm 7, every UAV has a list consisting of the ID of all the other UAVs. This list also contains a timestamp, which is updated every time a message of the corresponding UAV is received. When the UAV arrives at a waypoint, the first thing they do is check the list. If there is any UAV whose timestamp is older than *TimeToLive* (set at 5s), then that UAV is removed from the swarm. We also check if the master should be changed. With this approach, the swarm is resilient to any and

multiple failures.

This process, that takes place in every UAV, is shown in Figure 7.2. Here we can see that each UAV keeps track of the activity of all the other UAVs in the swarm by listening to the various messages that are sent. When no message is received for a period longer than a certain Time To Live (TTL) value, that UAV is placed in the failed state. Upon arrival at a new waypoint, the swarm is updated (locally in each UAV). This is done by removing all the UAVs that are in the failed state from the list of UAVs that are currently in the swarm. While removing the UAVs from that list, they check if that UAV was the master. In case that is true, a new master should be selected from the list of backup masters. This process is fully done locally in every UAV. If the ID of the new master corresponds with the ID of the that UAV, it will start broadcasting a message saying that he is the new master of the swarm. Once the other UAVs receive and acknowledge that message, the new master will continue by sending the *moveToWP* message, as explained in MUSCOP v1.0.

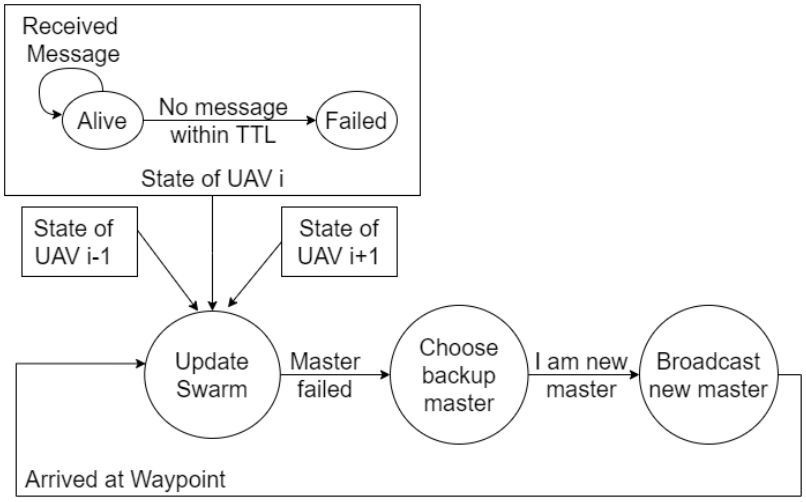


Figure 7.2: The different states for MUSCOP v2.0.

7.3 Validation

To validate our protocol we have set up multiple experiments. In each experiment, we have measured the additional time overhead introduced by MUSCOP V2, and we compare this with an experiment where no UAV fails (so it behaves as MUSCOP V1). For all experiments, we have used four UAVs

Algorithm 7 UpdateSwarm(numUAVs, listOfMasters)

Require: *listOfMasters.size* = *numUAVs*

```
1: TimeToLive = 5s
   Setup phase:
2: Let LastTimeUAV be a hashmap of size(numUAVs)
3: for Id in numUAVs do
4:     if Id != selfId then
5:         LastTimeUAV.put(Id, currentTime)
6:     end if
7: end for

   Fly phase:
8: while waypoint not reached do
9:     if Message received then
10:        Id = readMessage()
11:        LastTimeUAV.put(Id, currentTime)
12:        Perform actions related to message
13:    end if
14: end while
15: while waypoint reached do
16:    for UAV in LastTimeUAV do
17:        UAVTime = LastTimeUAV.get(UAV)
18:        if currentTime - UAVTime > TimeToLive then
19:            LastTimeUAV.pop(UAV)
20:            ListOfMasters.pop(UAV)
21:        end if
22:    end for
23:    if selfId == ListOfMasters.getFirst() then
24:        IamMaster = True
25:    end if
26:    if IamMaster == True then
27:        Perform actions related to master
28:    else
29:        Perform actions related to slave
30:    end if
31: end while
```

that fly at 10 m/s in a linear formation, with a distance of 50 meters between them. The target mission is shown in Figure 7.3. The value of *TimeToLive* was set to 5 seconds.

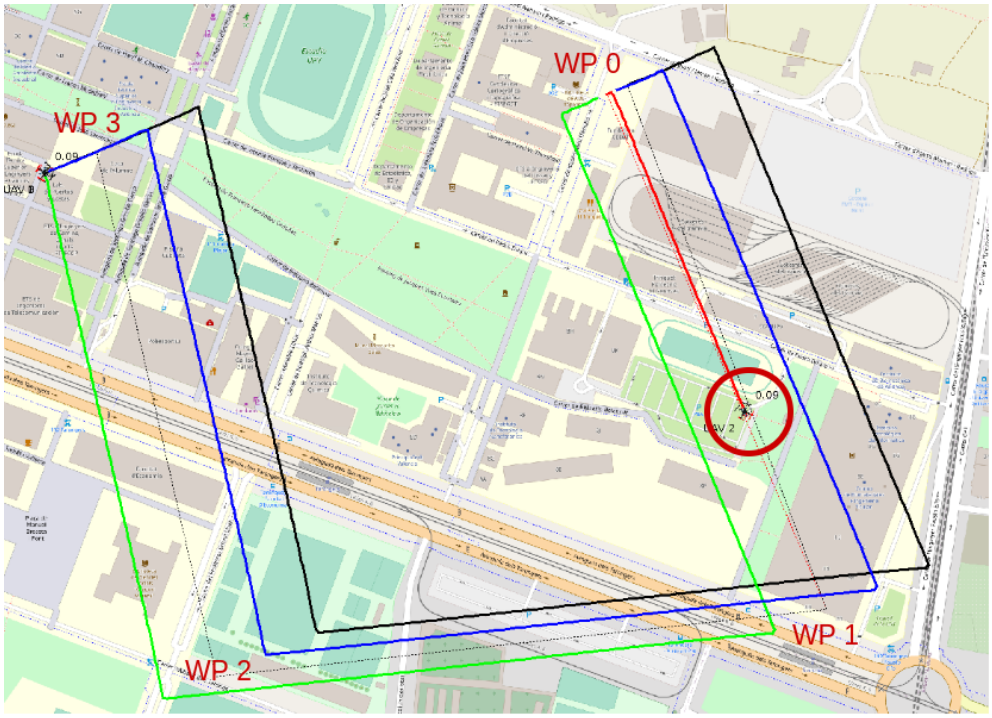


Figure 7.3: Example of MUSCOP V2.0 providing resilience, so that the swarm can continue its mission even when a UAV fails.

Since our protocol works with a timeout, we can classify the location where a UAV fails into three groups: (i) a UAV can fail exactly when it arrives at the waypoint (worst case scenario); (ii) just before a waypoint; and (iii) in-between waypoints. We categorize the locations into these three groups for a specific reason. In case (i) the UAVs will have to wait for the entire timeout time before removing the failed UAV from their list. This means that the entire swarm is halted for 5 seconds (i.e. the value of *TimeToLive*). If, on the other hand a UAV fails in-between waypoints (iii), then that timeout will already be surpassed, and thus the UAV will be removed from the list immediately, without any time overhead. Between those two extremes lies the second group. In that case, the swarm will have to wait until the timeout is surpassed, which will be less than the value of *TimeToLive*. For each group we tested the following five scenarios:

A: a single slave failing at waypoint 1.

B: a single master failing at waypoint 1.

C: two slaves failing at waypoint 1.

D: a master and its backup failing at waypoint 1.

E: a backup master failing at waypoint 1, and the master failing at waypoint 2.

Overall, this results in 15 different experiments, plus one control experiment where no UAV fails.

In our first set of experiments, the UAVs fail just when they arrive at the waypoint, which is the worst case scenario. As shown in the Table 7.1, the time overhead introduced is the same as the value of *TimeToLive*, i.e., 5 seconds. We can also observe that there is no difference between the failure of a slave and a master, and that neither is the time overhead increased when multiple UAVs fail. As one can observe, some values in the table are negative. This simply means that the execution was a bit faster than the control case. Overall, those values are very small, and caused by imperfections in ArduSim, Java itself, or the even in the load on the CPU.

Table 7.1: Time overhead for the different scenarios just when reaching the next waypoint (0 m).

Section	A [ms]	B [ms]	C [ms]	D [ms]	E[ms]
0	-198	383	234	-128	-135
1	4999	5601	5601	5383	5002
2	- 3	-397	-397	-394	4802
3	0	203	4	0	0

The results of experiments of group II are shown in Table 7.2. Here the UAVs fail at 15 meters from the waypoint, which coarsely corresponds to 2.5 seconds before they arrive at the waypoint (half the value of *TimeToLive*).

Table 7.2: Time overhead for the different scenarios at 15 m from the next waypoint.

Section	A [ms]	B [ms]	C [ms]	D [ms]	E[ms]
0	- 77	594	20	123	75
1	2554	2993	2555	2099	3006
2	57	102	-153	-153	2101
3	- 1	301	147	147	-1

In Table 7.3 we show the results of our third experiment: where UAVs fail at 200 m from a waypoint (group III). As shown and expected, no extra delay is introduced. Since the UAV failed long before the swarm arrived at the waypoint, its timeout was surpassed, and thus the swarm could respond without introducing any additional delay.

Table 7.3: Time overhead for the different scenarios at 200 m from the next waypoint.

Section	A [ms]	B [ms]	C [ms]	D [ms]	E[ms]
0	150	122	- 32	140	112
1	- 50	- 33	- 1	300	-150
2	-148	-184	-104	- 51	99
3	448	-185	197	200	-206

From these experiments we can conclude that the additional time overhead introduced by MUSCOP V2.0 is often zero (since in most practical cases the failure will belong to group III). In the unlikely event a UAV fails close to a waypoint, the delay is bounded to the value of *TimeToLive*, as described in Equation 7.1. Furthermore, we have shown that the delay is independent on the type of UAV that fails (master or slave) and/or the number of UAVs that fail.

$$Delay[s] = \begin{cases} TimeToLive - t_{wp_i}, & \text{if } t_{wp_i} \leq TimeToLive \\ 0, & \text{otherwise} \end{cases} \tag{7.1}$$

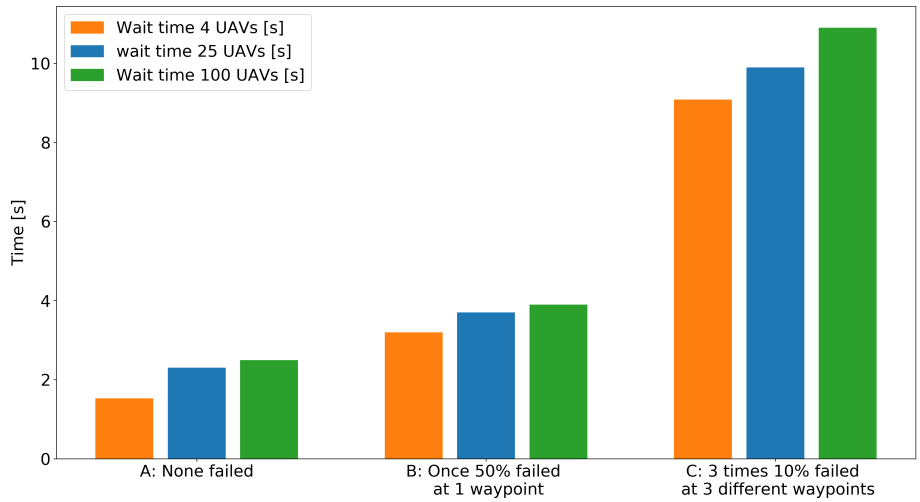
In the experiments above, just one UAV fails at one waypoint. Although, it is unlikely that in real experiments many more UAVs will fail at the same locations, we still wanted to measure the influence of scaling up. Therefore, we devised an experiment which tests MUSCOP V2.0 in a more extreme case. Specifically, we tested our protocol with a high number of UAVs (100), a moderate number (25), and with a low number (4). In those experiments, we let UAVs fail at 15 meters from the waypoint (group II). The flight speed was again set to 10 m/s, and the *TimeToLive* to 5 seconds. Again we measured the time, but this time we split it up into two parts: the flight time, and the wait time (at the waypoints). For each number of UAVs (100, 25, and 4) we performed three experiments:

a) A control flight where no UAV fails.

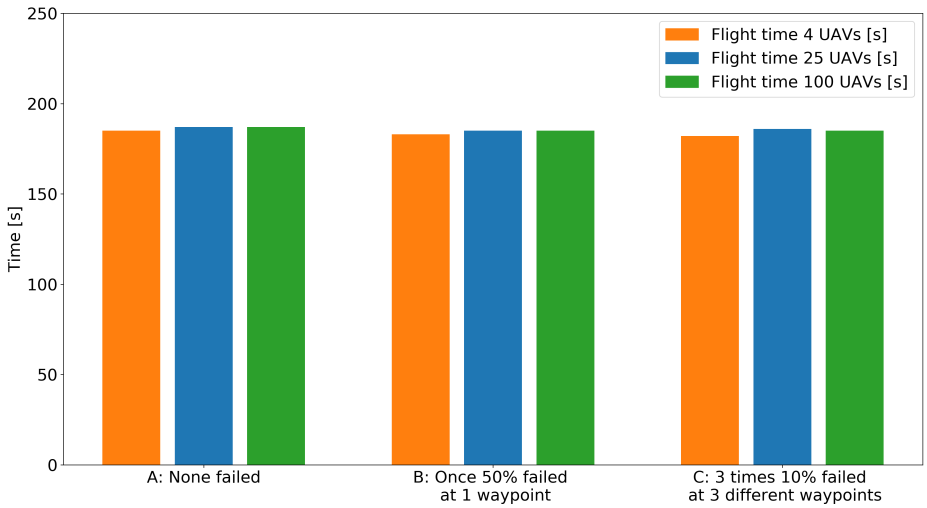
b) A flight where half the number of UAVs (and the master) fail at a particular waypoint.

c) A flight where 10% of the UAVs (and the master) fail at consecutive waypoints (3 failures overall).

The results shown in Figures 7.4a and 7.4b demonstrate that the impact of scaling-up the swarm is quite low. The flight time is unrelated to the number of UAVs, and of course it only depends on the travelled distanced and the flight speed. Notice that the wait time does increase slightly, due to message buffering. Furthermore, the time overhead is increased a bit when many UAVs fail at the same time (Scenario B). However, when compared to the overall flight time, it is negligible. This figure also confirms some of our previous statements, since one can clearly observe that, in those cases where UAVs are failing, a delay is introduced. As shown in the last case (10% of the UAVs failing at each waypoint), the overall delay grows with the number of times UAVs fail during the mission.



(a) Wait time overhead.



(b) Flight time overhead.

Figure 7.4: Time overheads when varying the number of UAVs that fail.

Chapter 8

Conclusions, publications and future work

8.1 Main conclusions

In the last decade there have been many technological advances in the field of UAVs. As a result, we see that nowadays UAVs are used for many different applications such as: surveillance, thermal inspections, the film industry, entertainment, etc. However, in most cases those applications only consider one UAV. We believe that, with the proper technologies, UAVs will fly in swarms in the near future. In many cases, swarms perform better because they can provide resilience, or complete the task more efficiently. Sometimes a swarm is even necessary, for instance to carry a heavy load, or to provide network connectivity over a large area. Swarms have one disadvantage, however; in general they are more complicated to manage, and the chance of collisions with other UAVs in the swarm is high. The most prominent challenges nowadays are the following: (i) swarm formation definition, (ii) takeoff assignment and procedure, (iii) in-flight coordination, (iv) handling the loss of swarm elements, (v) swarm layout reconfiguration, (vi) communications and data relaying optimization, and (vii) controlled landing.

In this Master thesis we have solved the first four problems. Below we briefly summarize how we solved those problems, and what results were obtained.

Swarm formation definition: In total we have designed and implemented four different swarm formations: Linear, Circle, Matrix, and random. The algorithms that create these formations are flexible, and they can be used for any number of UAVs; also, a minimal distance between each UAV can be set by the user.

Takeoff assignment and procedure: The takeoff is the first and arguably the most important step any swarm has to undertake. It should be safe (collision free) and efficient (short takeoff time). The takeoff consists of two phases: first, in the assignment phase, an algorithm calculates which UAV on the ground has to go to which position in the air; secondly, a takeoff procedure determines the order and when the UAVs have to take off. We developed two algorithms to calculate the assignment. One very fast heuristic that gives a sub-optimal (in terms of total distance travelled) solution, and one slower algorithm (the KMA) that returns the optimal solution. After many experiments, we came to the conclusion that the KMA is the most appropriate algorithm, even though its calculation time is longer. This is because the KMA reduces the number of flight paths crossing each other, as well as the total distance travelled. The calculation time is many magnitudes smaller than the time it takes to take off all UAVs. Therefore, the extra time spent on calculations is easily compensated. The safest takeoff procedure is a sequential one, because no UAVs will cross each other (at least for planar formations), and thus collisions are avoided at all times. However, a pure sequential algorithm takes a lot of time, especially when the number of UAVs is large. To keep the risk of collisions at zero, but also to decrease the takeoff time, we implemented a semi-sequential takeoff procedure where at most two UAVs are flying towards their location at the time.

The problem of **in-flight coordination and handling the loss of swarm elements** is solved by extending the MUSCOP protocol. In the new version, the swarm will maintain a stable flight and continue their mission even if multiple UAVs have failed, whereas in the first version, the swarm would be halted indefinitely if even a single UAV failed. This new functionality is made possible by: (i) creating a list of backup masters, so that a failed master can be replaced; and (ii) using timeouts to detect if the UAVs are still active. Through many experiments we have shown that the new version of MUSCOP will, in most cases, not introduce any additional time overhead. In worst case scenarios, only a small additional time overhead exists. This time overhead, which corresponds one-to-one to the timeout value to detect if the UAVs remain alive, can be set by the user, and it has a default value of five seconds. Other experiments have also shown that neither the time overhead nor the stability of the swarm are significantly effected by increasing the swarm size.

8.2 Publications

This section lists the publications that have been produced as a result of this master thesis.

The following works have been published:

- Jamie Wubben, Francisco Fabra, et al. "A novel resilient and reconfigurable swarm management scheme". In: *Computer Networks* 194 (2021), p. 108119. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2021.108119
- Jamie Wubben, Izan Catalán, et al. "Providing resilience to UAV swarms following planned missions". In: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 2020, pp. 1–6. DOI: 10.1109/ICCCN49398.2020.9209634
- Francisco Fabra, Jamie Wubben, et al. "Efficient and coordinated vertical takeoff of UAV swarms". In: *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. 2020, pp. 1–5. DOI: 10.1109/VTC2020-Spring48590.2020.9128488
- Jamie Wubben, Francisco Fabra, et al. "Mecanismo de resiliencia ante la pérdida de elementos en enjambres de VANTs". In: *2021 las Jornadas SARTECO*. 2021, pp. 1–7

8.3 Future work

At the beginning of this chapter we mentioned seven of the most prominent challenges that still have to be solved with respect to UAV swarms. Throughout this thesis we have given solutions for the first four, and thus three challenges still remain unsolved. At the moment we are working on: (v) swarm layout reconfiguration, and (vi) communications and data relaying optimization. The last challenge, (vii) controlled landing, will most likely also be resolved in the future. Currently, the takeoff procedure is semi-sequential, which represents an improvement upon a sequential takeoff, but still requires a lot of time if the swarm is large. In this thesis, we have tested an assignment algorithm that reduces the number of possible collisions. This can make a semi-simultaneous takeoff possible, and that is something that we definitely want to address in the future. Finally, once all the most prominent challenges are solved, it will become important to combine all solutions, and create applications such as search and rescue, precision agriculture, and providing network connectivity.

Acronyms

API Application Programming Interface

ESC Electronic Speed Controller

GCS Ground Control Station

KMA Kuhn-Munkres algorithm

LOS line-of-sight

MAVLink Micro Air Vehicle Link

MUSCOP Mission-based UAV Swarm Coordination Protocol

UAV Unmanned Aerial Vehicle

UAVs Unmanned Aerial Vehicles

UGV Unmanned Ground Vehicles

USV Unmanned Surface Vehicles

UUV Unmanned Underwater Vehicles

VTOL Vertical take off and landing

Bibliography

[1] Goldman Sachs Research. *Drones: Reporting for Work*. https://www.goldmansachs.com/insights/technology-driving-innovation/drones/. Accessed: 2021-05-17. 2014.

[2] BBVA Dory Gascueña. *Drones to stop the COVID-19 epidemic*. https://www.bbva.com/en/drones-to-stop-the-covid-19-epidemic/. Accessed: 2021-05-17. 2020.

[3] H. Shakhatreh et al. "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges". In: *IEEE Access* 7 (2019), pp. 48572–48634. DOI: 10.1109/ACCESS.2019.2909530.

[4] Patrick Vincent and Izhak Rubin. "A Framework and Analysis for Cooperative Search Using UAV Swarms". In: *Proceedings of the 2004 ACM Symposium on Applied Computing*. SAC '04. Nicosia, Cyprus: ACM, 2004, pp. 79–86. ISBN: 1-58113-812-1. DOI: 10.1145/967900.967919. URL: http://doi.acm.org/10.1145/967900.967919.

[5] M. Aljehani and M. Inoue. "Multi-UAV tracking and scanning systems in M2M communication for disaster response". In: *2016 IEEE 5th Global Conference on Consumer Electronics*. Oct. 2016, pp. 1–2. DOI: 10.1109/GCCE.2016.7800524.

[6] Francisco Fabra et al. "ArduSim: Accurate and real-time multicopter simulation". In: *Simulation Modelling Practice and Theory* 87 (July 2018). DOI: 10.1016/j.simpat.2018.06.009.

[7] Anam Tahir et al. "Swarms of Unmanned Aerial Vehicles — A Survey". In: *Journal of Industrial Information Integration* 16 (2019), p. 100106. ISSN: 2452-414X. DOI: https://doi.org/10.1016/j.jii.2019.100106. URL: https://www.sciencedirect.com/science/article/pii/S2452414X18300086.

[8] Yueyan Zhi et al. "Security and Privacy Issues of UAV: A Survey". In: *Mobile Networks and Applications* 25 (Feb. 2020). DOI: 10.1007/s11036-018-1193-x.

[9] Amir Mirzaeinia, Savannah Bradley, and Mostafa Hassanalian. "Drone-station matching in smart cities through Hungarian algorithm: power minimization and management". In: *AIAA Propulsion and Energy 2019 Forum*. 2019, p. 4151.

[10] J. Pestana et al. "A Vision-based Quadrotor Swarm for the participation in the 2013 International Micro Air Vehicle Competition". In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. May 2014, pp. 617–622.

[11] Jun-Woo Cho and Jae-Hyun Kim. "Performance comparison of heuristic algorithms for UAV deployment with low power consumption". In: *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE. 2018, pp. 1067–1069.

[12] Nicolas Dousse, Grégoire Heitz, and Dario Floreano. "Extension of a ground control interface for swarms of Small Drones". In: *Artificial Life and Robotics* 21.3 (Sept. 2016), pp. 308–316. ISSN: 1614-7456. DOI: 10.1007/s10015-016-0302-9. URL: https://doi.org/10.1007/s10015-016-0302-9.

[13] Victor Casas and Andreas Mitschele-Thiel. "Implementable Self-Organized Flocking Algorithm for UAVs Based on the Emergence of Virtual Roads". In: *Proceedings of the 6th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*. DroNet '20. Toronto, Ontario, Canada: Association for Computing Machinery, 2020. ISBN: 9781450380102. DOI: 10.1145/3396864.3399702. URL: https://doi.org/10.1145/3396864.3399702.

[14] Yazhe Tang et al. "Vision-aided Multi-UAV Autonomous Flocking in GPS-denied Environment". In: *IEEE Transactions on Industrial Electronics* PP (Apr. 2018), pp. 1–1. DOI: 10.1109/TIE.2018.2824766.

[15] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. Version ROS Melodic Morenia. May 23, 2018. URL: https://www.ros.org.

[16] Y. Mulgaonkar, G. Cross, and V. Kumar. "Design of small, safe and robust quadrotor swarms". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. May 2015, pp. 2208–2215.

[17] Ming Chen et al. "SIDR: A Swarm Intelligence-based Damage-Resilient Mechanism for UAV Swarm Networks". In: *IEEE Access* PP (Apr. 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.2989614.

[18] Gabriel Hoffmann et al. “Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment”. In: (Aug. 2007). DOI: 10.2514/6.2007-6461.

[19] Bart Theys et al. “Influence of propeller configuration on propulsion system efficiency of multi-rotor Unmanned Aerial Vehicles”. In: June 2016, pp. 195–201. DOI: 10.1109/ICUAS.2016.7502520.

[20] elprocus. *What is Electronic Speed Control (ESC) and Its Working*. https://www.elprocus.com/electronic-speed-control-esc-working-applications/. Accessed: 2021-05-21. 2020.

[21] European Union Aviation Safety Agency (EASA). *Civil drones (Unmanned aircraft)*. https://www.easa.europa.eu/domains/civil-drones-rpas. Accessed: 2021-05-21. 2020.

[22] PX4 autopilot. *Pixhawk 4*. https://docs.px4.io/master/en/flight_controller/pixhawk4.html. Accessed: 2021-05-19. 2020.

[23] ArduPilot Dev Team. *Introducing Copter*. https://ardupilot.org/copter/docs/introduction.html. Accessed: 2021-05-21. 2020.

[24] L. Meier. *MAVLink Micro Air Vehicle Communication Protocol*. URL: http://qgroundcontrol.org/mavlink (visited on 03/26/2019).

[25] GRCDev. *ArduSim: Accurate and real-time multi-UAV simulation*. https://github.com/GRCDEV/ArduSim. Accessed: 2021-05-26. 2020.

[26] Ross Arnold, Hiroyuki Yamaguchi, and Toshiyuki Tanaka. “Search and rescue with autonomous flying robots through behavior-based cooperative intelligence”. In: *Journal of International Humanitarian Action* 3 (Dec. 2018). DOI: 10.1186/s41018-018-0045-4.

[27] Panagiotis Radoglou-Grammatikis et al. “A compilation of UAV applications for precision agriculture”. In: *Computer Networks* 172 (2020), p. 107148. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2020.107148. URL: https://www.sciencedirect.com/science/article/pii/S138912862030116X.

[28] Sarra Berrahal et al. “Border surveillance monitoring using Quadcopter UAV-Aided Wireless Sensor Networks”. In: *Journal of Communications Software and Systems* 12 (Mar. 2016), pp. 67–82. DOI: 10.24138/jcomss.v12i1.92.

[29] James Munkres. “Algorithms for the assignment and transportation problems”. In: *Journal of the society for industrial and applied mathematics* 5.1 (1957), pp. 32–38.

[30] Harold W Kuhn. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[31] Francisco Fabra et al. "MUSCOP: Mission-based UAV Swarm Coordination Protocol". In: *IEEE Access* PP (Apr. 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.2987983.

[32] Jamie Wubben et al. "A novel resilient and reconfigurable swarm management scheme". In: *Computer Networks* 194 (2021), p. 108119. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2021.108119.

[33] Jamie Wubben et al. "Providing resilience to UAV swarms following planned missions". In: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 2020, pp. 1–6. DOI: 10.1109/ICCCN49398.2020.9209634.

[34] Francisco Fabra et al. "Efficient and coordinated vertical takeoff of UAV swarms". In: *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. 2020, pp. 1–5. DOI: 10.1109/VTC2020-Spring48590.2020.9128488.

[35] Jamie Wubben et al. "Mecanismo de resiliencia ante la pérdida de elementos en enjambres de VANTs". In: *2021 las Jornadas SARTECO*. 2021, pp. 1–7.