Wubben, J.; Fabra, F.; Tavares De Araujo Cesariny Calafate, CM.; Cano, J.; Manzoni, P. (2021). A novel resilient and reconfigurable swarm management scheme. Computer Networks. 194:1-12. https://doi.org/10.1016/j.comnet.2021.108119

A novel resilient and reconfigurable swarm management scheme

Jamie Wubben[a], Francisco Fabra[a], Carlos T. Calafate[a], Juan-Carlos Cano[a], Pietro Manzoni[a]

[a]*Departament of Computer Engineering (DISCA) Universitat Politècnica de València, Valencia, Spain*

Abstract

As we witness the fast growth of the Unmanned Aerial Vehicles (UAVs) field, new applications and services emerge at a rapid pace. Above all, the interest in groups of UAVs working together (swarms) is gaining momentum. This interest emerges since swarms are able to undertake more sophisticated tasks. Furthermore, they can also increase task performance and/or robustness. However, organizing a multi-UAV flight is not easy, involving challenges in terms of (i) swarm formation definition, (ii) takeoff procedure, (iii) in-flight coordination, (iv) communication between the swarm elements, (v) swarm layout reconfiguration, (vi) handling the loss of swarms elements, and (vii) controlled landing. These and other issues still hold back the mainstream adoption of swarms in sectors such as agriculture, border surveillance, and parcel delivery.

In this work we provide solutions for two of the main critical challenges: (a) swarm layout reconfiguration, and (b) handling the loss of swarm elements. A wide set of experiments were made using our own realistic UAV emulation tool (ArduSim) in order to validate our proposals. The experiments show that the chances of facing collisions during the reconfiguration are greatly reduced even in error-prone scenarios, and that, in many cases, the loss of a UAV is handled seamlessly; otherwise (in the worst-case scenarios) a delay of just a few seconds is introduced. Additionally, this work addresses cases where, due to the lack of proper communication between the swarm elements, a swarm splits up. Experiments show that with our swarm resilience mechanisms those cases are inherently solved by creating autonomous sub-swarms which will then complete their part of the mission independently.

Keywords: UAV swarm, resilience, flight coordination

Email addresses: jwubben@disca.upv.es (Jamie Wubben), frafabco@cam.upv.es (Francisco Fabra), calafate@disca.upv.es (Carlos T. Calafate), jucano@disca.upv.es (Juan-Carlos Cano), pmanzoni@disca.upv.es (Pietro Manzoni)

1. Introduction

Unmanned Aerial Vehicles (UAVs), also known as drones, are being increasingly used by the general public for different applications such as aerial photography and video, topography, entertainment, etc [1]. In addition, UAVs are being adopted for specialized, business-oriented applications such as precision agriculture, border surveillance, parcel delivery, thermal inspections, and even for people monitoring under the COVID-19 pandemic [2, 3]. Although, new useful applications are still created, working with only one UAV presents certain limitations. For instance, a single UAV cannot carry a high load, cannot cover a large area before the batteries are depleted, etc. Therefore, current research endeavors are moving their focus towards the use of groups of UAVs, also called swarms. The spectrum of possibilities arising by having UAVs working collaboratively in a swarm are countless. Not only can a swarm perform the previous applications in parallel, more efficiently, or with more redundancy, but they also give rise to brand-new applications [4, 5]. However, coordinating a swarm is not an easy task, and many new issues arise such as: (i) swarm formation definition, (ii) takeoff procedure [6], (iii) in-flight coordination, (iv) communication between the swarm elements [7], (v) swarm layout reconfiguration, (vi) handling the loss of swarms elements, and (vii) controlled landing.

This document focuses on handling two main issues related to a swarm flight: the loss of UAVs in the swarm, and the reconfiguration of the swarm layout. Handling the loss of a swarm element is of uttermost importance. When using multiple UAVs the probability that one of them will fail during flight increases significantly. If proper countermeasures are not foreseen, the swarm cannot continue performing the specified task, defeating all the earlier mentioned benefits of using a swarm. With proper mechanisms, we are able to detect when a UAV leaves a swarm (e.g. due to a failure), and update the current behaviour such that the other UAVs can continue their task. However, once one (or more) UAVs are no longer in their position in the formation, the necessity for reconfiguring the swarm emerges. This is, however, not the only reason why a reconfiguration might be useful. Consider for instance a search and rescue mission. Typically, at the beginning, the location of the target is unknown, and thus a large area should be covered, meaning that a more disperse formation is optimal. However, upon discovering the item of interest (by one of the UAVs), the swarm should flock together in order to provide a more specific service.

Taking the aforementioned issues into consideration, the first main contribution of this work is enhancing a swarm-based protocol called MUSCOP, that was first presented in [8, 9]. In that version, swarm reconfiguration was not considered at all, and only basic resilience was provided for the swarm leader. Although the swarm leader is the most important UAV in many applications, it is also important to provide resilience for the other UAVs. This is especially true in the case of MUSCOP, as the swarm leader will wait for arrival confirmation of all UAVs at certain checkpoints during the mission. As the experiments prove, the version that we present in this work is able to handle the loss of any swarm element, including cases where multiple UAVs fail at the same time. We

achieve this goal by adding additional logic to the MUSCOP protocol. Due to this extension, this proposal is also able to handle swarm split-up cases. Swarm split-ups occur when a group of UAVs is unable to communicate with another group of UAVs. In this work, the two groups will both form their own autonomous (sub)swarm. Lastly, we complete this work by adding the capability of reconfiguring a swarm mid-flight and in a safe manner, so as to minimize the chances of collision.

We tested our protocol extensively considering both robustness and scalability issues, and relying on our own realistic multi-UAV emulator called ArduSim [10]. The results show that the loss of a swarm element is handled seamlessly in nearly all cases, with only a small delay of a few seconds in worst-case scenarios. Other experiments show that our protocol is scalable since the overhead (which only occurs in worst-case scenarios) is independent of the number of UAVs. Furthermore, although a totally collision-free reconfiguration is not guaranteed, we also show that our algorithm greatly reduces the chances of collision.

The rest of this work is organized as follows: in Section 2 we provide an overview of related works on this topic. In Section 3 we offer a description of our own realistic simulator/emulator, used to develop and test the current work. Later, in Section 4, we describe the already developed protocol called MUSCOP, which forms the basis of this work. This is followed in Section 5 by our new approach, later referred to as RR-MUSCOP (Resilient Reconfigurable MUSCOP), which makes the MUSCOP protocol resilient to the loss of swarm elements. Our solution is then validated by a set of experiments which are described and evaluated in Section 6. Section 7 concludes this work, including some constructive criticism, and ideas for future works.

2. Related work

The specific topic of handling the loss of a swarm element has only been addressed by a reduced number of authors. However, the research into generic swarms is much more common. 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]. Just a year later they increased this number to 500 UAVs. A similar show was held by the Chinese company EHANG in December 2017[2], this time with 1.180 UAVs. Later, in July 2018, Intel showed the world a UAV-based light show with an astonishing amount of 2.018 UAVs, and with that they broke the Guinness world record[3]. While those light shows are highly entertaining, they are managed centrally, and very strict deployment conditions were enforced to guarantee success. This causes them to not be flexible enough to adapt to any number of UAVs, under any conditions, and for

[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://newsroom.intel.com/news/intel-breaks-guinness-world-records-title-drone-light-shows-celebration-50th-anniversary/

any swarm layout, while keeping computational overhead to a minimum. They are, however, not without value, as they showed the world (and not only the scientific community) what UAV swarms are capable of.

Focusing more on scientific research, we can take a look at the work by Pestana et al. [11]. They presented a modular multi-robot swarm architecture, where each swarm agent consists of an AR Drone 2.0 quadrotor connected to a laptop which runs the software architecture. Their approach relies on the Robot Operating System (ROS) software framework. This makes their work available for a great audience, since ROS makes code sharing and module reuse easy; also, the AR Drone 2.0 is readily available on the market. 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.

Mulgaonkar et al. [12] tested the performance of micro quadcopter swarms in tight/dense formations. The drones also demonstrated to be robust to collisions at velocities of 4 m/s. While we acknowledge the advantages of micro UAVs, we also believe that, due to their low mass, they are usually unable to withstand the natural elements in an outdoor environment. This makes the small, inexpensive and agile micro UAVs only useful for indoor applications, whereas we tend to focus on outdoor applications.

The work by Dano et al. [13] specifically addresses swarm resilience. They provide resilience from a systems engineering point of view. Their work focuses on a higher abstraction layer than ours, and addresses resilience as a whole, whereas we provide a specific solution for the loss of swarm elements. They implemented contract-based design invariant contracts in a Matlab-based flight simulator to quantify the defined location resilience metrics of their system.

Regarding the particular topic of flight configurations, it has been investigated by different authors. The work by V.T. Hoang et al. [14] presents an algorithm to reconfigure a UAV swarm based on the angle-encoded Particle Swarm Optimization (PSO), and a visual inspection of the infrastructure. They begin with a 3D representation of the surface to be inspected, and a set of intermediate waypoints, based on the assumption that an optimal path is produced by using the θ-PSO path planning algorithm; new constraints are proposed to decrease the chances of collision, and to increase task performance. Their work differs from ours as they solely rely on a limited number of reconfigurations. In particular, they only focus on alignment, rotation and shrinkage, while our proposal is able to change the entire topology of the formation.

Recently, Chen et al. published a paper [15] 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-resilient 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.

Furthermore, it is important to keep the usability of our research in mind. Tahir et al. [16] wrote a survey, and a substantial part was dedicated to review the public awareness of drone technology. To that end they performed a questionnaire; their results show that there is only a moderate knowledge about drones, even though the participants were exclusively academics. The results also showed that the participants were still hesitant to use drone technology in their activities/business due to security concerns. However, many participants also acknowledged the possible benefits UAVs can offer.

Traditionally, UAVs are controlled with a remote joystick controller. This is, however, not very intuitive and pilots need some training before they can confidently use the UAV. The complexity also rises when a swarm is used. Therefore, new interfaces are currently being studied. Tezza et al. [17] presented a survey on the state-of-the-art human-drone interaction. They state that, depending on the application and its level of autonomy, humans play different roles when interacting with drone systems. Those roles can be: active controller (e.g. drone racing), recipient (e.g. package delivery), social companions (e.g. Joggobot [18]), or supervisors (e.g. monitoring an autonomous inspection). They proceed by reviewing natural user interfaces such as: gesture, speech, gaze, touch, and even brain-computer interfaces. A similar survey, but studying different works, is provided by [19].

Finally, for any kind of UAV swarm applications, the UAVs have to be able to communicate with each other. This can be particularly difficult because UAVs can move in a 3D space and with a high velocity. For that reason, traditional routing protocols are not always suitable. Networks constrained by those characteristics (3D mobility, high velocity, uncertain connectivity, etc.) are called Flying Ad-hoc Networks (FANETs). Bujari et al. [20] published a work were they go in depth of what FANETs are, how they differ from Mobile Ad-Hoc Networks, and what sort of routing problems exists in FANETs. They then proceed by describing and comparing various state-of-the-art 3D routing algorithms. In an earlier work of Bujari et al. [21] they presented numerous application scenarios, and investigated how the mobility of the nodes impacts the performance of routing algorithms. As stated earlier, nodes in FANETs move in 3D space with a high velocity. However, this movement pattern is seldom random. In their work they state that, although random mobility is easy to simulate, it represents unrealistic flying movements. Therefore, they introduce other, more realistic, mobility patterns. For each pattern they go over the advantages, disadvantages and for what type of application it would fit the best.

Our work differs from the former ones as we specifically focus on the loss of any number of swarm elements, no matter what their role, and on the reconfiguration of a swarm into any new desired formation. We provide actual implementations (that run both in our simulated UAVs using ArduSim, and also in real UAVs being deployed), and discuss several swarm split-up scenarios,

a topic which is not addressed by the aforementioned research works.

3. ArduSim simulator: an overview

The presented protocol is developed and tested with the use of our own multi-UAV flight simulator/emulator called ArduSim [22]. It is available online [10] under the Apache License 2.0. The simulator has many features, being these and its inner workings explained in [22]. In this work we will briefly highlight its key characteristics, and provide a quick overview of its user interface.

ArduSim's key characteristics are:

- Protocol deployment: During the development of ArduSim, special attention was paid in order to ensure a fast and reliable deployment on real UAVs. To accomplish this, ArduSim (as a simulator) uses the same protocols and standards as real UAVs would use. Almost all the UAVs available on the market use the MAVLink communications protocol [23]. This lightweight protocol uses a modern hybrid publish-subscribe and point-to-point design pattern in order to facilitate communication. ArduSim also uses this protocol, and by merely adding a Raspberry Pi (a single board computer) to the UAV, and connecting it to the telemetry port of the flight controller, communication can be established. In order to make the deployment straightforward, all this is abstracted inside the core of ArduSim.

- Scalability: ArduSim was designed to be a multi-UAV flight simulator. Therefore, we put a lot of effort into making it scalable to a large number of UAVs. Resulting in a simulator that is able to run up to 100 UAVs in near real time, and up to 256 UAVs in soft real time on a high-end PC (Intel Core i7-7700, 32 GB RAM); higher number of UAVs can easily be supported through a cluster of machines.

- UAV-to-UAV communication: ArduSim uses the 802.11a standard to communicate, both between UAVs themselves, and between UAVs and the ground station. When ArduSim is used as a simulator, communication is accomplished with virtual links. Whenever protocols are thoroughly tested, they can be deployed on real UAVs. In this case, ArduSim will send messages via User Datagram Protocol (UDP) broadcasts.

- API: Many different protocols need to use similar basic behaviour like: taking off, landing, communicating between UAVs, etc. Ardusim gives access to this behaviour through an Application Programming Interface (API) in order to facilitate a faster protocol development.

- Data logging: ArduSim extensively logs data in various formats after a flight, to make it development and debug friendly.

The main interface of ArduSim is shown in Figure 1. It consists of three major parts. There is an area (1) where the movement and the mission of the

UAVs (if applicable) are shown. In this same area there are some controls to move around the map. Starting and stopping the experiment is done in area (2). Here the user is also able to ask for more information about the state of the UAVs (with button "Show progress"). Finally, there is a region (3) displaying the messages generated by the protocol under development.

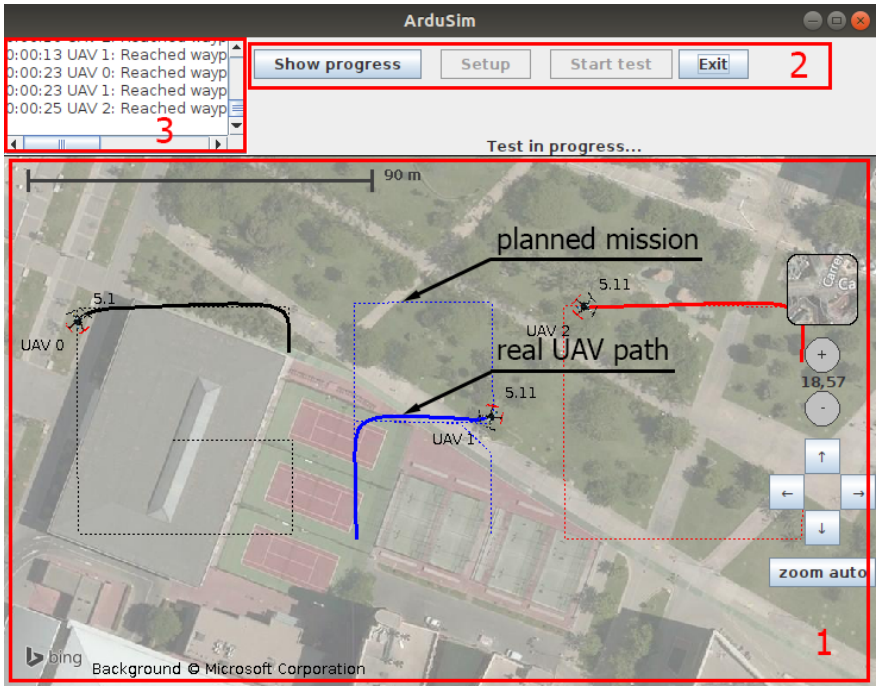


Figure 1: ArduSim user interface.

4. The MUSCOP protocol

Since this work is an extension of the MUSCOP protocol, we will first provide an overview of this protocol. However, a more in depth explanation can be found in [8]. The objective of the MUSCOP protocol is to maintain a stable flight formation when a swarm of UAVs follows a preplanned mission. The protocol uses a master-slave model to synchronize the swarm at each waypoint. Only when all the UAVs arrive at a waypoint does the master issue the command to go to the next waypoint. In between the waypoints, all the UAVs are following their own mission. This mission is calculated before taking off, and it is a modification of the original mission defined by the user. The modification takes the relative position in the swarm into account. Throughout the flight, messages are broadcasted periodically by both master and slaves. In order to simultaneously send and receive messages, two threads are used: the

Talker Thread and the Listener Thread. The Talker Thread sends the messages periodically, and there are a few different types of messages:

- **Hello:** This message is sent from slave to master in the setup phase so that the total number of UAVs can be determined.
- **Data:** This message is sent from the master to the slaves. It contains the original mission, as well as information such as position in the formation, for the slaves to calculate their own mission.
- **DataAck:** Acknowledgement messages are used so that the master knows the slave received a certain packet.
- **ReadyToFly:** Once all the slaves have received their data message and acknowledged it, the master will start sending ReadyToFly messages. These messages will prepare the slaves to take off.
- **ReadyToFlyAck:** The previous message is acknowledged by this message.
- **ReachWaypointAck:** Once all the UAVs have taken off and have reached the first waypoint, the slaves will start sending this message. The message contains the number of the last waypoint that has been reached.
- **MoveToWaypoint:** The master listens and waits until all the UAVs have reached the next waypoint. When this happens, the master will start sending messages of the *MoveToWaypoint* type. When a slave receives these messages, it will start flying to the next waypoint.
- **Land:** Once the last waypoint has been reached, the master will send land messages. These messages include a location such that the slaves are able to calculate their own landing location. During the landing phase, the slaves will move towards the master so that they all land in a compact manner.

The listed messages are received by the Listener Thread of all the other UAVs within range. This information, along with the location of the UAV, is used in order to determine what the UAV should do next: move to the next waypoint, wait at a waypoint, or land.

5. Proposed resilience mechanisms

As stated before, we extend our MUSCOP protocol with two new resilience mechanisms. The MUSCOP protocol, as presented in Section 4, was rudimental and imposes a constant flight formation during the mission. Besides that, the loss of any UAV in the swarm will halt the entire swarm mission. This occurs because, if the master fails, the message *MoveToWaypoint* will never be sent, causing slaves to wait indefinitely. Hence, it assumes perfect operations at all times. Since the master is the most important UAV in a swarm, we first started

working on its resilience. A preliminary attempt was made in our previous work [9]. Although this improved the robustness of the MUSCOP protocol, there was still no way to recover from a failing slave. Therefore, in this work we start by extending MUSCOP again, this time providing resilience for any (and multiple) failing swarm elements. At the same time, this new extension allows for a swarm split-up. Afterwards, we disburden MUSCOP from the constant flight formation by providing a computationally efficient reconfiguration scheme. And thus, the new version of MUSCOP, referred to as RR-MUSCOP provides Resilience and Reconfiguration to the swarm.

5.1. Resilience for failing UAVs

We use the expression *a UAV fails* in order to describe all the situations where a UAV is no longer able to communicate with the rest of the swarm members. Keep in mind that this does not necessarily mean that the UAV is no longer able to fly, although this situation may also occur. In the case of this work, we use it to state that a UAV is not able to receive and/or send protocol messages mentioned in Section 4. A UAV can fail for many reasons: the batteries could be depleted, the wireless antenna could be broken or disconnected due to mobility, a bug in the code could impede the sending and/or receiving of messages, or it could simply be too far away from the other UAVs (outside their radio range). In particular, the latter issue could cause the swarm to split up when radio range limitations cause UAVs to form independent clusters, an issue that will be addressed in section 5.1.3.

Our novel solution consists of two parts: before taking off it is decided who will be the master, and who will be the backup masters. During flight, we work with timeouts to check if a UAV is still alive. If a UAV fails during the flight, that UAV will be excluded from the list of masters. If that UAV was the current master, the following UAV in the list of masters will become the new master. This will be done at every waypoint, and by each UAV individually.

5.1.1. List of masters

Before taking off, an ordered list of masters is created that encompasses all UAVs. We include every UAV of the swarm in that list since, theoretically, all the UAVs could potentially become a master in extreme conditions, although in practice it is very rare that the entire swarm would fail during a mission. The list of masters simply consists of the *IDs* of the UAVs in a specific order. This order could be random, or in order of ID (high to low). However, we decided that there is an optimal ordering, as we now detail. Since in the RR-MUSCOP protocol it is necessary that the slaves can communicate with the master, and vice versa, it is in our best interest to choose the master so that it is located at a strategic position in terms of radio range, i.e. in the center of the formation. Fortunately, when developing a protocol to let UAVs take off safely, we needed the opposite: a list starting with the UAVs furthest away from the center [6]. Since RR-MUSCOP uses the safe takeoff feature while taking off, we can reuse this list so that no extra calculation, apart from reversing the list, needs to be done.

Algorithm 1 is the proposed approach to generate this list. Basically, it consists of the following four steps:

1. Find a central location with respect to the UAVs deployed on the ground (before takeoff).
2. Calculate the euclidean distances from that central location to the positions in the flight formation.
3. Sort this list in descending order.
4. Assign each location in the flight formation to the closest UAV on the ground, in descending order, given that evaluated distance.

This algorithm has a computational cost that grows with $O(n^2)$ (where n equals the number of UAVs). Nevertheless, it is feasible to run it on embedded devices with a low computing power (e.g. a Raspberry Pi). When a failure occurs, we consider that such additional delay and resource consumption during the flight should be avoided whenever possible, especially when there are many UAVs in the swarm. Hence, we decided to use the outcome of this algorithm, that is run at takeoff time, to simultaneously create the ordered list of masters, instead of running it separately at a later time.

Algorithm 1 SafeTakeOff(numUAVs, groundLocations, flightFormation)

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

```
centerLocation = mean(groundLocations)
airLocations = f(centerLocation, flightFormation)
airList = (∅, ∅)
for loc in airLocations do
    airList ← (loc, loc.distance(centerLocation))
end for
sort airList in descending distance order
fit = (∅, ∅, ∅)
totalError = MAX_VALUE
for aLocation in airList do
    bestError = MAX_VALUE
    for gLocation in groundLocations do
        error = gLocation.distance(aLocation)^2
        if error < bestError then
            bestError = error
            bestID = gLocation.ID
        end if
    end for
    totalError += bestError
    fit ← (id, groundLocations[bestID], aLocation)
    groundLocations.remove(bestID)
end for
return fit
```

5.1.2. Updating the swarm

Now that we have established who will be the next master in case the current one fails, we need to determine when a UAV is considered to have failed, and how to solve that critical issue.

In order to know when a UAV fails we propose using timeouts. As shown in Algorithm 2, every UAV has a list consisting of the *ids* of the other UAVs, along with a timestamp. That timestamp is updated periodically whenever a message is received from that UAV. At each waypoint that list is checked and, if the time elapsed since the last timestamp exceeds a certain threshold (assigned empirically in later experiments), we assume that the UAV has failed. Therefore, that UAV will be excluded from the list of masters, and if the UAV that failed was the current master, the next one in the list will become the new master. All the actions usually performed by the MUSCOP protocol are, of course, also executed.

5.1.3. Swarm split-up

Since the procedure described above is executed in all the UAVs, there is a possibility that the outcome is not the same for all of them. Most commonly, the UAVs in the swarm will fly close enough to each other so that they can receive all the messages that are broadcasted. However, when flying further apart, this may not be true in some cases, causing clusters to be formed. Still, a swarm could perfectly complete the mission as long as swarm members can communicate with some master, despite there are cases where swarm split up causes clusters to operate separately from the splitting time onwards. To give a better explanation of this scenario, an example of a flight formation is given in Figure 2.

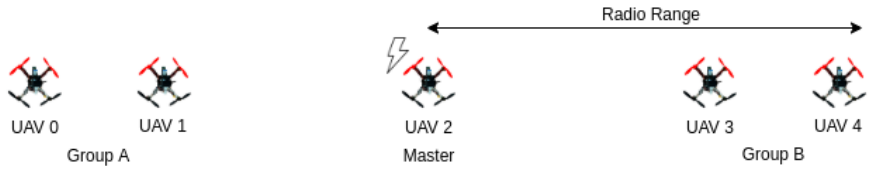


Figure 2: Flight formation example.

In this formation, UAV 2 will be the master, and all the UAVs can communicate with it, hence allowing the mission to be completed without any problem. However, if for some reason the master UAV 2 fails during the flight, UAV 1 will be (in our example) the new master. Due to the large distance, UAVs 3 and 4 are unable to communicate with UAV 1. If the procedure to make a master switch was executed centrally, this situation would cause a problem, resulting in failure of the mission, or an extra time overhead. Therefore, in our solution, all the UAVs take that decision individually. In this case, such an approach will result in two independent swarms to be created, groups A and B, and both of them will continue the mission without interacting with the other.

Algorithm 2 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
```

5.2. Reconfiguration scheme

The reconfiguration of the swarm will start after a triggering event. This could be a user input, or some predefined event in the ground control station. It will take place in two stages: an analysis stage where the calculations are done, and a mobility stage where the UAVs move to their target locations in a intelligent manner to avoid collisions. Both stages and their sub-steps are presented in Figure 3, and detailed in the following paragraphs.

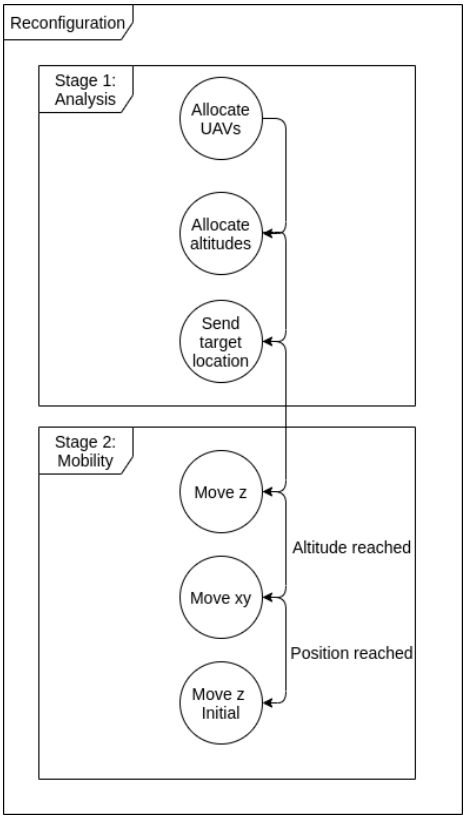


Figure 3: Flowchart of the reconfiguration scheme

5.2.1. Stage 1: Analysis

In the analysis stage the master will first calculate the new locations of the slaves. The idea is that the overall flight distance is minimised by choosing the UAV that is already closest to a new flight position. In order to do so we reuse Algorithm 1, where we substitute the *groundlocation* for the current location of the UAVs, and the *flightformation* for the new (desired) flight formation. In Stage 2 the UAVs will move from the current position to their new position. In order to reduce the chances of collision, the UAVs are divided into sectors based on their direction. Each sector will have a different altitude assigned

to it. In that way, UAVs who were likely to collide will now fly at different altitudes, and thus the risk will be decreased. Implementation details can be seen in Algorithm 3. Once those calculations are done; the master will send the result (i.e. the targetlocation(x,y,Δz)) to the slaves, who in turn will reply with an acknowledge message. Once the master receives the acknowledge message from each slave, the swarm will transition to the mobility stage.

Algorithm 3 Section select procedure.

Require: $numberOfSections > 0$

for *UAV* **in** *UAVs* **do**
  $\Delta x \leftarrow UAV.targetLoc.x - UAV.startLoc.x$
  $\Delta y \leftarrow UAV.targetLoc.y - UAV.startLoc.y$
  $\alpha \leftarrow atan2(\Delta y, \Delta x)$
  if $\alpha < 0$ **then**
    $\alpha = \alpha + 2 \times \pi$
  end if
  $sectorWidth = \frac{2 \times \pi}{numberOfSections}$
  $sector \leftarrow 0$
  for i in range(0, *numberOfSections*) **do**
    $min \leftarrow i \times sectorWidth$
    $max \leftarrow (i + 1) \times sectorWidth$
    if $min \leq \alpha < max$ **then**
      $Sector = i$
    end if
  end for
end for

5.2.2. Stage 2: mobility

The mobility stage is further divided into three steps: first the UAVs will ascend to their new altitude (i: Move_Z), depending on the sector they were assigned, as previously explained. Once the new altitude is reached, they will fly in a straight line towards their target location (ii: Move_XY); finally they will return to their initial altitude (iii: Move_Z_Initial). In each step the master will send messages to the slaves. When a slave receives the message it will perform the movement, and reply with an acknowledgement once the movement is finished. The master receives the acknowledgements and, when all the slaves have sent an acknowledge message (and the master has reached its position), the master will transition to the next state. At that moment, the master will start sending messages advertising its new state; slaves will receive those messages, and transition too. The messages sent by the master only contain an id which represents the current state. They do not have to contain any location information because it was already sent in phase 1.

6. Experimental results

We have performed an extensive amount of experiments, which are divided into two sections. First, experiments were made to validate the resilience scheme of RR-MUSCOP. After those experiments we continued by testing different reconfigurations.

6.1. Resilience for failing UAVs

We started by testing how often a UAV receives messages from another UAV in the swarm based on channel parameters obtained from actual real-life tests [24] (see Section 6.1.1). This experiment allowed us to set the variable *TimeToLive* (see Algorithm 2) to a realistic value. We continued our research by performing multiple experiments where we tested the loss of master(s), slave(s), and different combinations of UAVs lost(see Section 6.1.2). We also performed tests (in Section 6.1.3) with a high number of UAVs that are failing at the same time, and also experimented with swarm split-ups. For all the tests, we measured the time overhead introduced by our protocol. Each experiment, together with the obtained results, is discussed in more detail below.

6.1.1. Message frequency

The talkerThread running on all UAVs sends a message every 200 ms, as explained in Section 4. So, in an ideal environment, a UAV would receive messages with that same frequency. However, since we are using UDP, it is possible that messages are lost. Inside ArduSim we have a realistic model (based on real experiments) for a transmitter based on IEEE 802.11a technology, and using a 5dBi antenna. This model is used to simulate the broadcasting behaviour as accurately as possible. Since our approach heavily depends on the messages received, we wanted to investigate how the message revival rate changes when the distance between the UAVs is increased.

For this experiment we simulated two UAVs flying at different distances from each other. We started at a distance of 2 meters and increased it by 50 meters until communication was no longer possible. After the experiment we calculated how many messages were received per second, and normalized those values. We omitted the messages exchanged during the setup phase, since they are not used to check if a UAV is still alive, and therefore are not relevant for our analysis.

The result is shown in Figure 4, where we can see that the UAVs can communicate in a range between zero and 1400 meters. As expected, the percentage of messages received drops w.r.t. the distance between UAVs. At a distance of 850 meters, almost half of the messages are lost on average. For that reason, it is important that we keep the parameter *TimeToLive* large enough. We want to avoid that a UAV is assumed to have failed when it was actually a mere false-negative event. The only reason why we would like a low value for the *TimeToLive* parameter is that, in the worst-case scenario, a UAV could fail just when arriving to a waypoint. This would mean that the whole swarm would have to wait for the entire timeout period to elapse prior to continuing with the mission, hence causing an unwanted delay. Since, at this moment, there is no

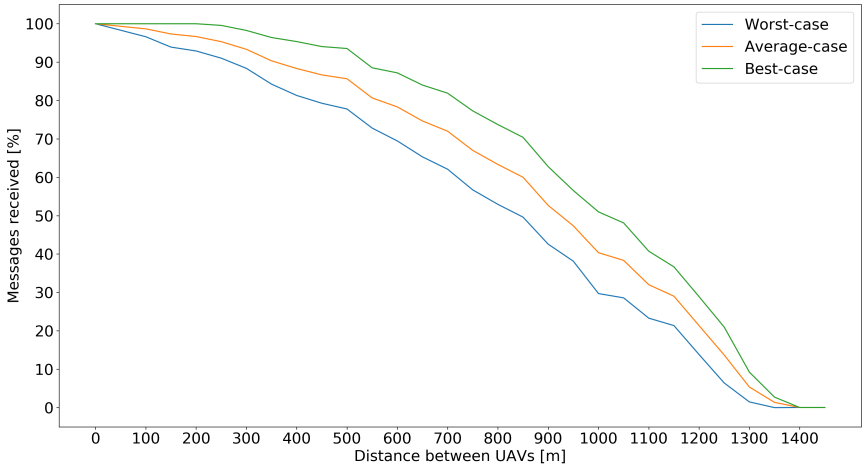


Figure 4: Message received percentage w.r.t. distance between UAVs.

possibility for a UAV to reenter the swarm, we tune our scheme so as to prioritize reducing any false-negative events. So, we decided to set the *TimeToLive* parameter to 5 seconds, and thereby we virtually remove all false-negative cases, as only UAVs flying at distances above 1250 meters from the master could face such a problem. Furthermore, we believe that, in the unlikely event of a UAV failing exactly at a waypoint, introducing a delay of 5 seconds is a small price to pay when considering the mission as a whole.

6.1.2. Handling UAV loss

To test if our protocol works in a wide range of cases, we have considered multiple experiments. In each experiment we have measured how long a UAV stays at a waypoint. Then, we compared this scenario to a flight scenario where no UAV fails. The time difference between both provides us the extra time in the experiment associated to handling UAV failures. In all experiments, four UAVs are flying by following a linear formation (at 10 m/s), with a distance of 50 meters between consecutive UAVs; the *TimeToLive* timer was set to 5 seconds. The target mission (see Figure 5) has four waypoints, where waypoint 0 is right above the takeoff position, and waypoint 3 is at the landing place.

In our experiments, we classify the location where a UAV can fail into three groups: (i) at a waypoint, (ii) just before a waypoint, and (iii) in-between waypoints. We differentiate among these types of events because the exact location where a UAV fails has a direct impact on the overall time overhead. This is because the UAVs only detect if another UAV has failed after a certain timeout has expired. At each waypoint, the swarm has to wait until all the UAVs arrive before they are able to continue their flight. For that reason, if a UAV fails just before reaching a waypoint, the swarm will have to wait for the timeout to expire, causing a longer delay than if that same UAV had failed

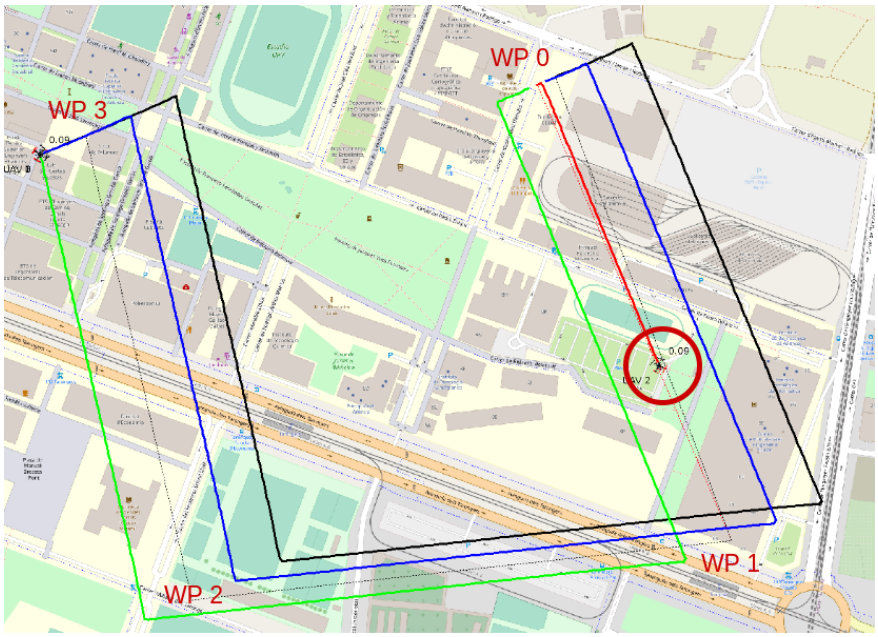


Figure 5: Example of RR-MUSCOP providing resilience, so that the swarm can continue its mission even when a UAV fails.

at another point during the mission. Therefore, we conclude that failing just when arriving to a waypoint is a worst-case scenario. With the same reasoning, failing far away from the waypoint is a best-case situation. In this case, the UAVs will arrive at the waypoint and, since the timer has already expired, the time overhead will be significantly lower than in the former case. For each group we tested the following five different 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.

Table 1 describes the first experiments, where UAVs fail at 200 m from a waypoint. We can observe that there is no extra delay introduced by our protocol, and this is because, during those experiments, the UAVs had enough time to recognize that another UAV had failed. Upon arriving at the waypoint

they can act accordingly, without any extra delay. We can also see that some values are negative; this simply means that, in that case, it was a bit faster than the control case itself, where no UAV failed. Overall the values remain small, and they are caused by ArduSim, and not by our protocol.

Table 1: 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

Table 2 describes the same experiments, but this time the UAVs fail at 15 meters from the waypoint. The delay ranges between 0 and 5 seconds, because the UAVs fail near to the waypoint. We can also observe that the delay is unrelated to which UAV fails (the master or the slave), and that it is also unrelated to how many UAVs fail. Hence, the delay is only related to when (or where) a UAV fails.

Table 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 the last set of experiments (see Table 3) the UAVs fail just when arriving at the waypoint, before actually sending the message that confirms their arrival to that waypoint. Therefore, in this case, the delay is the longest one, and it is equal to the value of *TimeToLive*, which was set to 5 seconds. Also, in this case, we can observe that the delay is unrelated to which UAV fails, or to how many UAVs have failed.

Table 3: 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

From our experiments, we can conclude that the delay depends on the time when a UAV fails, specifically on the pending time to reach the next waypoint i in the mission, represented as t_{wp_i}. As described in Equation 1, the delay will always vary from 0 to *TimeToLive*, which in our case was of 5 seconds.

$$Delay[s] = \begin{cases} TimeToLive - t_{wp_i}, & \text{if } t_{wp_i} \leq TimeToLive \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

6.1.3. Extreme communication conditions

This experiment will focus on scalability. We will compare experiments with a high number of UAVs (100), a moderate number (25), and with a low number of UAVs (4). In those experiments, some of the UAVs will fail 15 meters before a waypoint. The mission that we used had 3 waypoints, the flight speed was set to 10 m/s, and the *TimeToLive* value was set to 5 seconds. We measured the time it takes for UAVs to fly from one waypoint to another (flight time), and the time waiting at the waypoints (wait time). To this purpose, we designed three scenarios:

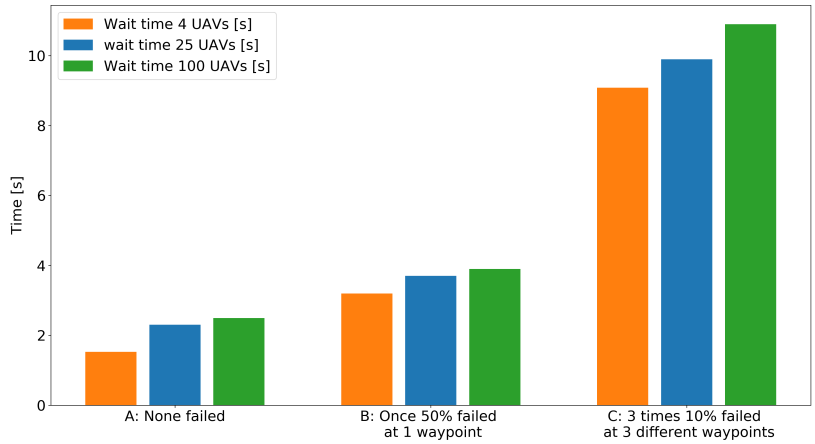
a) A control flight where no UAV fails.

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

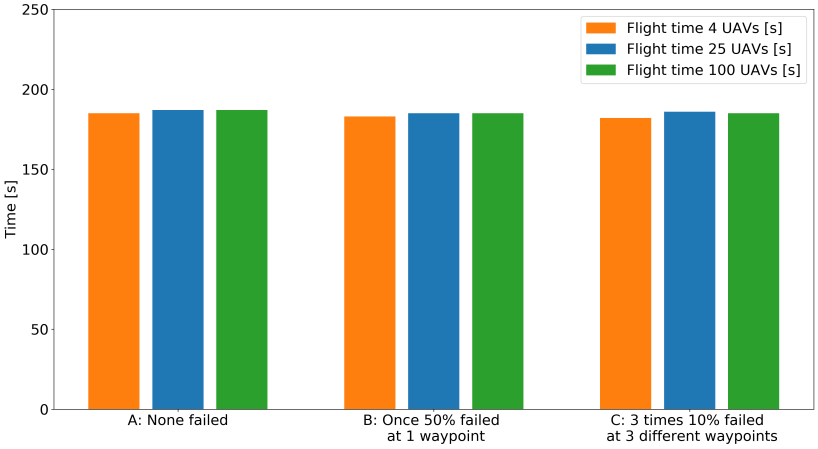
c) A flight where 10% of the UAVs (and the master) fail at consecutive waypoints (3 failures overall).

We believe that those three scenarios are enough to validate the scalability of our approach sufficiently. Scenario *b* is designed to test what happens in the very unlikely case of having many UAVs failing at once, whereas scenario *c* is more realistic, although the fail rate is still high.

The results are shown in Figures 6a and 6b. As shown, 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. The wait time increases slightly due to message buffering. The delay is the largest in scenario c, since three UAV failures occur, and thus we have three times a delay of about 3 seconds. Overall, the delay remains low compared to the total flight time. In addition, the delay is independent of the flight time itself; that is to say, the overall flight time is primarily dependent on the flight distance, while the delay is primarily influenced by where the UAVs fail, and by message buffering in the cases where a high number of UAVs is used. Furthermore, ArduSim can at anytime introduce small and unpredictable delays, as shown in earlier experiments. This figure also confirms some of our previous statements, since one can clearly see 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 6: Time overheads when varying the number of UAVs that fail.

6.1.4. Swarm split-up

As explained in Section 5.1.3, a swarm split-up can occur whenever UAVs are too far away from each other, thus making direct communication impossible. Our protocol has been developed in such a way that all the UAVs take individual decisions about whether or not another UAV is still alive. Therefore, the protocol is inherently able to handle a swarm split-up correctly. In our experiment 13 UAVs flew according to a specific flight formation to force partitioning to occur, as exemplified in Figure 2, which makes a swarm split-up possible. We will provide an overview of the events occurring during the mission:

1. The UAVs are placed close to each other, so that all the UAVs know about the existence of the other UAVs.

2. The UAVs take off and fly to their place in the swarm formation.

3. They reach their place in the swarm formation, and we can visually see (Figure 7a) that different groups (or subswarms) are formed, being the master UAV always in the middle. Notice that the groups are only formed visually; logically, the UAVs still belong to a single swarm.

4. The mission starts, and the UAVs go to the first waypoint.

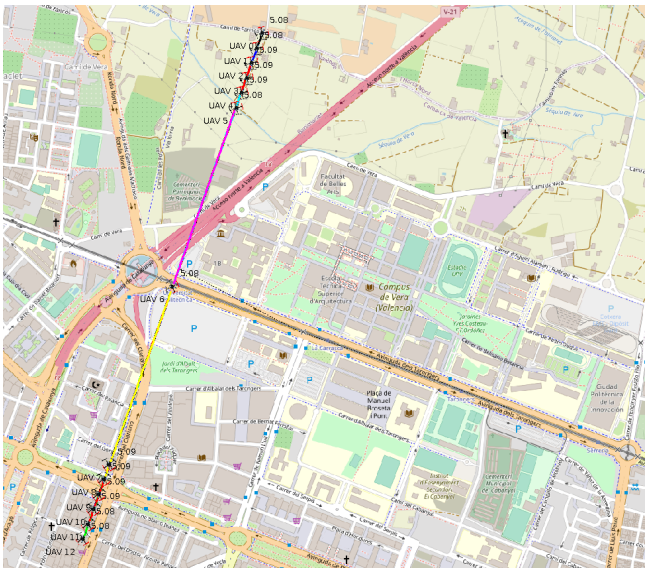
5. Upon reaching the first waypoint, the UAVs inside one group find they are not able to communicate with the UAVs of other groups, and so they consider that all other groups have failed. However, since they are still in contact with their master, there is no need to switch between masters. Hence, they only remove the presumed failed UAVs from the list of potential masters.

6. In-between waypoint one and two, we let the master UAV fail.

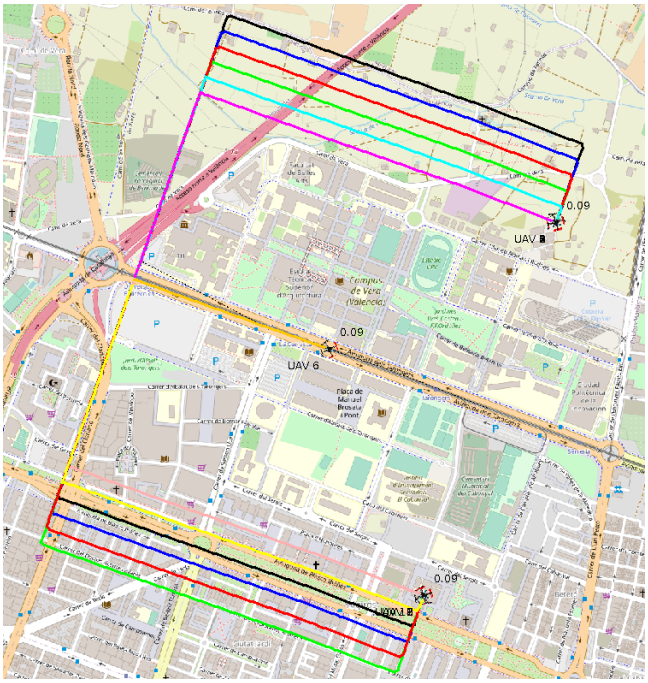
7. Upon reaching the second waypoint, the UAVs notice that the master UAV has failed, an thus a new master will be chosen. This master is different for each group.

8. The mission continues without any problem, and the UAVs reach the last waypoint.

9. Upon reaching the last waypoint, the slave UAVs will move towards their master and land. This means that, after the mission, two groups of UAVs remain with a large distance between them, as shown in Figure 7b. This happens because, in our protocol, we have chosen to land near the current master, rather than picking the original landing position.



(a) Start of the mission.



(b) End of the mission.

Figure 7: Working example of a swarm split-up scenario in ArduSim.

6.2. Swarm Reconfiguration

Following the swarm resilience experiments, we then focused on the proposed reconfiguration scheme. Since our approach is a combination of an intelligent UAV assignment (see Section 5.2.1) and a sectorization procedure (see Section 5.2.2), we started with an experiment which compares our approach against simpler versions. Afterward, we focused on scalability, and devised a formula to estimate the time overhead introduced by our reconfiguration protocol.

6.2.1. Safety analysis

As stated before, our approach is a combination of both an intelligent UAV assignment (Section 5.2.1) and a sectorization procedure that groups UAVs moving in similar directions (Section 5.2.2). We validated the effectiveness of this combined approach, by comparing it to three other (simpler) variants where such mechanisms are not used, or are only partially used. Therefore, we propose to compare our solution (D) to approaches A, B and C:

A. Random position assignment, no altitude change.

B. Random position assignment, different altitudes.

C. Intelligent positioning, no altitude change.

D. Intelligent positioning, different altitudes.

In the following experiments the swarm changes from a linear formation (Figure 8a) towards a mesh formation (Figure 8b). In both formations the minimal distance between the UAVs was set to 10 meters. Furthermore, we used three sectors, and the altitude difference between each sector was set to 5 meters. We have chosen the above mentioned values since, in real experiments, they would provide enough clearance to prevent UAVs from colliding due to wind gusts or GPS errors. During the experiments we measured the minimum distance between the UAVs and the number of potential collisions. We detect a potential collision when the euclidean distance between two UAVs is smaller than 5 meters (i.e. typical GPS error). Furthermore, we measured the time that the UAVs spend in each step of the mobility stage (Section 5.2.2). The results are shown in Table 4 and Table 5. As one might observe from Table 4, changing the formation layout without any precautions is very dangerous and will lead to (multiple) collisions. Simply changing altitude or implementing the intelligent position assignment does decrease the chances of collision, although not sufficiently. A safe distance between the UAVs could only be maintained when both of the mechanisms were used. However, changing the altitude does introduce an additional time overhead. Of course, that overhead depends on the distance travelled, which in this case is the product of the number of sectors and the altitude difference between consecutive sectors.

Table 4: Collisions and minimum distance analysis.

Ex	Nr. collisions	Min. Distance between UAVs
A	4	0.44
B	2	0.33
C	2	3.58
D	0	6.15

Table 5: Time UAVs spend in each state.

Ex	Move z [ms]	Move XY [ms]	Move Z initial [ms]
A	404	13607	400
B	6802	13030	7980
C	380	12425	400
D	8600	12415	8600

6.2.2. Scalability

After obtaining the preliminary results of the previous experiment, we now want to investigate how our reconfiguration scheme behaves for various different formations, and for a higher number of UAVs. The four different swarm formations used in this experiment are shown in Figure 8. Since in every formation the distance between the UAVs is rather small ($\leq$ 10m), all reconfigurations are prone to collisions. We started with 9 UAVs (as in the last experiment), and increased it up to 25. The results are shown in Figure 9.

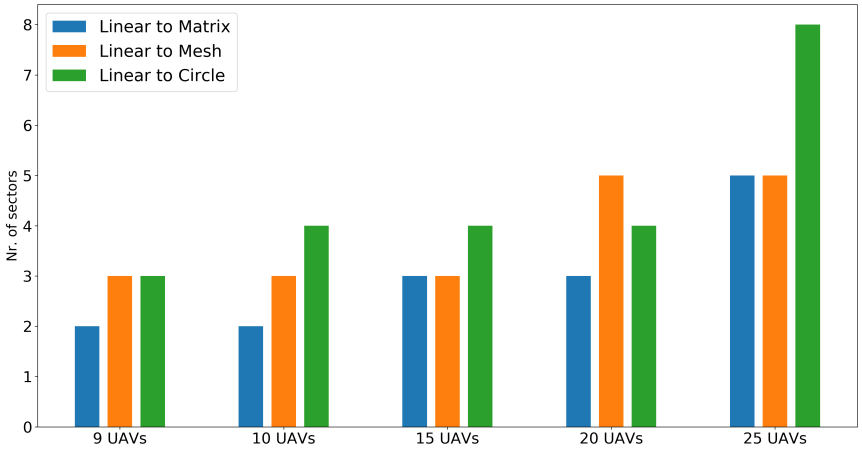
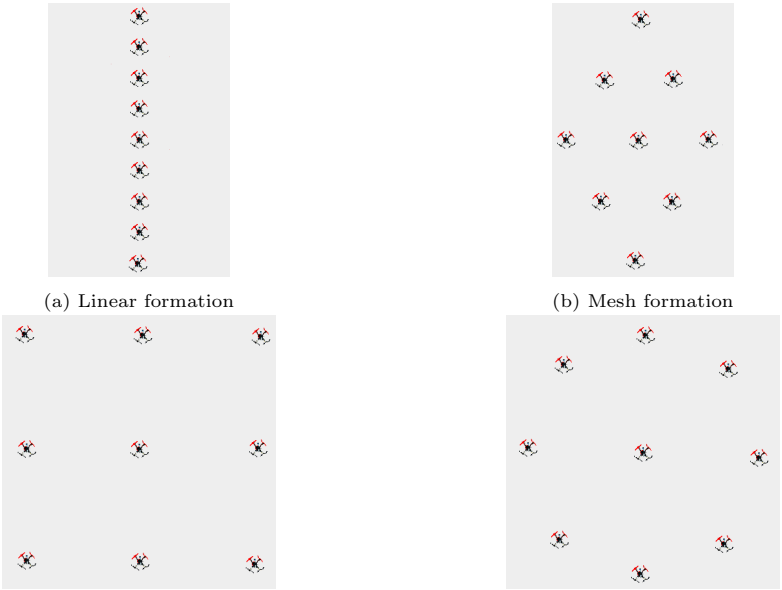


Figure 9: Minimum number of sectors required for a collision-free reconfiguration procedure.

As expected, the number of sectors needed for a collision free reconfiguration increases with the number of UAVs. We can also observe that not all reconfigurations (with the same number of UAVs) are equal. For instance moving from



(a) Linear formation

(b) Mesh formation

(c) Matrix formation

(d) Circle formation

Figure 8: Four different swarm formations

a linear formation to a circle formation requires more sectors. This is because, in certain reconfigurations, the flight path of several UAVs is similar, and thus we need more sectors to safely separate them.

During all the experiments we also measured the time when the UAVs were changing their altitude. We used those measurements to create a formula to estimate the time overhead introduced by our reconfiguration scheme. To achieve this we start by finding the value of the one-way delay T for which:

$$\int_0^T v(t)dt = D \tag{2}$$

where

$$D = num.sectors \times sectors\_offset$$

This one-way delay refers to both upward or downward movements. We can approximate this one-way time overhead T as:

$$T \cong \frac{D}{\hat{v}_{0 \to T}} + \epsilon \tag{3}$$

where $\hat{v}_{0 \to T}$ refers to the expected speed during the entire mobility from time 0 to T, and ϵ accounts for the additional time associated to acceleration and deceleration processes. In our experiments $\hat{v}_{0 \to T}$ was set to 2 m/s, and the distance between the sectors ($sector\_offset$) to 5 meters. Using the results

from the previous experiments, where the number of sectors ranged between 2 and 8, we calculated the value of ϵ, obtaining a delay offset of 1.5s.

While the speed of the UAVs does influence the time overhead directly, it does not alter the chances of a collision. This is because all the UAVs are flying at the same speed, and thus the distance between them will not change.

7. Conclusions and future work

Research in the UAV field has matured over the last years, and as a result the main interest is shifting more and more towards groups of UAVs working together. These so-called swarms are able to extend the UAV-based applications by allowing work to be done in parallel, with more redundancy, and the ability to carry heavier loads.

Coordinating a swarm of UAVs is not an easy task, and so in this work we first focused on providing resilience to swarm protocols. Our solution consists of two parts: a setup phase before taking off, where a list of master and backup masters is created. During the flight a monitoring system checks if all the UAVs are still alive, and handles UAV failures accordingly. Based on the experiments we have performed, we can conclude that our protocol is able to provide resilience against the loss of any swarm element. Hence, not only can the protocol be used in many environments, it also does not matter how many UAVs fail, or what role they had in the swarm. Also, we find that the delay is only dependent on the actual place where the UAV failed. For most real world applications, the delay is found to be negligible, and for worst-case scenarios, it is still bounded to only a few seconds. Our approach is also able to handle swarm split-ups in such a way that a smaller (subset) of the original swarm is still able to work together whenever communication is lost with the rest of the swarm members.

Another focus of this work was the reconfiguration of a swarm mid-flight. Our proposal is based on an intelligent position assignment system that reduces the chances of flight paths crossing. The chances of collisions are further reduced by distributing the UAVs over different altitude levels during the reconfiguration period. Our proposal is computationally efficient, and it can be easily applied to various environments. However, it does not guarantee a collision-free reconfiguration, and the time overhead becomes substantially larger when many UAVs are involved.

Our future work will focus on more complex algorithms that combine path prediction with machine learning approaches to avoid collisions in a more time-efficient manner. Furthermore, we want to introduce the possibility for UAVs to join a swarm during the flight.

8. Acknowledgments

This work was partially supported by the “Ministerio de Ciencia, Innovación y Universidades, Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, Proyectos I+D+I 2018”, Spain, under Grant RTI2018-096384-B-I00.

References

[1] G. S. Research, Drones: Reporting for work, https://www.goldmansachs.com/insights/technology-driving-innovation/drones/, accessed: 2020-06-04 (2014).

[2] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, M. Guizani, Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges, IEEE Access 7 (2019) 48572–48634. doi:10.1109/ACCESS.2019.2909530.

[3] B. Dory Gascueña, Drones to stop the covid-19 epidemic, `https://www.bbva.com/en/drones-to-stop-the-covid-19-epidemic/`, accessed: 2020-06-08 (2020).

[4] J. Zhou, J. Yang, L. Lu, Research on multi-uav networks in disaster emergency communication, IOP Conference Series: Materials Science and Engineering 719 (2020) 012054. doi:10.1088/1757-899X/719/1/012054.

[5] P. Tosato, D. Facinelli, M. Prada, L. Gemma, M. Rossi, D. Brunelli, An autonomous swarm of drones for industrial gas sensing applications, 2019, pp. 1–6. doi:10.1109/WoWMoM.2019.8793043.

[6] F. Fabra, J. Wubben, C. Calafate, J. Cano, P. Manzoni, Efficient and coordinated vertical takeoff of UAV swarms, in: IEEE 91st Vehicular Technology Conference (VTC2020-Spring), 2020.

[7] M. Azari, G. Geraci, A. Garcia-Rodriguez, S. Pollin, Uav-to-uav communications in cellular networks (12 2019).

[8] F. Fabra, W. Zamora, P. Reyes, J. A. Sanguesa, C. T. Calafate, J. Cano, P. Manzoni, MUSCOP: Mission-Based UAV Swarm Coordination Protocol, IEEE Access 8 (2020) 72498–72511.

[9] J. Wubben, I. Catalán, M. Lurbe, F. Fabra, F. J. Martinez, C. T. Calafate, J. C. Cano, P. Manzoni, 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.

[10] ArduSim. accurate and real-time multi-UAV simulation, `https://bitbucket.org/frafabco/ardusim/src/master/`, accessed: 2020-05-11 (2017).

[11] J. Pestana, J. L. Sanchez-Lopez, P. de la Puente, A. Carrio, P. Campoy, 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), 2014, pp. 617–622.

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

[13] E. B. Dano, Resilient system engineering in a multi-UAV system of systems (SoS), ISSE 2019 - 5th IEEE International Symposium on Systems Engineering, Proceedings (2019). doi:10.1109/ISSE46696.2019.8984509.

[14] V. T. Hoang, M. D. Phung, T. H. Dinh, Q. Zhu, Q. P. Ha, Reconfigurable multi-uav formation using angle-encoded pso, in: 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE), 2019, pp. 1670–1675.

[15] M. Chen, H. Wang, C.-Y. Chang, X. Wei, Sidr: A swarm intelligence-based damage-resilient mechanism for uav swarm networks, IEEE Access PP (2020) 1–1. doi:10.1109/ACCESS.2020.2989614.

[16] A. Tahir, J. Böling, M.-H. Haghbayan, H. T. Toivonen, J. Plosila, Swarms of Unmanned Aerial Vehicles — A Survey, Journal of Industrial Information Integration 16 (2019) 100106. doi:https://doi.org/10.1016/j.jii.2019.100106.
URL https://www.sciencedirect.com/science/article/pii/S2452414X18300086

[17] D. Tezza, M. Andujar, The State-of-the-Art of Human-Drone Interaction: A Survey, IEEE Access 7 (2019) 1–1. doi:10.1109/ACCESS.2019.2953900.

[18] E. Graether, F. Mueller, Joggobot: A flying robot as jogging companion (05 2012). doi:10.1145/2212776.2212386.

[19] S. Mirri, C. Prandi, P. Salomoni, Human-Drone Interaction: state of the art, open issues and challenges, 2019, pp. 43–48. doi:10.1145/3341568.3342111.

[20] A. Bujari, C. E. Palazzi, D. Ronzani, A Comparison of Stateless Position-based Packet Routing Algorithms for FANETs, IEEE Transactions on Mobile Computing 17 (11) (2018) 2468–2482. doi:10.1109/TMC.2018.2811490.

[21] A. Bujari, C. Calafate, J.-C. Cano, P. Manzoni, C. Palazzi, D. Ronzani, Flying ad-hoc network application scenarios and mobility models, International Journal of Distributed Sensor Networks 13 (2017) 155014771773819. doi:10.1177/1550147717738192.

[22] F. Fabra, C. T. Calafate, J.-C. Cano, P. Manzoni, ArduSim: Accurate and real-time multicopter simulation, Simulation Modelling Practice and Theory 87 (1) (2018) 170–190. doi:10.1016/j.simpat.2018.06.009.
URL https://doi.org/10.1016/j.simpat.2018.06.009

[23] L. Meier, QGroundControl, MAVLink Micro Air Vehicle Communication Protocol, https://mavlink.io/en/, accessed: 2019-05-11 (2007).

[24] F. Fabra, C. Calafate, J.-C. Cano, P. Manzoni, A methodology for measuring uav-to-uav communications performance, 2017, pp. 280–286. doi:10.1109/CCNC.2017.7983120.