Departamento de Informática de Sistemas y Computadores

Flight coordination solutions for multirotor unmanned aerial vehicles

Thesis submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy in Computer Science

By

Francisco José Fabra Collado

Advisor:

Dr. Carlos Tavares Calafate

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Valencia, Spain

September, 2019

To my wife, mother, and father.

With them my life makes sense.

Do not just teach your children to read, teach them to question what they read, teach them to question everything.

George Carlin

Acknowledgements

First of all, I would like to thank my tutor Carlos Tavares Calafate for his continuous support, providing useful and necessary guidance to reach this goal. In the same way, I want to thank all the professors from the Computer Networks Research Group (GRC), as they have also contributed with both their knowledge and experience.

I also want to thank my coworkers, a great group of Ph.D. students that made me enjoy the whole process, providing hours of fun, inside and outside of the laboratory.

Finally, I want to thank my family and my wife for their support, and for their patience towards me during these years.

Francisco José Fabra Collado
Valencia, October 14, 2019

Abstract

As the popularity and the number of Unmanned Aerial Vehicles (UAVs) increases, new protocols are needed to coordinate them when flying without direct human control, and to avoid that these UAVs collide with each other. Testing such novel protocols on real UAVs is a complex procedure that requires investing much time, money and research efforts. Hence, it becomes necessary to first test the developed solutions using simulation. Unfortunately, existing tools present significant limitations: some of them only simulate accurately the flight behavior of a single UAV, while some other simulators can manage several UAVs simultaneously, but not in real time, thus losing accuracy regarding the mobility pattern of the UAV. In this work we address such problem by introducing Arducopter Simulator (ArduSim), a novel simulation platform that allows controlling in soft real-time the flight and communications of multiple UAVs, being the developed protocols directly portable to real devices. Moreover, ArduSim includes a realistic model for the WiFi communications link between UAVs, which was proposed based on real experiments.

The chances that two UAVs get close to each other during their flights is increasing as more and more of them populate our skies, causing concerns regarding potential collisions. Therefore, this thesis also proposes the Mission Based Collision Avoidance Protocol (MBCAP), a novel UAV collision avoidance protocol applicable to all types of multicopters flying autonomously. It relies on wireless communications in order to detect nearby UAVs, and to negotiate the procedure to avoid any potential collision. Experimental and simulation results demonstrate the validity and effectiveness of the proposed solution, which typically introduces a small overhead in the range of 15 to 42 seconds for each risky situation successfully handled.

The previous solution aims at UAVs performing independent flights, but they can also form a swarm, where more constraints have to be met to avoid collisions among them, and to allow them to complete their task efficiently. Deploying an

UAV swarm instead of a single UAV can provide additional benefits when, for example, cargo carrying requirements exceed the lifting power of a single UAV, or when the deployment of several UAVs simultaneously can accelerate the accomplishment of the mission, and broaden the covered area. To this aim, in this work we present the Mission-based UAV Swarm Coordination Protocol (MUSCOP), a solution that allows multiple UAVs to perfectly coordinate their flight when performing planned missions. Experimental results show that the proposed protocol is able to achieve a high degree of swarm cohesion independently of the flight formation adopted, and even in the presence of very lossy channels, achieving minimal synchronization delays and very low position offsets with regard to the ideal case.

Currently, there are some other scenarios, such as search and rescue operations, where the deployment of manually guided swarms of UAVs can be necessary. In such cases, the pilot's commands are unknown *a priori* (unpredictable), meaning that the UAVs must respond in near real-time to the movements of the leader UAV in order to maintain swarm consistency. Hence, in this thesis we also propose the FollowMe protocol for the coordination of UAVs in a swarm where the swarm leader is controlled by a real pilot, and the other UAVs must follow it in real-time to maintain swarm cohesion. Simulation results show the validity of the proposed swarm coordination protocol, detailing the responsiveness limits of our solution, and finding the minimum distances between UAVs to avoid collisions.

Resumen

A medida que la popularidad de los Vehículos Aéreos No Tripulados (VANTs) se incrementa, también se hacen necesarios nuevos protocolos para coordinarlos en vuelos sin control humano directo, y para evitar que colisionen entre sí. Probar estos nuevos protocolos en VANTs reales es un proceso complejo que requiere invertir mucho tiempo, dinero y esfuerzo investigador. Por lo tanto, es necesario probar en simulación las soluciones previamente implementadas. Lamentablemente, las herramientas actuales tienen importantes limitaciones: algunas simulan con precisión el vuelo de un único VANT, mientras que otros simuladores pueden gestionar varios VANTs simultáneamente aunque no en tiempo real, perdiendo por lo tanto precisión en el patrón de movilidad del VANT. En este trabajo abordamos este problema introduciendo Arducopter Simulator (ArduSim), una nueva plataforma de simulación que permite controlar en tiempo real el vuelo y la comunicación entre múltiples VANTs, permitiendo llevar los protocolos desarrollados a dispositivos reales con facilidad. Además, ArduSim incluye un modelo realista de un enlace de comunicaciones WiFi entre VANTs, el cual está basado en el resultado obtenido de experimentos con VANTs reales.

La posibilidad de que dos VANTs se aproximen entre sí durante el vuelo se incrementa a medida que hay más aeronaves de este tipo surcando los cielos, introduciendo peligro por posibles colisiones. Por ello, esta tesis propone Mission Based Collision Avoidance Protocol (MBCAP), un nuevo protocolo de evitación de colisiones para VANTs aplicable a todo tipo de multicópteros mientras vuelan autónomamente. MBCAP utiliza comunicaciones inalámbricas para detectar VANTs cercanos y para negociar el proceso de evitación de la colisión. Los resultados de simulaciones y experimentos reales demuestran la validez y efectividad de la solución propuesta, que introduce un pequeño aumento del tiempo de vuelo de entre 15 y 42 segundos por cada situación de riesgo correctamente resuelta.

La solución anterior está orientada a VANTs que realizan vuelos independientes, pero también pueden formar un enjambre, donde hay que cumplir más res-

tricciones para evitar que colisionen entre sí, y para que completen la tarea de forma eficiente. Desplegar un enjambre de VANTs en vez de uno solo proporciona beneficios adicionales cuando, por ejemplo, la necesidad de carga excede la capacidad de elevación de un único VANT, o cuando al desplegar varios VANTs simultáneamente se acelera la misión y se cubre un área mayor. Con esta finalidad, en este trabajo presentamos el protocolo Mission-based UAV Swarm Coordination Protocol (MUSCOP), una solución que permite a varios VANTs coordinar perfectamente el vuelo mientras realizan misiones planificadas. Los resultados experimentales muestran que el protocolo propuesto permite al enjambre alcanzar un grado de cohesión elevado independientemente de la formación de vuelo adoptada, e incluso en presencia de un canal de comunicación con muchas pérdidas, consiguiendo retardos en la sincronización insignificantes y desfases mínimos en la posición con respecto al caso ideal.

Actualmente hay otros escenarios, como las operaciones de búsqueda y rescate, donde el despliegue de enjambres de VANTs guiados manualmente puede ser necesario. En estos casos, las órdenes del piloto son desconocidas *a priori* (impredecibles), lo que significa que los VANTs deben responder prácticamente en tiempo real a los movimientos del VANT líder para mantener la consistencia del enjambre. Por ello, en esta tesis proponemos el protocolo FollowMe para la coordinación de VANTs en un enjambre donde el líder es controlado por un piloto, y el resto de VANTs lo siguen en tiempo real para mantener la cohesión del enjambre. Las simulaciones muestran la validez del protocolo de coordinación de enjambres propuesto, detallando los límites de la solución, y definiendo la distancia mínima entre VANTs para evitar colisiones.

Resum

A mesura que la popularitat dels Vehicles Aeris No Tripulats (VANTs) s'incrementa, també es fan necessaris nous protocols per a coordinar-los en vols sense control humà directe, i per a evitar que col·lisionen entre si. Provar aquests nous protocols en VANTs reals és un procés complex que requereix invertir molt de temps, diners i esforç investigador. Per tant, és necessari provar en simulació les solucions prèviament implementades. Lamentablement, les eines actuals tenen importants limitacions: algunes simulen amb precisió el vol d'un únic VANT, mentre que altres simuladors poden gestionar diversos VANTs simultàniament encara que no en temps real, perdent per tant precisió en el patró de mobilitat del VANT. En aquest treball abordem aquest problema introduint Arducopter Simulator (ArduSim), una nova plataforma de simulació que permet controlar en temps real el vol i la comunicació entre múltiples VANTs, permetent portar els protocols desenvolupats a dispositius reals amb facilitat. A més, ArduSim inclou un model realista d'un enllaç de comunicacions WiFi entre VANTs, que està basat en el resultat obtingut d'experiments amb VANTs reals.

La possibilitat que dues VANTs s'aproximen entre si durant el vol s'incrementa a mesura que hi ha més aeronaus d'aquest tipus solcant els cels, introduint perill per possibles col·lisions. Per això, aquesta tesi proposa Mission Based Collision Avoidance Protocol (MBCAP), un nou protocol d'evitació de col·lisions per a VANTs aplicable a tota mena de multicòpters mentre volen autònomament. MBCAP utilitza comunicacions sense fils per a detectar VANTs pròxims i per a negociar el procés d'evitació de la col·lisió. Els resultats de simulacions i experiments reals demostren la validesa i efectivitat de la solució proposada, que introdueix un xicotet augment del temps de vol de entre 15 i 42 segons per cada situació de risc correctament resolta.

La solució anterior està orientada a VANTs que realitzen vols independents, però també poden formar un eixam, on cal complir més restriccions per a evitar que col·lisionen entre si, i perquè completen la tasca de forma eficient. Desplegar un

eixam de VANTs en comptes d'un només proporciona beneficis addicionals quan, per exemple, la necessitat de càrrega excedeix la capacitat d'elevació d'un únic VANT, o quan en desplegar diversos VANTs simultàniament s'accelera la missió i es cobreix una àrea major. Amb aquesta finalitat, en aquest treball presentem el protocol Mission-based UAV Swarm Coordination Protocol (MUSCOP), una solució que permet a diversos VANTs coordinar perfectament el vol mentre realitzen missions planificades. Els resultats experimentals mostren que el protocol proposat permet a l'eixam aconseguir un grau de cohesió elevat independentment de la formació de vol adoptada, i fins i tot en presència d'un canal de comunicació amb moltes pèrdues, aconseguint retards en la sincronització insignificants i desfasaments mínims en la posició respecte al cas ideal.

Actualment hi ha altres escenaris, com les operacions de cerca i rescat, on el desplegament d'eixams de VANTs guiats manualment pot ser necessari. En aquests casos, les ordres del pilot són desconegudes a priori (impredictibles), el que significa que els VANTs han de respondre pràcticament en temps real als moviments del VANT líder per a mantindre la consistència de l'eixam. Per això, en aquesta tesi proposem el protocol FollowMe per a la coordinació de VANTs en un eixam on el líder és controlat per un pilot, i la resta de VANTs ho segueixen en temps real per a mantindre la cohesió de l'eixam. Les simulacions mostren la validesa del protocol de coordinació d'eixams proposat, detallant els límits de la solució, i definint la distància mínima entre VANTs per a evitar col·lisions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Unmanned Aerial Vehicles (UAVs), colloquially known as drones, are flying devices able to perform programmed flights or being remotely controlled. During the past few years they have gained high relevance thanks to their capabilities when performing a wide range of tasks. For instance, planned missions can be defined to supervise farmlands, deliver packages to remote locations, or contribute to create Delay-Tolerant Networks (DTNs) in the scope of Smart Cities [25]. Moreover, by adopting adequate algorithms, it is possible to develop new routing protocols [24], control the flight of a group of UAVs acting as a swarm [59], or dynamically create an aerial network infrastructure in a dynamic, on-demand fashion [55].

These new applications demand for the establishment of communication protocols between UAVs to avoid collisions when they are in close proximity, and to coordinate them when performing complex tasks, such as those undertaken by UAV swarms. The development of these protocols is part of the current research effort in the commercial UAVs field, which nowadays represents a hot topic according to Gartner (see Figure 1.1).

Experimenting with UAV-based networking in order to develop and validate new protocols presents several restrictions including: (i) pilots should meet the regulation requirements of each country, (ii) weather conditions should be favorable, (iii) battery lifetime is quite limited, and (iv) certain applications require testing with a high number of UAVs simultaneously. For instance, Lee et al. [39] analyze a new routing protocol between UAVs and a Ground Control System (GCS) using up to six UAVs simultaneously, while Y. Chai et al. [9] test a UAV formation

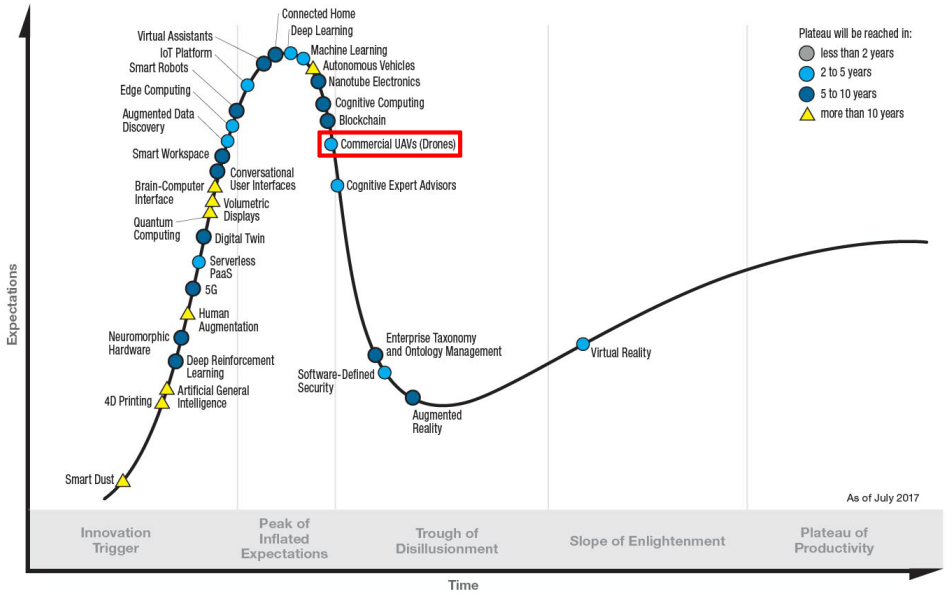


Figure 1.1: Gartner hype cycle for Emerging Technologies, 2017.

protocol with up to six virtual UAVs.

In general, the approach adopted by most researchers relies on simulation. However, simulations should be as realistic as possible, that is, they should account for the physical properties and flight behavior of the aerial vehicle, and they should also integrate a model for wireless communications between UAVs that resembles real-life behavior. In addition, it is important that the simulation environment is able to manage several UAVs simultaneously, and that the code developed is compatible with existing flight controllers, thereby simplifying the process of porting the developed protocols to real UAVs to complete the development cycle. Current flight simulators lack of some of these features, which supposes a handicap for protocol development.

Regarding emerging UAV applications, privacy, security and flight safety [50] remain a concern, especially in urban environments where the consequences of any flight disruption are typically much more severe due to the risks of injuries for citizens. To address this issue, several efforts are taking place worldwide to make UAV flights safer. For instance, in Europe, U-space [48] is an initiative that aims at making UAV traffic management safer and more secure. In particular, U-space attempts to provide an appropriate interface with manned aviation and air traffic control so as to facilitate any kind of routine mission, in all classes of airspace, and even in congested environments like urban areas, so as to achieve the ambitious

Single European Sky (SES) goal. The SESAR Joint Undertaking [60] was set up in order to manage this large scale effort, coordinating and concentrating all European Union (EU) research and development activities focused on Air Traffic Management. This way, a wide range of drone missions that are currently being restricted will be possible thanks to a sustainable and robust European ecosystem that is globally interoperable.

Among the different areas where UAV flight safety is being considered, there is a particular area that has not yet been fully addressed: the development of *Sense & Avoid* mechanisms to enable a UAV to become aware of its environment, allowing it to take evasive action if necessary [51]. There are protocols to avoid collisions between large fixed-wing UAVs, but currently there are no such protocols for multicopter UAVs following independent planned missions.

Collision avoidance is not the only field where coordination protocols have to be developed. There are applications where employing a swarm of UAVs can help to optimize some tasks through cooperation, or to parallelize tasks by supporting the redundancy of different sensors, or with the simultaneous usage of different types of sensors, among other scenarios.

Although there are already some solutions for the automation of UAV swarm flights, in certain situations automatic guidance can be required. Examples of such situations may include applications for large-scale agriculture in search of pests or weeds [19, 2], wild life recordings [5], or border surveillance [13], among others. In these specific cases, the different UAVs that make up the swarm must be coordinated when carrying out the mission. Such mission must be planned beforehand. Then, the communications between UAVs should enable near-real-time responsiveness to maintain the consistency of the swarm.

The reliability of communications is a major problem in the creation of swarms, as UAV synchronization directly depends on the reliability of such communications. Also, the distance separating the different UAVs that integrate the swarm must remain consistent to avoid possible collision problems. Another problem that may be experienced by swarms is associated with the transient or long-term interruption of communication, which hinders synchronization, causing delays to the entire process, or even a reduction of the number of elements in the swarm.

Finally, we focus on applications where UAV guidance must be manual. In this particular case, the different UAVs that make up the swarm have to dynamically adjust their routes in order to follow the master UAV acting as the leader of the swarm. Such a solution may be required in scenarios such as search & rescue [66, 3], fire tracking [65], or the monitoring of disaster areas. In these cases, the pilot must respond to visual stimuli in real time, and adapt the UAV course accordingly. Our focus on UAV swarms also addresses situations where, in addition to manual guidance, there is a need to carry multiple items or sensors that go beyond the lifting capacity of a single UAV. An example would be a rescue scenario where different UAVs carry food, water, medicine, or shelter. Thus, we need solutions that would be very useful in these situations, by allowing the pilot to control

the leader UAV following the usual manual procedures, while seamlessly dragging along the rest of the UAVs conforming the swarm.

1.2 Objectives

The main objective of this thesis is to propose and implement ArduSim, a flight simulation platform for UAVs, designed to provide the research community a tool to develop new coordination protocols that are needed for the new emerging applications in this field. Using ArduSim as a development platform, we then propose three novel protocols: MBCAP, a collision avoidance solution for UAVs following independent missions; MUSCOP, a coordination protocol for swarms following a predefined mission; and FollowMe, a coordination protocol for swarms with manual guidance.

To achieve these global objectives, it becomes necessary to accomplish several specific objectives.

ArduSim:

- Provide a communications link between UAVs, and an embedded system that connects to the flight controller for inter-UAV coordination purposes.
- Develop a tool to measure the wireless communications link quality between UAVs.
- Measure the link quality between UAVs under different circumstances in order to model that link and integrate it in the ArduSim platform.
- Design and implement the simulation platform, providing all the necessary features.
- Perform a thorough study of scalability and performance of the proposed tool.

MBCAP:

- Design the protocol behavior so as to meet the desired collision avoidance goals.
- Implement the proposed protocol in ArduSim.
- Perform basic simulation tests to validate the correctness and performance of the protocol.
- Perform tests with real UAVs to check the similarity between simulation and real experiments.

- Perform large-scale simulation tests for a thorough validation of the correctness of MBCAP.

MUSCOP and FollowMe protocols:

- Design and implement the protocols.
- Perform a formation stability analysis.
- Test the proposed protocols under different formations.
- Evaluate the performance of the protocols in terms of time shift for the different UAVs with respect to the master UAV.

1.3 Structure of the Thesis

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

- **Chapter 2. UAVs and UAV-based systems: An overview**: we provide a review of general aspects related to UAVs architecture and current UAV flight simulators. We finish by detailing current solutions for UAV coordination.
- **Chapter 3. UAVs built for real experiments**: we briefly introduce the custom UAV models built and used on real experiments.
- **Chapter 4. ArduSim simulation platform**: we design, develop and validate our novel simulation platform. Among others, we analyze scalability, and to what extent ArduSim meets real-time requirements.
- **Chapter 5. Mission Based Collision Avoidance Protocol (MBCAP)**: we propose a distributed collision avoidance protocol, finding weaknesses and improving the initial solution with MBCAP-e, an enhanced version that optimizes the flight time required to avoid collisions, and that solves some of the flaws detected on the first version of the protocol.
- **Chapter 6. Mission-based UAV Swarm Coordination Protocol (MUSCOP)**: we present a protocol that is able to maintain a stable flight formation for a UAV swarm that is following a global planned mission.
- **Chapter 7. FollowMe protocol**: we develop a protocol aimed to tasks that require using a UAV swarm with manual guidance, where a leader UAV is guided by a pilot, and the remaining UAVs in the swarm follow the leader, adapting to its unexpected movements dynamically.

- **Chapter 8. Conclusions, Publications and Future Work**: we conclude this thesis, and we present the related publications, as well as a list of future research lines.

Chapter 2

UAVs and UAV-based systems: An overview

Before introducing the current efforts from the research community, in this section we include an overview of the nature and functioning of the UAV types addressed in this thesis.

Currently there are several types of UAVs with different sizes, flight time, load capacity, and price. Figure 2.1 shows a classification of UAVs depending on these factors [21, 14, 68].

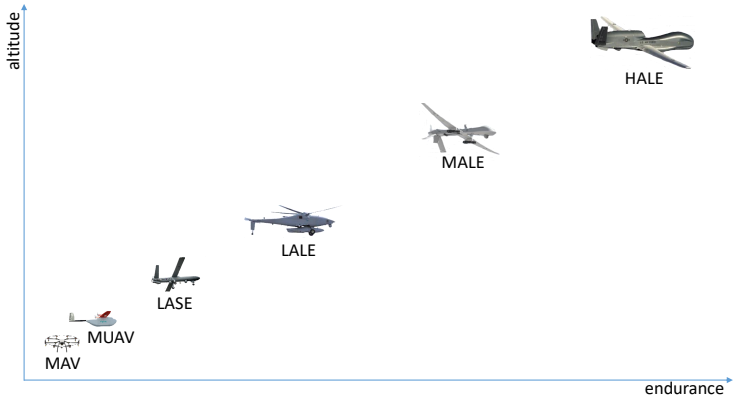


Figure 2.1: Types of UAVs given range and size.

In detail, these are the characteristics of UAVs belonging to each of these categories:

- **Micro Aerial Vehicle (MAV)**. They are the smallest UAVs available, and are mainly used for entertainment and research. With a weight lower than 2 kg and a price typically less than 1,000 euros, they are limited to a flight time below 30 minutes, and a flight altitude lower than 200 meters. The pilot is required to maintain Line Of Sight (LOS) with the device during the flight in many countries.
- **Mini Unmanned Aerial Vehicle (MUAV)**. The weight is increased up to 20 kg, and they are used for surveillance, and data gathering.
- **Low Altitude, Short Endurance (LASE)**. With a limited payload of 20 kg, they can fly for less than 2 hours.
- **Low Altitude, Large Endurance (LALE)**. They are able to carry payloads of several kilograms at an altitude of a few thousand meters for extended periods. They are used for forest inventory, monitoring, etc.
- **Medium Altitude, Large Endurance (MALE)**. They are much larger than low-altitude UAVs, and they can fly for many hours, for hundreds of km, and up to an altitude of 9,000 meters. MALE UAVs cost a hundred thousand euros or even more.
- **High Altitude, Large Endurance (HALE)**. Large-sized UAVs that can fly for more than 30 hours at an altitude up to 20,000 meters, being mainly used for military purposes.

Figure 2.2: Example of a MAV.

Many UAVs that fly at a low altitude belong to the **Vertical Take Off and Landing (VTOL)** family whenever they do not require a take off or landing strip, which makes them specially suitable for many applications. Between VTOL

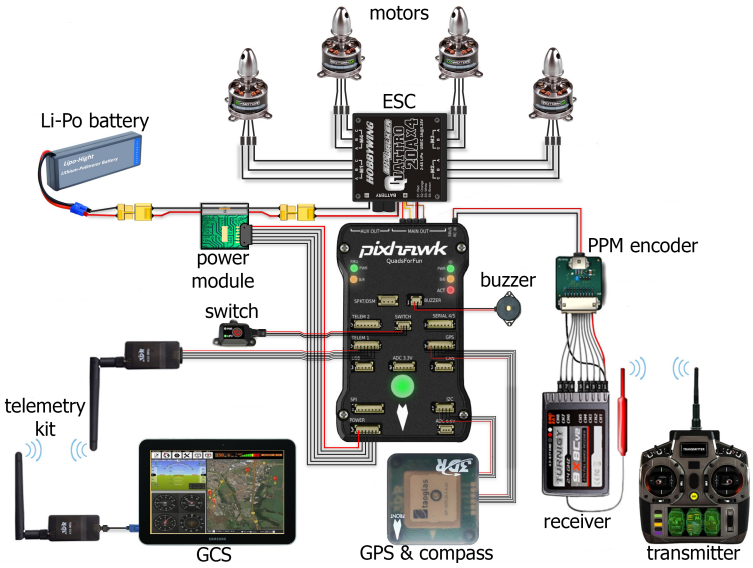


Figure 2.3: Open source multicopter wiring.

UAVs, this work focuses in multirotor UAVs (see Figure 2.2), a type of MAV that is rapidly spreading due to the reduced cost and the capability of flying in a complex area, e.g. urban environment or inside buildings.

We can find many multicopter flight controllers in the market based on proprietary hardware and software [16], but we are specially interested in open source solutions to easily implement and deploy protocols on real multicopters. Most of the open source solutions available in the market are based on the Pixhawk flight controller [46], containing an implementation of the Ardupilot firmware [64]. The wiring scheme of a typical quadcopter with this configuration is shown in Figure 2.3:

- Pixhawk. Flight controller.
- Electronic Speed Controller (ESC). Provides power to the motors and controls their speed individually. The flight controller provides the signal needed to vary the speed.
- Motors. Engines used to move the propellers.
- Power module. It transmits power to the flight controller, and to the ESCs.
- GPS & compass. External unit usually mounted over all other UAV elements.

- Li-Po battery. Lithium-ion Polymer battery used to feed all the UAV systems.
- Remote control kit. The transmitter allows the user to operate the UAV remotely with 8 or more channels mapped to the controls included in it. The signal usually operates in the 2.4 GHz frequency band. The receiver transmits low latency messages to the Pulse Position Modulation (PPM) encoder, that translates the commands from Pulse Width Modulation (PWM) to PPM signals that the Pixhawk can understand. The transmission is bidirectional, so the user is able to read status information in the transmitter screen.
- Radio telemetry kit. It provides the pilot with feedback of the flight in a smartphone, tablet or GCS. Moreover, a GCS allows to control the multicopter through this link. The transmission uses the 433 MHz band in Europe, and the 915 MHz band in USA. The receiver is connected to the first telemetry port of the flight controller with a serial link. This communication link is achieved with the Micro Air Vehicle Link (MAVLink) protocol [47], a *de facto* standard to communicate with open source flight controllers. We found it really important to develop solutions compatible with this protocol, to guarantee a straightforward deployment of any developed solution on currently available UAVs.
- Safety switch. It prevents takeoff from taking place unless it is pressed to ensure the pilot is ready to start the flight.
- Buzzer. Optional element that provides feedback about the current state of the controller.

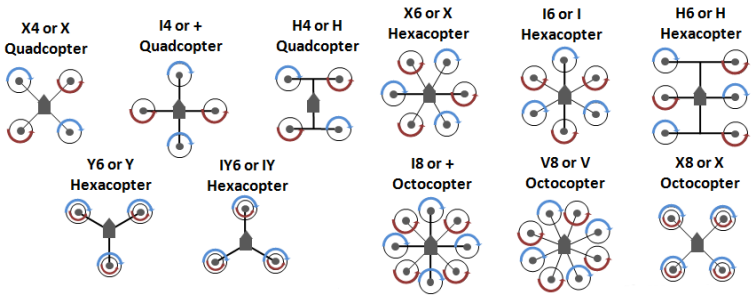


Figure 2.4: Multicopter layouts.

Several layouts are available to configure a multicopter depending on the number of motors and their distribution (see Figure 2.4), typically varying from three

to eight motors, being the tricopter a very unusual model. Quadcopters are very popular due to their low price, but hexacopters and octocopters are more convenient, as they can keep flying even if one motor fails. Propellers alternate spinning directions to achieve torque compensation.

2.1 Flight simulation alternatives

Protocol testing with real multicopters presents several restrictions:

- Pilots should meet the regulation requirements of each country.
- Weather conditions should be favourable.
- Battery lifetime is limited.
- Some applications require a high number of UAVs flying simultaneously.

These restrictions make protocol testing an expensive and time consuming task, reason why researchers rely on simulation to test their developments. There are many flight simulators available, but none of them allows developing new protocols where the UAVs may need to modify their flight depending on the information gathered from other devices (typically other UAVs). Moreover, they have to adequately emulate communications, and also the physic properties of the multicopter.

In [18] we can find a list of existing flight simulators. Several of them mimic with considerable accuracy the characteristics and physical properties of the UAV, although (i) their code is proprietary, (ii) they are only compatible with a few platforms, and (iii) they only allow controlling a single UAV at a time, thereby failing to offer inter-UAV communications support. On the other hand, generic network simulators like The Network Simulator (ns-2), ns-3, and Objective Modular Network Testbed in C++ (OMNeT++) [34, 32] allow simulating with great accuracy the communications link, but they are unable to accurately simulate the physical properties of a UAV, or the UAV flight behavior, in a realistic manner. On the other hand, Ben-Asher et al. [8] created IFAS, a network simulator aimed at Ad-hoc networks; their solution is only oriented at developing routing protocols, failing to provide real-time network simulation.

Other authors have addressed the simulation of multiple UAVs. An example is the work of Richard Garcia et al. [22], where they introduce a simulator based on X-Plane that is able to emulate up to 10 planes or helicopters; however, differently from ours, their solution is not oriented to multicopters. Moreover, their solution requires a PC for each simulated UAV. In [27], J. Holt et al. develop a symbiotic simulation architecture, although it is exclusively focused on the development and analysis of collision avoidance protocols.

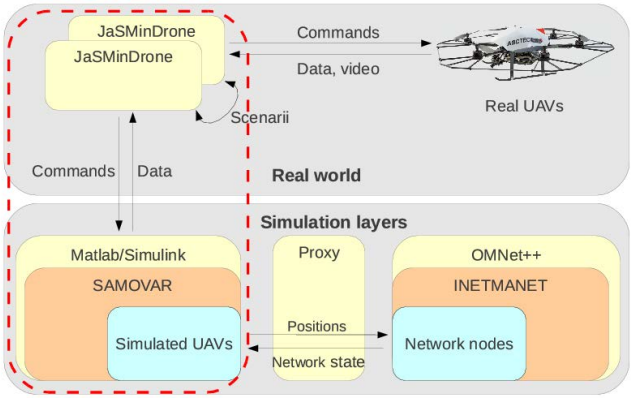


Figure 2.5: AEOTURNOS pilot interface, UAV, and communications network simulation tools coupling architecture.

UAVSim [29] is a simulator focused on securing the communications, and runs over OMNeT++. UAVSim works by extending OMNeT++, introducing a mobility model based on the properties defined for a specific UAV model which allows updating the route of each UAV based on the interactions with other neighboring UAVs.

Simbeeotic [33] is able to simulate with great accuracy an UAV swarm using JBullet, but it relies on its own language to control these virtual UAVs, which difficults bringing the developed protocols to real-world devices. UB-ANC [49] also simulates a set of UAVs, but it fails to model the communications channel, and it does not include a graphical interface to allow analyzing UAV mobility, meaning that all the information analysis depends on interpreting log files and using third-party applications.

We can also find approaches like AETOURNOS [12], based on MATLAB, that attempt to combine, on the long term, the simulation of a custom multicopter model in real-time, or even using a real UAV (see Figure 2.5), with simulated communications using OMNeT++. Its initial development is limited to the use of the Transport Control Protocol (TCP), and authors fail to evaluate the temporal mismatch between real-time UAV simulation and simulation-time communications between UAVs.

In this work we present the Arducopter Simulator (ArduSim) (see chapter 4), a multi-UAV simulation platform where mobility and the communications between many UAVs are simulated in real time, and protocols are directly portable to real devices, thereby avoiding the problems detected in the simulators described above.

2.2 Current solutions for UAV coordination

UAV swarms have been studied for many years, and some coordination protocols have been proposed in the literature. Most works focus on fixed wing UAVs, as commercial devices were released many years ago, but only few studies exist related to multicopter UAVs. In this section we introduce some relevant protocols in this field, with an emphasis on solutions for multicopter UAVs.

2.2.1 Collision avoidance protocols

Regarding collision avoidance protocols, Mahjri et al. [44] did a theoretical study of the characteristics that such a protocol should have, describing its elements. In this work, the authors differentiate between two techniques for collision risk detection: non-cooperative sensors, such as a proximity sensor [61] or a camera [35], and cooperative sensors, such as the dissemination of flight information to nearby UAVs, as occurs with ADS-B [1] technology in the solution proposed by Liu and Foina [42]. In general, non-cooperative sensors can help at avoiding collisions with static objects, but they do not allow to react fast enough to avoid collisions with moving objects, like other UAVs performing independent tasks. In these scenarios, cooperative sensors are more effective, as the collision risk can be detected well in advance.

Jinwu et al. [30] defined a collision risk detection strategy based on space discretization. They assign a degree of danger to each location in the space following a probabilistic model that predicts the place a UAV will be in the future. This work focuses on UAVs moving very fast at a constant speed, and defines a vast protected area around the UAV, forcing other aircrafts to scatter over a wide area to avoid collisions. Furthermore, the authors did not explain the collision avoidance strategy used to change the direction of the UAVs during flight.

Lin et al. [41] presented a UAV collision avoidance solution which can achieve cluster situational awareness, autonomous formation control, and intelligent collaborative decision making. The main idea of their algorithm is to consider all swarm members as a whole, and control the internal and external parameters of the UAV swarm separately. Among the UAVs, a communication topology is set up. They use a consensus algorithm to maintain the formation and avoid collisions between UAVs. Moreover, they use the weight coefficient to set the priority for every UAV. Beyond single UAV control, an improved artificial potential field method is adopted to control swarm mobility. They improved the safety distance and the traditional artificial potential approach to make them more suitable for the UAV collision avoidance task. This way, even though UAVs approach obstacles at a high speed and with a small angle, they will still have enough time and space to change their flight direction. Authors validated the effectiveness of their cooperative obstacle avoidance algorithm using MATLAB alone.

Zhou et al. [70] presented a trajectory planning strategy for UAV collision avoidance. They propose a varying cells strategy to integrate aerodynamic constraints into trajectory planning. They also adapt basic avoidance actions in the varying cells strategy to go through different cells, enabling more flexible avoidance maneuvers. Authors used Monte Carlo simulations to demonstrate that the proposed method satisfies aerodynamic constraints, while both the convergence and collision avoidance rates improve.

Kim and Ben-Othman [37] introduced a surveillance model for multi-domain IoT environments, which is supported by reinforced barriers with collision-avoidance using heterogeneous smart UAVs. Formally, they define a problem whose goal is minimizing the movement of smart UAVs having as a condition that the collision-avoidance among UAVs is assured when flying between their initial positions and specific spots in a limited area.

Wang et al. [67] proposed an approach based on a 2D Laser Imaging Detection and Ranging (LIDAR) that offers a method to represent the objects in the environment in a compact manner, which is significantly more efficient in terms of both memory and computation in comparison with similar previous proposals. Their approach is also capable of classifying objects into categories such as static and dynamic, and tracking dynamic objects, as well as estimating their velocities with reasonable accuracy. The main problem of this proposal is that it was not designed for UAVs.

In [43], Ma improved a previous work by introducing collision and obstacle avoidance capabilities to target tracking. In particular, the author increases the control input with a repulsion term that resolves collisions with other team members and nearby obstacles. Assuming that each UAV travels at a constant speed, a control component is added that adjusts the UAV's heading angle to the opposite direction in relation to the UAV's closest neighbors, and to obstacles that could provoke collisions. This repulsion term can also be expressed as a function of the relative bearing angles alone, making it possible to be estimated/measured by on board vision sensors in the presence of communication losses. Regarding the communication topology tested, an all-to-all communication, a ring topology, and a cyclic pursuit topology are studied. The effectiveness of the proposal is demonstrated using only numerical simulation examples.

Chen and Lee [11] focused on proposing a novel and memory efficient deep network architecture named UAVNet for small UAVs to achieve obstacle detection in urban environments. The proposal shows that UAVNet can detect obstacles at a rate of 15 fps, meeting real-time application requirements.

Most protocols analyze how to avoid collisions between UAVs, but to the best of our knowledge, no protocol has specifically addressed the issue of collision avoidance between multirotor UAVs from independent owners that are following planned missions, like our Mission Based Collision Avoidance Protocol (MBCAP) does (see chapter 5).

2.2.2 Swarm protocols

UAV swarms require safety, avoiding possible collisions among them and with obstacles, and also require efficiency, maintaining the formation and finishing the programmed task as soon as possible. Some applications, like the search for pests or weeds in large-scale agriculture, can be achieved with a single UAV traveling over a wide area until it is fully covered. In this context, the use of a stable swarm can highly speedup the task, searching with several UAVs at the same time. Below we introduce several works that try to maintain the swarm formation based on different approaches.

In [31] the authors proposed an automatic control system for UAV swarms. Specifically, for their analysis, they use two fixed-wing aerial vehicles to maintain the cohesion of the formation. The general idea is to provide a mechanism based on radio-frequency pulses through which each UAV can detect its relative rank and orientation compared to its neighbors. To achieve this, the authors use a variant of the Frenet-Serret equations of motion for the trajectories of each UAV. Unlike our proposals, this solution does not focus on scalable UAV swarms.

In [40] the authors presented a communications protocol for autonomous UAV swarms focusing on a search mission. This proposal combines inter-UAV communication with geographic routing to improve the search efficiency. The authors evaluate their novel protocol by simulation in only two dimensions. In [63] the authors presented a swarm coordination proposal using the traditional 3G/4G communications infrastructure. For communication and coordination among the UAVs, they use the Scalable Data Delivery Layer (SDDL). In [62] the authors proposed the use of a swarm of UAVs with the aim of establishing a wireless backbone over a specified disaster-struck area. For this purpose, they use autonomous agents on each UAV to control them cooperatively. The proposed system achieves the goal of establishing communication between multiple Ground Stations (GSs), maintains a decentralized cooperative control based on behavior to search for unknown GSs, and to retransmit packets from one GS to another.

Later, in [69], the authors presented a topology that adopts Ad-hoc networks as mechanism that could be applied to control the mobility of UAV swarms. Its main features are focused on connectivity and coverage area. Then, in [7], extensive studies on the use of Flying Ad-hoc Networks (FANETs) were made, describing the main problems of deploying Ad-hoc networks based on UAVs. The authors describe features such as topology changes, radio propagation model, adaptability, scalability, latency, UAV platform constraints, and bandwidth.

In [38] the authors proposed the use of microdrones equipped with Zigbee modules for communication purposes. The solution supports complex mobility patterns, although it does not cover the needs of an autonomous swarm in the open field since the synchronization of the different aircrafts is achieved via a sensor network installed in a test laboratory. It also presents details about the algorithm and the hardware used for implementation; they validated their solutions through

real experiments using 20 UAVs. Similarly, in [6], the authors used a virtual structure based formation controller for UAV swarm systems moving in the three-dimensional space. This proposal was extensively evaluated using simulation.

In [58] the authors used a controller based on a virtual leader structure to provide a rigid training. They use an approach where the controller cooperates in a decentralized way with the UAVs, allowing them to have a synchronization signal so that it achieves a predefined formation in the presence of a time-varying formation topology. Later, the authors of [17] used a similar approach, but instead they adopt a system having a switching interaction topology to achieve time-varying formations. The switching interaction topology consists of two parts. The first one uses a formation control solution based on two-loops, where the internal loop controller stabilizes the altitude, and the external loop controller drives the UAVs to the desired positions. The second one uses a formation control protocol using the adjacent information of each UAV, and where the formation can be time-varying. Also, they validated their approach in real scenarios using four quadrotors. Both proposals use algorithms based on Lyapunov to analyze the stability of their controllers.

In this work, we propose the Mission-based UAV Swarm Coordination Protocol (MUSCOP) (see chapter 6), that differs from all the previous ones since it defines and maintains the formation of UAVs in a swarm following a planned mission. The swarm leader will use the different waypoints defined for the mission as checkpoints to make sure all UAVs remain available and maintain their relative positions in the swarm.

On the other hand, some tasks like the search for missing persons in wide areas require the swarm to be manually guided as the target location is unknown and, at the same time, the swarm formation must be kept stable. Hence, below we refer to several works that instead try to dynamically control a swarm.

In [56, 45] the authors proposed to control swarms using the Dynamic Data-Driven Application System (DDDAS) [15]. These techniques are used in [56] for testing with real data, which are then injected into a simulation environment having multiple UAVs. The solution described uses the MASON library to simulate swarms. DDDAS allows the different nodes in a swarm simulation to receive location and other types of information from either real-world UAVs or simulated UAVs, and in return the swarm simulation environment is able to steer these UAVs. In [45], the authors use these frameworks to test different scenarios, and determine if a region can be cleaned or not using swarms.

Palat et al. [54] propose an algorithm to create swarms of robots. The control algorithm is based on the indirect pheromone communication typical of social insects, such as ants. The authors implement two mobile software agents: i) ant agents, and ii) pheromone agents. The first one generates and maintains the local information about the formation that serves to guide the others and determine the target location. The second pheromone agents clone themselves and move to other robots to find the target ant. When ant agents receive pheromone agents, which

have information to guide the ant agents, the ant agents move to the locations that the pheromone agents point to.

In [4], authors proposed using swarm clouds as multiple GCSs. The idea of this work is to reduce the efficiency problem in several missions, being controlled by a GCS. The communication and accessibility of the different UAVs are improved since each UAV has a single connection to an individual cloud. The problem with this solution is that it needs a Ground Station that is in charge of synchronizing the aircrafts.

Our FollowMe protocol (see chapter 7) fills-in a gap by focusing on applications where multicopters must be manually guided. This is applicable to situations where the pilot must respond to visual stimuli in real time, and in such a way that the other UAVs in the swarm that follow the leading multicopter have to adapt to these unexpected movements dynamically.

Chapter 3

UAVs built for real experiments

The main objective of this work is to develop ArduSim, and then several flight coordination protocols using this simulation platform. Afterwards, we plan to deploy these protocols in real multicopters, which requires adapting the UAVs to enable UAV-to-UAV communications among them, and between the flight controller and the hardware needed to run ArduSim. In this chapter we introduce prototypes that were built to achieve this purpose.

The Pixhawk controller (see chapter 2) is an advanced flight controller, and it includes several ports to connect different devices, but it is not possible to connect a WiFi adapter, or even to run a UAV coordination protocol due to its reduced computational power. To this end, we built three prototypes where a Raspberry Pi model 3B+ is attached to the multicopter (see Figure 3.1).

Nowadays, there is no standard available for UAV-to-UAV communications, but some efforts have been made with ADS-B [1] to provide them the current location of other UAVs. Some other efforts try to communicate using LTE [28],

(a) GRCQuad (b) GRCHexa (c) GRCHexa2

Figure 3.1: Multicopters used in real testbed.

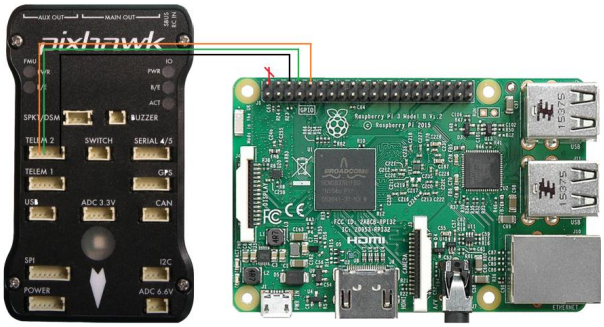


Figure 3.2: Raspberry-Pixhawk serial link setup.

as there is a large infrastructure already deployed in many countries, but this second option requires UAV-to-UAV communication to be relayed through an LTE node, therefore introducing unnecessary delay. Most authors propose to establish and Ad-hoc network for UAV-to-UAV and UAV-to-GCS communication. This approach provides a direct link between UAVs, but also introduces some challenges related to the fast evolution of the network topology with the relative movement of the UAVs. Current routing protocols are not able to adapt fast enough to these changes, so new routing protocols are being developed by the research community [36]. Considering the reduced amount of alternatives to establish a link, we decided to build an Ad-hoc network based on the 802.11a protocol, where messages are broadcasted to make all the UAVs receive the same data. This solution fits most of the possible situations, as the UAVs forming a swarm usually maintain direct LOS among them, and therefore there is no need to introduce routing. At initial stages, we used external dual-band WiFi adapters connected to a Raspberry Pi 3 B. Later, when the Raspberry Pi 3 B+ model was released, we used its internal WiFi adapter instead, as it is already compatible with the 5 GHz frequency band.

ArduSim runs in the Raspberry Pi single-board computer, and requires a communication link with the flight controller to modify the behavior of the multicopter. To this aim, we connected the Raspberry Pi through a serial port to the second telemetry port of the Pixhawk, as shown in Figure 3.2. ArduSim runs the deployed protocols in the Raspberry, using the MAVLink protocol. Once a protocol is thoroughly tested, it can be ported to commercial multicopters, writing a new implementation for the microcontroller already included in the specific multicopter. This solution provides researchers a way to build cheap multicopters to develop and test their own protocols.

Chapter 4

ArduSim simulation platform

As the popularity of UAVs increases, new applications appear that require the coordination of a group of multicopters to speed up a task, or even perform tasks impossible to complete without this kind of devices. Nowadays, the implementation of a new UAV coordination protocol is a slow process, as it must be validated in a simulator, and then it must be implemented again for real multicopters. Moreover, the debugging process with real multicopters is expensive and time consuming, and it can lead to crashes until the protocol is fine-tuned for the real world.

Several UAV simulators are able to emulate the behavior of a real multicopter with great accuracy, but they can only run a single UAV at a time, and they also lack the ability to establish a communication link among several UAVs, an essential functionality for many protocols where the UAVs react to the information received from other UAVs. On the other hand, several network simulators, like OMNeT++, can establish the required communication link among UAVs, but they cannot simulate the required realistic and dynamic mobility pattern for the UAVs. In this chapter we introduce ArduSim, a simulation platform designed to fill this gap, simulating many UAVs at the same time and with great accuracy, and establishing a simulated wireless communications link among them. Moreover, any protocol implemented in ArduSim is directly portable to real multicopters, making the deployment somewhat trivial.

4.1 ArduSim design and implementation

ArduSim was developed in Java, and it has a modular structure, that is, the graphical interface, the communication with the virtual UAVs and between them, and the usability of the simulator itself, are all implemented on independent packages. This eases the implementation of new inter-UAV communication protocols, while avoiding having to learn all the details associated to communication with virtual UAVs using the MAVLink protocol [47].

Some of the features of ArduSim include:

- **Effortless protocol deployment on real UAVs**. Current open-source flight controllers use the MAVLink communications protocol to communicate the UAV with an optional Ground Control Station (GCS). ArduSim uses this protocol to fully control the behavior of the UAV while it is flying. The only requirements to deploy a protocol in a real multicopter are to attach a Raspberry Pi with a WiFi adapter (or a similar device capable of running Java), and to connect it to the telemetry port of the flight controller, following the instructions detailed on the ArduSim repository[1]. ArduSim was designed to abstract the UAV control and communication layers to the developer, so that the same developed code works equally in simulation and in real UAVs, making the deployment straightforward.

- **Soft real-time simulation**. Simulations in ArduSim are performed in near real-time, which speeds up the debugging process while the protocol is implemented.

- **High scalability**. On a high-end computer (Intel Core i7-7700, 32 GB RAM, SSD hard drive), ArduSim is able to run up to 100 UAVs in near real-time, and up to 500 UAVs in soft real-time.

- **UAV-to-UAV communication simulation**. The communication among virtual UAVs is performed through virtual links based on 802.11a technology, using a model based on the results gathered from experiments with real multicopters. When the protocol is deployed in real multicopters, ArduSim automatically broadcasts UDP datagrams, requiring a WiFi adapter connected to an Ad-hoc network.

- **Complete Application Programming Interface (API)**. ArduSim provides a complete set of functions to perform the most common maneuvers during a flight: take-off, start a mission, pause a mission, land, and so on.

- **Deployment through a PC Companion**. ArduSim can be run in three different roles: (i) protocol testing on simulation, (ii) protocol deployment in

[1] https://bitbucket.org/frafabco/ardusim

a real multicopter, and also (iii) as a PC Companion that helps to start and control the execution of the distributed protocol when deploying a real UAV swarm. Moreover, the PC Companion tool allows to recover control over the UAVs in case the protocol does not behave as expected, thus avoiding any crash during the first tests with real UAVs.

- **Automatic collision detection**. Safety is a critical aim for any protocol. ArduSim informs the user if any collision happens during a simulation to help the researcher to detect failures in the protocol design.
- **Comprehensive experiment data logging**. When the experiment ends, either in simulation or in a real multicopter, ArduSim stores, among others, the path followed by the UAV including coordinates, heading, speed, acceleration, distance to origin for each data recorded, as well as the same path in Google Earth [26], ns-2 [52], and OMNeT++ [53] formats.

4.1.1 Simulation Architecture

To simulate a great number of UAVs simultaneously, we have used the Software In The Loop (SITL) application as a basic development module. SITL contains control code resembling a real UAV, simulating its physical and flying properties with great accuracy. A SITL instance is executed for each virtual UAV, and it runs together with its physical engine on a single process. The main limitation of SITL is that it only simulates a single UAV, being thus inadequate to develop communication protocols between UAVs.

Figure 4.1 shows the proposed simulation platform, which relies on a multi-agent simulation architecture that implements a high-level control logic above SITL itself. ArduSim allows configuring UAVs and starting experiments directly from its graphical interface *(GUIControl)*. In addition, it includes the simulation of packet broadcasting between UAVs *(Simulated broadcast)*, and the detection of possible physical collisions *(UAV Collision detector)*. The latter has been solved using a thread that periodically checks if the simulated UAVs are close enough to assert that a collision has happened, based on the information provided by the flight controller of the virtual UAVs.

Each virtual UAV is composed of an agent in charge of controlling the UAV behaviour, and the different threads required for the protocol being tested. The communication between UAVs requires two threads, one for sending data packets *(Listener)*, and another one for their reception *(Talker)*. Moreover, the developed protocol can use an additional thread *(Protocol logic)* to command the UAV taking into account the logic of the protocol, and the messages received by other surrounding UAVs.

An ArduSim agent includes a SITL instance, and a thread *(Controller)* in charge of sending commands to the multicopter, and of receiving the information that it generates. Such communications rely on the MAVLink protocol (see section

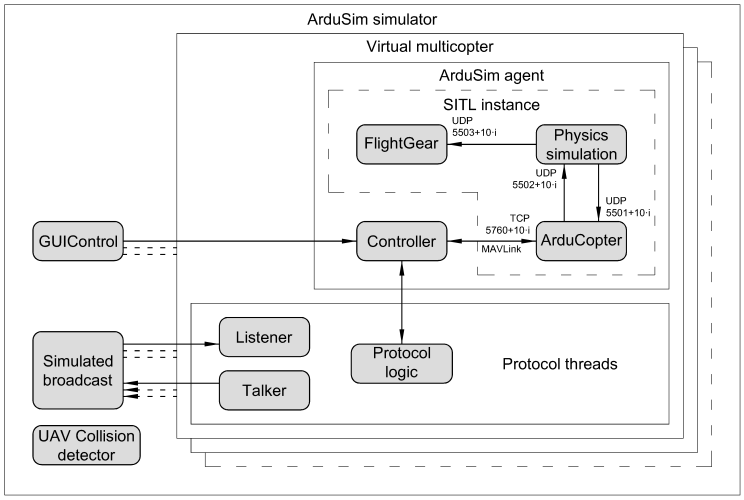


Figure 4.1: ArduSim internal architecture.

4.1.2). When running ArduSim on a real UAV it becomes a controller agent, as explained in detail in section 4.1.5.

4.1.2 Controlling multicopters

The *Controller* thread transmits control messages in the MAVLink format via TCP to a SITL instance. Simultaneously, it receives and processes the answers to the given instructions, and the information messages provided by the virtual flight controller. This information (e.g. current position, speed, etc.) can be used by the protocol being developed in order to achieve the desired functionality. In real UAVs, such communications rely on a serial port connection towards the flight controller (see section 4.1.5).

4.1.2.1 ArduSim-to-UAV communications API

In table 4.1 we show a small sample of the extensive API of more than 100 functions provided for the developer to control multicopters. They allow to implement new protocols easily, without needing to know how the MAVLink protocol works.

The commands are grouped into different categories according to the type of message that is being transmitted between the simulator and the virtual flight controller, as detailed in section 4.1.2.2.

Regarding the implemented set of commands, several clarifications are due. First, the flight mode used by the flight controller is implementation-dependant, meaning that the different flight modes used should be tested when porting any

Table 4.1: Control commands as shown in Figure 4.2.

Simple CMD commands	
setParameter	Modifies a UAV parameter.
getParameter	Retrieves a UAV parameter value.
setFlightMode	Switches to another UAV flight mode.
armEngines	Allows arming the engines before takeoff.
takeOffGuided	Takes off until reaching a specific height (m).
setPlannedSpeed	Changes the flight speed (m/s).
setCurrentWP	Indicates the waypoint where to move to.
moveTo	Moves the UAV to specific GPS coordinates.
removeMission	Eliminates the current mission.
Throttle on command	
stabilize	Stabilizes the UAV height before stopping it.
Send wp list command	
setMission	Loads the specified mission on the UAV.
Get wp list command	
getMission	Retrieves details about the current mission.

protocol to a real UAV. Second, if more than 15 seconds pass between the engine arming and the takeoff processes, the flight controller will disarm the UAV for security reasons. Third and last, to move a UAV to a specific set of coordinates, it must previously be in the *guided* flight mode, as required by the MAVLink protocol.

The set of commands defined make the communications with the UAV transparent to the developer, and they return a boolean value to indicate whether execution was successful or not, thereby simplifying the handling of communication errors at a high level. At a low level, they are in charge of complying with the communication protocols defined in the MAVLink standard.

4.1.2.2 MAVLink communications implementation

The finite state machine depicted in Figure 4.2 shows part of the communications taking place between the *Controller* thread and the virtual flight controller. It is in fact a simplified version of the actual state machine, which has a total of 42 states, and that takes into account all the commands implemented.

Each time a data packet is received from the flight controller, the simulator checks its current state, and analyzes whether it should take any action. If the state is *OK*, then no command has to be executed. Otherwise, it means that a command was issued, or that some message sent from the flight controller requires a reply. In addition to the answers to the different commands, the simulator

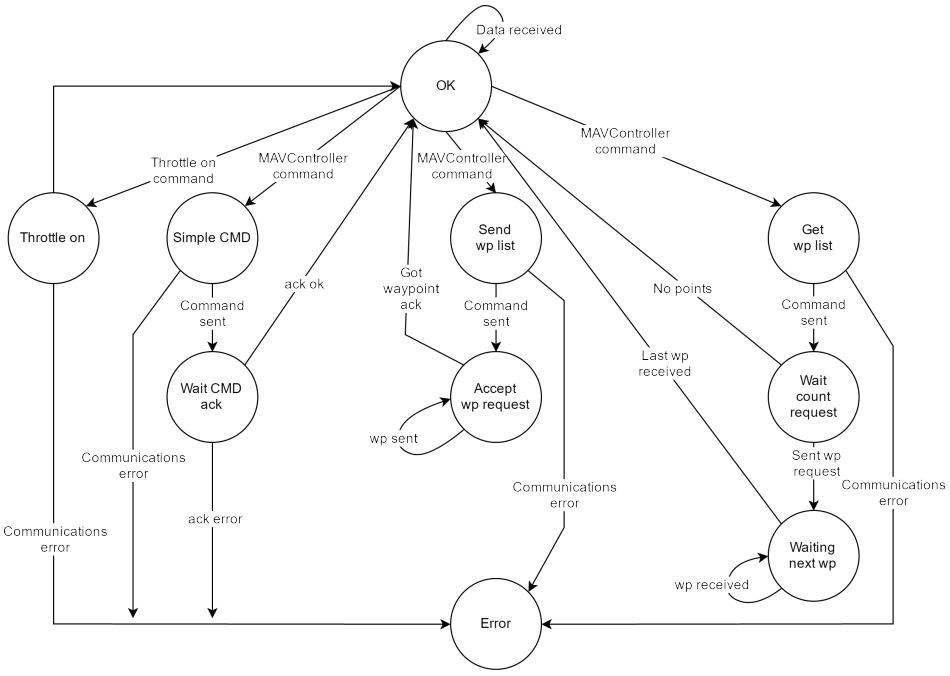


Figure 4.2: MAVLink communications finite state machine.

constantly receives a great amount of MAVLink messages with information about the actual situation of the UAV. Among others, it receives data regarding the position, speed, attitude, and flight mode.

Concerning the implemented functions (see table 4.1), there are four types of interaction between the simulator and the flight controller of a virtual UAV, as shown in Figure 4.2:

- *Simple CMD.* Adopted by the overwhelming majority of commands. A command is issued, and the flight controller must return an acknowledgment (ACK). When this ACK is received, the interaction ends.

- *Throttle on.* Used to take control of the flight altitude during a flight. The interaction ends just after the command is issued, and no ACK is required. This command simulates the presence of a remote control when none is controlling the UAV, i.e., when the protocol under development does not require the intervention of a pilot. This command must be used when the UAV leaves the *auto* flight mode, or when it starts a flight in *guided* flight mode, as the flight controller considers that the communication with the

remote control has been lost, causing the UAV to perform an emergency landing.

- *Send wp list.* Required to send a planned mission to the UAV. First we submit the total number of waypoints associated to the mission, and the controller reacts by requesting, one by one, the different waypoints; the thread will then submit them sequentially until the flight controller returns an ACK to confirm that all waypoints have been successfully received.

- *Get wp list.* Employed to recover a mission stored in the UAV. It starts by requesting the mission. The controller returns a message to indicate the number of waypoints conforming the mission. If the UAV has a mission stored, that is, if the number of waypoints is not null, the thread will request them sequentially until all are received; at that time, an ACK is sent back to the controller.

4.1.3 UAV-to-UAV communications

Currently, the most straightforward way to provide communications between real UAVs is to rely on broadcasting using UDP. Since a wireless link is created, the simulator should take into account the signal range in order to determine whether a packet arrives or not to the neighboring UAVs. This means that a realistic communications model should be adopted. Some existing simulators rely on a simple model, where a distance threshold is used to discriminate between received and discarded packets, while others rely on the Friis equation, or a theoretical model such as Okumura or Nakagami [10], all having significant computational costs when running a large amount of UAVs at the same time.

ArduSim includes three different channel models depending on the desired degree of accuracy:

- *Unrestricted.* It uses an ideal medium where data packets always arrive to all possible destinations (basic model).

- *Fixed range.* Data packets arrive to another UAV only if the distance between them is lower than the defined threshold (simple model).

- *Realistic 802.11a with 5dBi antenna.* The probability that a data packet is received by another UAV depends on the distance between UAVs according to a model obtained from real experiments (realistic model), where the packet loss rate between two multicopters was measured using a WiFi Ad-hoc network link in the 5 GHz band (channel 36, 23 dBm transmission power).

Overall, the simulator determines whether a data packet transmitted by a UAV is received by each of the neighboring UAVs according to the model used, and the inter-UAV distance.

The third wireless model was obtained by studying the communication link properties between two real multicopters during flight, and provides a model with a reduced computational cost that enables to simulate communications among a large number of UAVs in a single computer. Below we provide a detailed explanation on the process followed to obtain this model.

4.1.3.1 Dronning tool

In order to model the communications link between two multicopters, the first step followed consisted on preparing two quadcopters (see Figure 2.2) with a Raspberry Pi and a dual-band wireless adapter, and developing the *Dronning* tool to measure the packet loss ratio when a UAV sends data packets to another UAV. This way we can define the experiment conditions, manually controlling the relative location of both multicopters, and changing the transmission rate, the datagram size, the experiment duration, and whether the messages are sent via broadcast or unicast.

The *Dronning* tool is able to run with three different roles: client, server or controller. The client and server instances of the tool run on the Raspberry Pi connected to the flight controllers of the multicopters, and the controller instance runs on a laptop. The three different application instances are connected to each other via sockets, over an IEEE 802.11-based Ad-hoc network. The controller instance is in charge of setting the experiment parameters on client and server, and it coordinates the start of the test. An additional link is established between server and client, and it allows to send data packets from the former to the latter to measure the packet loss ratio under several circumstances.

If running at the controller, it provides a simple and yet complete graphical interface, as shown in Figure 4.3. When the application starts, UAVs having client and server roles attempt to discover and connect to an existing controller, which should be locally accessible at a predefined IP address and port. Once connections take place, the progress bar on top turns to green when this initial step is completed, and the two compass elements below show yaw angles for both UAVs; additionally, altitude is computed, along with relative distance and relative height information.

The operator of the controller application starts by defining experiment parameters, including trace file name, test duration, transmission rate, packet size, and the type of transmission (broadcast or unicast). Once this information is introduced, it sends the configuration to both UAVs by pressing the appropriate button. Once that step is highlighted in green, tests can start by pressing the *test* button. After tests are completed, the bottom right of the interface shows a test summary including the throughput achieved, and packet loss information.

Figure 4.4 shows the anatomy of a test using the *Dronning* tool. The controller uses the control link to define the tests conditions and parameters, and to synchronize the beginning of experiments, while the server sends data to the client through a different link. Notice that a single Ad-hoc network is created for com-

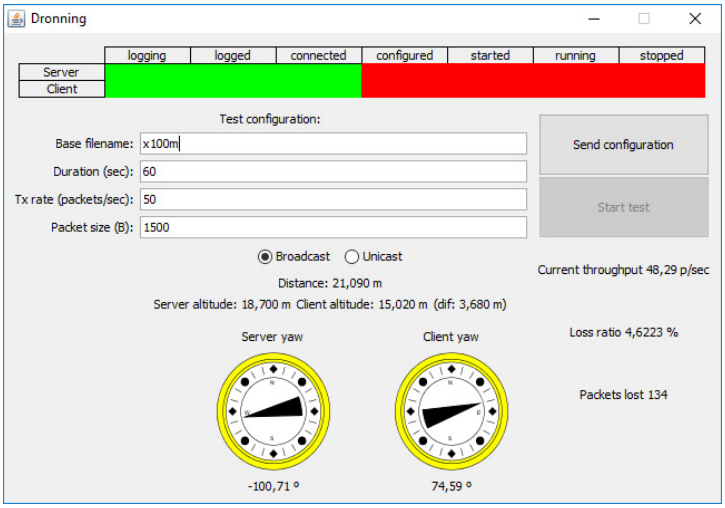


Figure 4.3: Dronning controller graphical user interface.

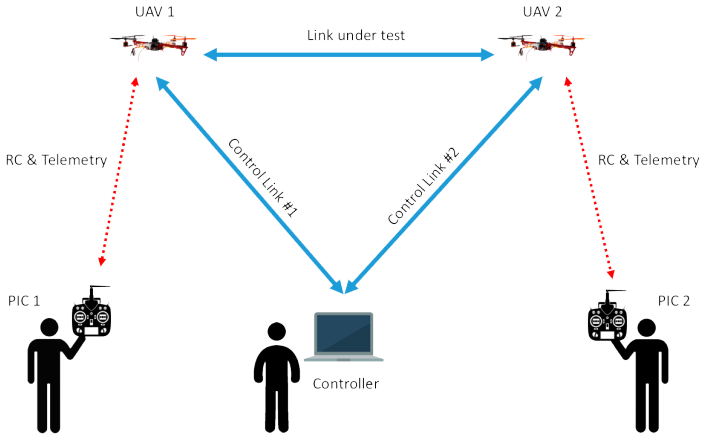


Figure 4.4: Diagram of the different elements involved in performance tests.

munications between UAVs, and from UAVs to Controller. For this reason, the control link becomes idle when tests are ongoing to avoid interfering with results.

In addition to the controller terminal and operator, two additional people are involved in real tests; these are the two Pilots in Command (PICs), responsible for positioning the UAVs at the desired locations according to the instructions of the

controller. The purpose is to achieve the target parameters for the test in terms of distance between UAVs, relative and absolute height, etc. Notice that, between the PICs and the UAVs, additional channels are created for Radio Control tasks and telemetry data.

The Raspberry Pi of each multicopter is connected to a telemetry port of the flight controller, which enables to read flight data during the experiment by means of the MAVLink protocol. We specifically gather location, speed, and attitude data. The server instance sends the current value of these parameters to the client on each message sent, so it can store the received and its own data on a file for further analysis. Only 86 Bytes are enough to transfer the needed information, so padding is added to the message to achieve the size defined by the user.

The stored file is processed later on a computer using macros that calculate the following parameters: number of packets sent/received/lost, and minimum and maximum sizes of loss bursts. In addition, they also calculate the mean, standard deviation and 95% confidence interval for the following variables: mean loss burst size, temperature, roll, pitch, yaw (and their respective angular speeds), latitude, longitude, altitude, elevation, Universal Transverse Mercator (UTM) coordinates, 3D speed components and total speed, bearing, distance between UAVs, height difference between UAVs, and relative angle between UAVs (with and without accounting for elevation differences).

In addition to calculating the aforementioned statistics, the following charts are also calculated automatically: histogram and time sequence for packet loss bursts, histograms for UAV positions in the 3 axis, zenith chart of UAV position variations, time sequence of the total UAV speed, and time sequence for the relative angle between UAVs (with and without accounting for elevation differences). Based on all the aforementioned data, it is possible to easily assess channel conditions, and relate them to the actual mobility and environmental conditions (e.g. wind, obstacles,...).

4.1.3.2 Results for the 2.4 GHz band

We firstly tested the 2.4 GHz frequency band, as it is available in all the WiFi adapters currently available. We also knew that most remote controllers use the same band to guide the multicopter, which could be a serious issue when attempting to achieve a stable communications link. Moreover, they use a frequency-hopping strategy that occupies the entire 2.4 GHz band, meaning that all WiFi channels are affected to a same extent.

The Ad-hoc network was established with TP-Link TLWN722N wireless adapters operating in channel 1 using the IEEE 802.11g annex. Both cards were endowed with a 5 dBi external antenna, and the transmission power was of 100 mW. Each experiment had a duration of 60 seconds, and a packet transmission ratio of 50 packets per second. If not stated otherwise, the packet size was 1500 Bytes (maxi-

mum Ethernet MTU). Since packets were broadcasted, the transmission data rate was of 6 Mbit/s.

The analyzed factors and experiment parameters were the following:

- **Separation between UAVs** (signal attenuation) and **remote control activation** (electromagnetic interference). The transmitted signal decreases with the distance. In addition, significant interferences are expected as the remote control link uses the same frequency band as the UAVs communication link, as discussed above. The packet loss ratio was measured for different distances between UAVs, with the remote controls off and on.

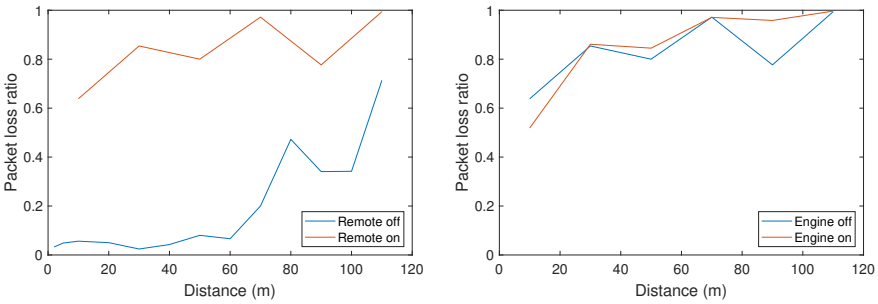
- **Engine power on** (electromagnetic interferences). The running engines of the UAV produce electromagnetic interferences that may affect the wireless signal. For this experiment the UAVs were located on the floor. Then, the packet loss ratio at different distances between UAVs was measured in two situations: with the engines off, and with the engines on but at a very low power. The remote controls were turned on during both test series.

- **Engine speed** (structural vibrations). If the engine power increases, the UAV starts to vibrate. Furthermore, the engines emit more electromagnetic radiation, which could increase the effect already studied in the previous experiment. Both UAVs where anchored to the ground at a distance of 20 meters between them. The engine power was increased from 0 to 100% in intervals of 25%.

- **Separation between UAV and remote control** (electromagnetic interference). Since the control signal can produce interference, the distance between the remote control and the UAV could significantly influence the communications link quality. As in previous experiments, the UAVs where positioned on the ground at a distance of 20 meters between them. The link quality was measured when separating the remote control from the UAV at short-range distances.

- **Data packet size** (media noise). The longer the packet being sent, the longer time the wireless media is busy, and the probability of noise transmission errors increases. Both UAVs were 20 m apart, and measurements were made with packet sizes of 300, 1000 and 1500 bytes.

- **Ground elevation** (multi-path fading and Fresnel zone occupancy). The transmitted signal reflects on the ground, which could produce multi-path fading. Furthermore, the Fresnel zone around the LoS could be affected by the ground, lowering the strength of the received signal. During the tests, both UAVs were 20 meters apart, and measurements were made for different elevation values.



(a) Effect of remote control interference and distance.

(b) Effect of turning on the drones engines (25% power).

Figure 4.5: Packet loss vs. distance.

- **Relative antenna orientation**. Real antennas are not isotropic, and so the relative orientation between transmitting and receiving antennas affects the link strength. The Dronning tool provides charts with the evolution of relative antenna orientation and bursts of packets lost throughout the test time. The comparison of these charts allows to detect the correlation between the relative antenna orientation and the link quality.

Figure 4.5a shows the results for the first experiment, where we measure the packet loss ratio with the UAVs static near the ground at different distances, and with the remote controls off and on. We can observe that, when the remote controllers are turned off, the packet loss ratio is relatively low and increases with the distance. When the remote controllers are turned on, the packet loss ratio shifts to high values even at close distances. The presence of the remote control makes the communication link nearly unusable, which encourages us to switch to the 5 GHz frequency band, as it is also free and supported by many wireless adapters.

The second experiment (see Figure 4.5b) studies if the electromagnetic interference generated by the engines has an influence on the communications link quality. We find that the packet loss ratio is very high, and confirmed that it increases with the distance between UAVs. Nevertheless, there is no significant difference in terms of loss magnitude, as both series overlap, which means that merely turning on the engines does not significantly affect the communications link quality.

The third experiment (see figure 4.6) shows that the packet loss ratio increases with the engine power from about 70% to about 84.5%. This is mostly due to the vibration of the structure of the quadcopter, causing antennas to tilt, thereby affecting the signals sent and received.

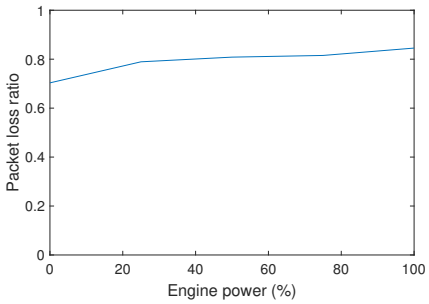


Figure 4.6: Packet loss vs. engine power. Effect of vibration due to engine lift power.

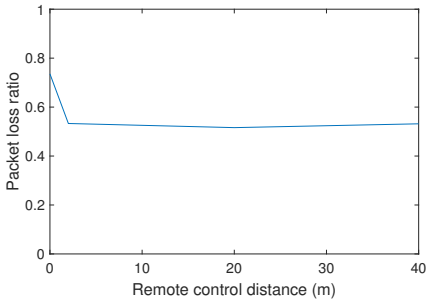


Figure 4.7: Packet loss vs. remote control distance. Effect of remote control interference.

Accounting for the aforementioned issues, and considering that remote controls are also a source of interference in the 2.4 GHz band, we proceed to study how the distance between the remote controllers and the UAVs affects the packet loss ratio. Figure 4.7 shows that, indeed, the proximity between the remote control and the UAV has a clear influence on link quality. However, the effect is only significant when the remote control is very close to the UAV (<1m). Since such distances never take place during normal UAV operation, we can discard this effect.

In general, sending larger data packets increase the probability of transmission errors, as the physical media is busy for a longer period. Thus, the effect of the packet size has also been analyzed (see Figure 4.8) by testing with packet sizes of 300, 1000, and 1500 bytes. The initial hypothesis was verified, as the packet loss increases with the packet size. However, the overall packet loss ratio is quite low (<4%). This means that, compared to other factors affecting inter-UAV communications performance in the 2.4 GHz band, packet size is a parameter with little overall relevance.

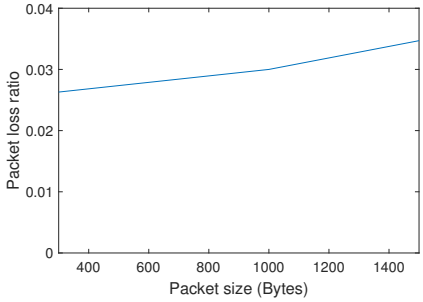


Figure 4.8: Packet loss vs. datagram size. Effect of datagram size with the UAVs on the ground and the remote controls off.

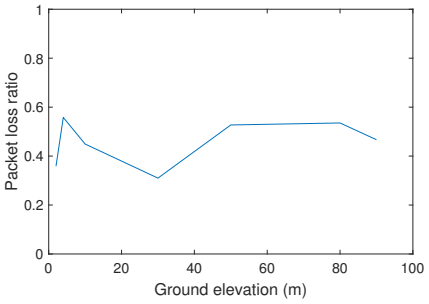
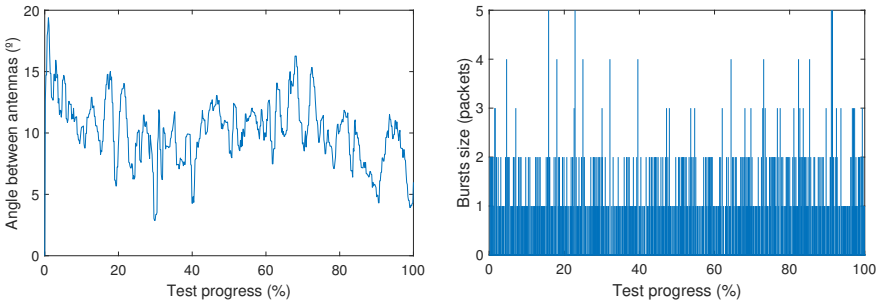


Figure 4.9: Packet loss vs. elevation. Multi-path fading effect at different altitudes.

Regarding the effects related to signal propagation, the WiFi signal is reflected by the ground and other surfaces like any wireless signal. Moreover, the ground can represent a significant part of the Fresnel zone when assuming free-space communications, thereby reducing the link quality. However, this experiment (see Figure 4.9) failed to clearly confirm these effects, as the interference produced by the remotes mask the results.

Finally, we attempted to find a correlation between the relative antenna orientation and the packet loss. To achieve it we flew UAVs at a 15 m distance between them, and at a ground elevation of 30 m, in GPS hold mode, and under windy conditions (∼20 km/h), which caused UAVs to continuously adjust their position, thereby causing antennas to tilt. Simultaneously, we captured packet losses along with flight attitude parameters and GPS information, using this information to obtain the relative angle between UAVs.

Figure 4.10a shows the evolution of the relative orientation of the UAV antennas along one test, while Figure 4.10b shows the burst sizes associated to lost



(a) Angle between antennas vs. test progress.

(b) Size of packet loss bursts vs. test progress.

Figure 4.10: Packet loss vs. antenna orientation.

packets. We find that, despite some peak angular values have a match in terms of packet loss bursts, no correlation can be found between both charts in the strict sense. Again, we consider that the interference from remote controllers contributes to masking this effect, preventing us from having a clear view of the results.

Along these experiments we found that a resilient WiFi transmission in the 2.4 GHz band is incompatible with the UAV remote controls widely available in the market. In fact, we found that the packet loss ratio is unacceptable for nearly all applications. Thus, it would be convenient to use remote controls that work in other frequency bands, or shift the communications link to another frequency band.

Overall, it has been found that several factors influence the communications link quality, such as the distance between UAVs, or between UAV and remote control, in addition to the data packet size and the structural vibration caused by UAV engines. In particular, the greater the distance between UAVs or the data packet size, the higher becomes the packet loss ratio, which means that there will be a lower quality for the communications link. On the other hand, the remote control proximity only affects communications performance at very short distances that are unfeasible during normal UAV operation. Based on telemetry parameters and GPS information, we have also measured the influence of ground reflection and relative antenna orientation on link quality; however, the high interference levels caused by the remote control prevented reaching statistically representative differences.

4.1.3.3 Results for the 5 GHz band

As argued before, the 2.4 GHz frequency band is not suitable for UAVs to communicate between them as we usually keep the remote controls turned on in case it becomes necessary to recover control over the device if the protocol under test

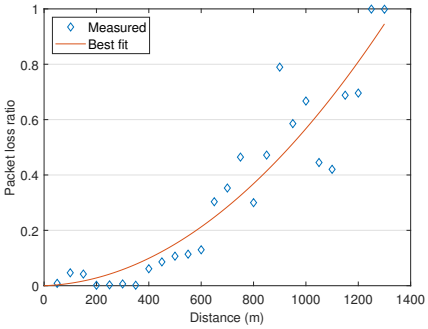


Figure 4.11: Packet loss vs. distance (IEEE 802.11a, 5 dBi antenna).

fails. Thus, we repeated the first experiment of the previous list to analyze the effect of the distance between UAVs on the link quality in the 5 GHz band. We used an Alfa AWUS051NH wireless adapter with a 5 dBi antenna, and with a transmission power of 200 mW.

Figure 4.11 shows the packet loss rate obtained when varying the distance between UAVs. Now, the remote controllers do not affect the link quality, and so we could use these results to define a realistic wireless channel model for ArduSim. Beyond 1350 meters we consider that packet losses reach 100%, while for lower distances the following polynomial applies:

$$y = 5.335 \cdot 10^{-7} \cdot x^2 + 3.395 \cdot 10^{-5} \cdot x \tag{4.1}$$

4.1.4 Virtual link implementation

IEEE 802.11-based networks rely on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), an algorithm for medium access arbitration. Thus, to make communications more realistic in the scope of our simulator, we have implemented the carrier sensing functionality. Regarding the collision avoidance mechanism used in 802.11, it involves very short waiting times (DIFS) before transmitting a data packet, which is not possible to implement in real-time simulation without performing active waiting. This occurs because the time slice that the system grants to each Java thread is larger than this value, and so there are no guarantees that the packet will be transmitted after that time if a passive wait is made. On the other hand, the solution is not scalable if active waiting is performed, because each thread tries to use a CPU core completely, preventing the simulation of more than 2 or 3 simultaneous UAVs on standard PCs. Such limitations forced us to implement a mechanism to detect collisions on the wireless channel, while discarding some of the collision avoidance features of CSMA/CA. We consider that our solution offers an adequate trade-off between channel be-

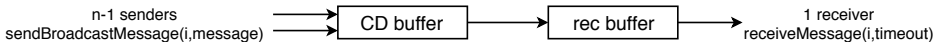


Figure 4.12: Simulated broadcast model.

haviour accuracy and performance, meeting real-time constraints despite CPU limitations.

The carrier sensing, collision detection (physical level), and reception buffers, have been simulated together by means of two functions, the first one for the packet transmission, and the second one for the packet reception process; the data structures shared between both these functions act as reception buffers. This way, carrier sensing is simulated when a message is sent, while collision detection is done when it is received.

Figure 4.12 shows the model used to simulate packet transmission via broadcast. The reception buffers *(rec buffer)* are First In First Out (FIFO), and they can block the thread until a data packet is received, having a configurable size (163840 bytes by default). In addition, each reception buffer is preceded by another buffer *(CD buffer)* used to detect collisions on the channel.

When simulating n UAVs, each of them can simultaneously receive messages from a maximum of *n-1* UAVs (thread *Talker*, see Figure 4.1, using the *send Broadcast Message* function), which are then inserted in the *CD buffer* and ordered according to the instant when transmission starts. If carrier sensing is activated, the transmission does not start until the medium is available.

When a protocol being developed requests a message (thread *Listener*, using the *receiveMessage* function), it first checks if there is any message in the reception buffer. Otherwise, collision detection is applied to the messages available in the *CD buffer*, eliminating those messages that have collided, and moving the rest to the reception buffer for its own use. This solution allows us to detect collisions only when there is no data in the reception buffer, and not whenever a new message is requested, thereby reducing the computational cost considerably.

The intermediate buffer, in charge of simulating the wireless medium, allows us to detect collisions, and it could grow indefinitely if the receiver does not request any message. Although this approach would be the ideal solution from the collision detection mechanism perspective, it is not viable since RAM memory is a limited resource. For this reason, the size of the *CD buffer* is limited to twice the size of the reception buffer *(rec buffer)*. It is worth mentioning that, if collision detection is not required, the intermediate buffer is deemed unnecessary. In this case, the threads insert the messages directly into the *rec buffer*.

In order to determine if the medium is busy (carrier sensing), and whether two received messages have collided (collision detection), it is necessary to determine the start and end times of a message transmission taking into account the transmission speed and the length of the frame. Regarding the transmission speed, the communications model uses the 5 GHz band, and the transmission is made via

broadcasting, meaning that the transmission rate is 6 Mbps. The end of the transmission is determined by also taking into account the size of the frame, including the preamble, according to the specifications of the 802.11 protocol. In addition to the start and end times for message transmission, it is also necessary to store the value of two variables (isChecked, isOverlapped) for each message in order to detect collisions, as explained below.

Algorithm 1 details the message transmission process. If the communications protocol is deployed in real UAVs, the transmission is done directly over UDP; otherwise, broadcast transmission is simulated. Once the transmission of the last message has been completed, it checks if no other UAV within range of the transmitting UAV has began a new transmission (carrier sensing). We determine whether a UAV is within the range of the transmitter (function *isInRange*) by relying on any of the communication models described at the beginning of this section, which can be selected by the user. The transmission consists of storing a copy of the message on the *CD buffer* of each UAV within range. If collision detection is not activated, the message is directly stored in the reception buffer *(rec buffer)*. If any of the two buffers is full, the message is discarded, meaning that it is not received at that particular destination.

Every time a UAV sends a message, it stores a copy *(prevSentMessage)*. The instant of completion of a transmission is saved along with the message *(prevSentMessage.end)*, thus allowing to determine if the transmission has finished, and if the medium is available (carrier sensing).

Algorithm 2 details the process of receiving a message. If there are no messages in the reception buffer, algorithm 3 is executed to discard the messages that have collided, moving to the *rec buffer* the remaining messages.

If the protocol is deployed on real UAVs, message reception is done via UDP; otherwise, the transmission medium is simulated.

If collision detection is not enabled, it waits until there is some message available in the reception buffer whose transmission has been completed, and it is delivered. Otherwise, if there are no messages available, the buffer that simulates the medium is analyzed. If this buffer does not contain messages either, it is necessary to wait for a message to arrive; otherwise, if it contains some message(s), the collision detection algorithm is executed.

The collision detection process has a computational cost $\mathcal{O}(2n)$ on the number of received messages, $\mathcal{O}(2n^2)$ on the number of UAVs, and it consists of two steps. In the first one, the messages, already sorted according to the transmission start time *(start)*, are marked as analyzed *(isChecked)*, and among them, those that have collided with other messages are also identified *(isOverlapped)*. A second step is used to eliminate overlapping messages, and to transfer the remaining marked ones to the reception buffer, discarding the message in case this buffer is already full.

The first step of the analysis process is stopped when the last message is analyzed, or when a message is found whose transmission has not yet been completed.

Algorithm 1 sendBroadcastMessage(message)

Require: $message \neq \emptyset$
1: **if** *is a real UAV* **then**
2: *send message through UDP broadcast*
3: **else**
4: **if** *landed* **then**
5: *block communications*
6: **end if**
7: **if** $prevSentMessage \neq \emptyset$ **then**
8: **while** $prevSentMessage.end > now$ **do**
9: *sleep 1ms*
10: **end while**
11: **end if**
12: **if** *carrier sensing is enabled* **then**
13: **while** $\exists i \neq currentUAV \wedge i.prevSentMessage \neq \emptyset \wedge i.prevSentMessage.end > now \wedge i.isInRange$ **do**
14: *sleep 1ms*
15: **end while**
16: **end if**
17: $i = 0$
18: **while** $i < number\ of\ UAVs$ **do**
19: **if** $i \neq currentUAV \wedge i.isInRange \wedge (i.prevSentMessage = \emptyset \vee i.prevSentMessage.end < now)$ **then**
20: **if** *collision detection is enabled* **then**
21: **if** $\neg i.virtualQueue.isFull$ **then**
22: *add message to i.virtualQueue*
23: **end if**
24: **if** $i.virtualQueue.size \geq threshold$ **then**
25: *process algorithm 3*
26: **end if**
27: **else**
28: **if** $\neg i.queue.isFull$ **then**
29: *add message to i.queue*
30: **end if**
31: **end if**
32: **end if**
33: $i{+}{+}$
34: **end while**
35: $prevSentMessage \leftarrow message$
36: **end if**

Algorithm 2 receiveMessage(timeout)

Require: $timeout \in \mathbb{N}$
Ensure: $message \neq \emptyset$
if *is a real UAV* **then**
  if $timeout > 0$ **then**
    setSoTimeout(timeout)
  end if
  $message \leftarrow$ *receive through UDP*
else
  if *landed* **then**
    block communications
  end if
  $elapsedTime = 0$
  $start = now$
  if *collision detection is enabled* **then**
    while $(message = \emptyset \wedge timeout = 0) \vee (message = \emptyset \wedge elapsedTime < timeout)$ **do**
      if *queue.isEmpty* **then**
        if *virtualQueue.isEmpty* **then**
          sleep 1ms
          $elapsedTime = now - start$
        else
          process algorithm 3
        end if
      else
        $message \leftarrow queue.poll()$
        while $mesage.end > now$ **do**
          sleep 1ms
        end while
      end if
    end while
  else
    while $(queue.isEmpty \wedge timeout = 0) \vee (queue.isEmpty \wedge elapsedTime < timeout)$ **do**
      sleep 1ms
      $elapsedTime = now - start$
    end while
    if $\neg queue.isEmpty$ **then**
      $message \leftarrow queue.poll$
    end if
  end if
end if
return *message*

Algorithm 3 packetCollisionDetection(isReceiver)

Require: $isReceiver \in [true, false]$
1: **if** $isReceiver \vee virtualQueue.size \geq threshold$ **then**
2: $iterator \leftarrow virtualQueue$
3: $previous \leftarrow iterator.next$
4: $previous.isChecked \leftarrow true$
5: **while** $previous.end > now$ **do**
6: *sleep 1ms*
7: **end while**
8: **if** $previous.end > endMax$ **then**
9: $endMax \leftarrow previous.end$
10: **end if**
11: **while** $iterator.hasNext$ **do**
12: $following \leftarrow iterator.next$
13: $following.isChecked \leftarrow true$
14: **if** $following.start < endMax$ **then**
15: $previous.isOverlapped$
16: $following.isOverlapped$
17: **end if**
18: **if** $following.end > now$ **then**
19: $following.isChecked \leftarrow false$
20: **if** $following.isOverlapped$ **then**
21: $previous.isChecked \leftarrow false$
22: **end if**
23: *break*
24: **else**
25: **if** $following.end > endMax$ **then**
26: $endMax \leftarrow following.end$
27: **end if**
28: $previous \leftarrow following$
29: **end if**
30: **end while**
31: $iterator \leftarrow virtualQueue$
32: **while** $iterator.hasNext$ **do**
33: $message \leftarrow iterator.next$
34: **if** $\neg message.isChecked$ **then**
35: *break*
36: **else**
37: $iterator.removeMessage$
38: **if** $\neg message.isOverlapped \wedge \neg queue.isFull$ **then**
39: *add message to queue*
40: **end if**
41: **end if**
42: **end while**
43: **end if**

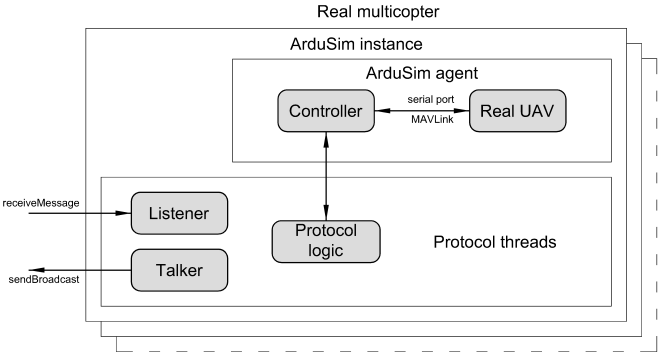


Figure 4.13: ArduSim architecture on real UAVs.

The second run stops after the last message, or when an unchecked message is found ($isChecked = false$), that is, a message not found during the first run. This solution takes into account the possibility of inserting a message among existing ones that have already been analyzed during the first run, since insertions are made concurrently. If the last message analyzed has collided with the second-last one, both are preserved for the next analysis to account for those cases when a message that collides with one of them arrives later on.

4.1.5 Protocol deployment on real UAVs

The ArduSim simulator has been designed to facilitate the deployment of the protocols implemented in real UAVs. The application was developed using Java, and it communicates with virtual UAVs via TCP, simulating the communication among UAVs through buffers that are shared by different threads (Figure 4.1). However, when the application is executed in a real UAV (Figure 4.13), the graphical interface is not shown, the communication with the virtual UAV is replaced by a serial port connection, and the wireless communication between UAVs relies on broadcasting of UDP datagrams. All the simulation-dependent software elements are disabled merely by changing an execution parameter, which makes the deployment of a newly developed protocol somewhat trivial.

To be able to deploy new protocols, it becomes necessary to port the Java application to a Linux or Windows®-based device, with Java 7 pre-installed, along with a physical serial port connection with one of the telemetry ports of the flight controller, in addition to following the instructions provided with ArduSim. Several tests have been performed with a Raspberry Pi having its *ttyAMA0* serial

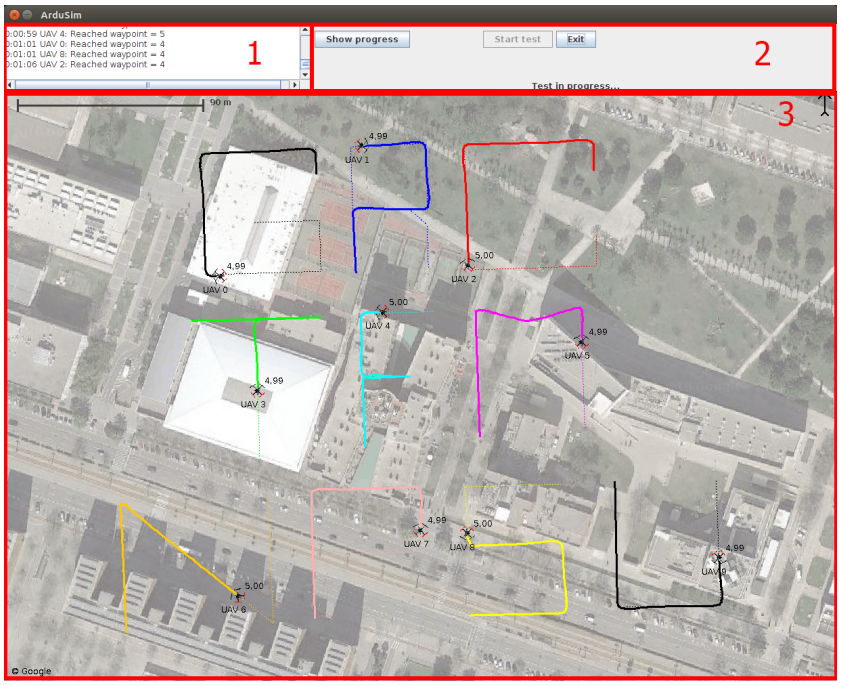


Figure 4.14: ArduSim main window: experiment in progress.

port connected to the *Telem2* telemetry port of the Pixhawk controller embedded in our custom multicopters, allowing us to check the proper deployment of the implemented flight coordination protocols.

4.1.6 ArduSim graphical user interface

ArduSim is oriented to the development of protocols applicable to UAVs performing planned missions, or conforming a UAV swarm. As an example, Figure 4.14 shows ten UAVs performing a mission, represented as letters 'GRC-TFM-NPSU'.

On the upper left corner of the window (1) we can find the application log. It details the functioning of the simulator, as well as the results of the commands sent to the UAVs.

On the right (2) we have the controls that allow the user to manage the application, to start the test, or to close ArduSim. While running a swarm experiment, it also shows an additional button to perform the setup step. In addition, it provides general information about the simulator itself.

Most of the window space (3) is used to visualize UAV flights during tests. On the upper right corner we show the wind direction (if defined for the test). The

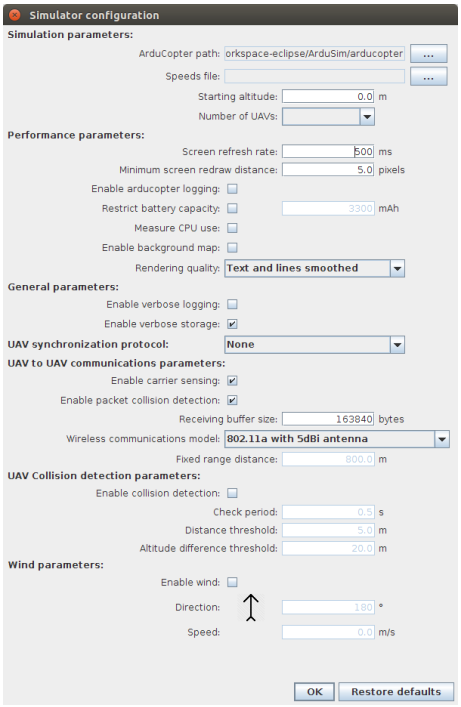


Figure 4.15: Initial configuration dialog.

discontinuous lines represent the mission assigned to each UAV. On each UAV, we indicate its identifier and its altitude. Before starting, each UAV loads the mission to be completed, and simulated wind is also applied. A thick stroke represents the real path followed by each UAV. If the UAVs collision detection feature is enabled, a red circle centered on each UAV is drawn. When a UAV invades that circle, we consider that a collision between UAVs has occurred.

In addition to the main window, an additional dialog window is also opened to show the position, the speed, and the flight mode according to the MAVLink protocol, as well as state information relative to the protocol during the experiment, if needed.

The dialog box of Figure 4.15 is shown when ArduSim is started. It allows the user to specify several simulation parameters, including the flight speed, starting altitude, some performance parameters, the synchronization protocol to be tested, the wireless model to be used and some of its properties, whether or not to detect when collisions happen, in addition to simulated wind speed and orientation.

When an experiment ends, the user decides whether to save the results obtained or not. A dialog is shown (see Figure 4.16) with the configuration and general

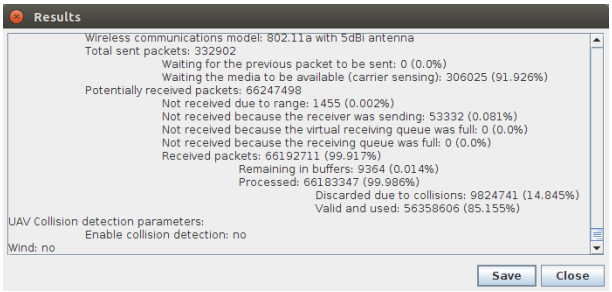


Figure 4.16: Results dialog.

results of the experiment, which includes detailed statistics of the communications among the virtual UAVs, such as the total number of data packets sent, how many had to wait for the media to become available *(carrier sensing)*, or were discarded due to collisions, among others. Several independent files per UAV are also saved with additional information, such as the actual path followed by each UAV during the experiment.

4.1.6.1 Swarm formations

ArduSim integrates tools that allow us to easily define and use different swarm flight formations, optimized for a master-slave communications pattern:

- **Linear.** The UAVs are arranged according to a straight line, which is perpendicular to the heading of the UAV located in the center of the swarm. The distance among contiguous multicopters remains constant.

- **Matrix.** The UAVs are ordered according to a square matrix. Again, a multicopter is located in the center of the swarm, and the distance between contiguous multicopters is the same.

- **Compact matrix.** Similarly to the previous formation, the UAVs are arranged forming a matrix with a constant distance between contiguous UAVs, but instead of building the matrix from a corner, it is built starting from the UAV at the center, and adding multicopters so that the distance to it is minimal, according to the matrix pattern. This way, the mean distance of the UAVs to the center multicopter is minimized, which improves communications, as the packet loss ratio is affected by the distance between sender and receiver.

- **Compact mesh.** The UAVs are ordered around the center multicopter, and forming equilateral triangles among them. This is the most compact planar

Figure 4.17: Swarm layouts: i) matrix with 9 UAVs, ii) linear with 5 UAVs, and iii) circular with 9 UAVs.

formation that can be built, which means that the mean distance between UAVs and the center multicopter is even less than in the previous formation.

- **Circular.** A multicopter is located in the center of the circle, and the remaining UAVs surround it. The distances between the central UAV and the rest of the multicopters, and between the latter ones, can be defined by the user.

Notice that the main parameter used on each formation is the distance between multicopters, as it is directly related to the probability of collisions during flight.

Figure 4.17 illustrates three of the different formations available. The multicopter located in the central position of the swarm formation can be the master, if the master-slave communication pattern is used, in order to optimize communications performance.

Notice that the chosen patterns provide different trade-offs between communication link reliability and area coverage. The linear scenario can provide the greatest coverage area, but with a greater distance between the master UAV and the slaves (worst-case approach in terms of reliability), given a same number of UAVs, and for a same distance between contiguous UAVs. In particular, the multicopters located on edge positions will be quite far away from the master, so the messages transmitted between slave and master are prone to be lost more often, thereby having a negative effect in terms of swarm coordination times. On the contrary, the matrix and mesh formations are more compact (worse area coverage), but the distance between master and slaves is lower, thus minimizing distance-related losses. Finally, the circular formation is in-between the previous two cases

Table 4.2: Hardware used for experiments.

	i7 PC	*i5 PC*
Processor	Intel Core i7-7700	Intel Core i5-2500K
Speed (GHz)	3.6 (max 4.2)	3.3 (max 3.7)
Cores	4	4
Hyper-Threading	yes	no
Cache L1-2-3	4x64KB-256KB-8MB	4x64KB-256KB-6MB
RAM	32GB DDR4 2133MHz	8GB DDR3 1333 MHz
HHDD	480GB SSD	2TB 7200 rpm
GPU	NVIDIA GeForce 8400	HD Intel 3000
Monitor	1920x1080 & 1280x1024	1280x1024
OS	Ubuntu 16.10	Ubuntu 16.10
Java RE	SE 8	SE 8

regarding distance from the master to the rest of the UAVs. Furthermore, this distance is the same for all the slaves, meaning that they are expected to experience similar message loss levels.

4.2 ArduSim validation

Once the ArduSim platform has been introduced, we now proceed to validate its correctness and scalability. To this end, we performed a wide set of experiments by having a variable number of UAVs (i.e., from 1 to 256 UAVs) following a straight path from origin to destination during 5 minutes, being that all UAVs are overlapped, and collision detection is disabled. The flight altitude was set to 5 meters, the speed was of 10 m/s, and the default values of the simulator parameters were used. Experiments were made on two different computers (see table 4.2) to evaluate the influence of the hardware used on the performance of ArduSim.

The target metrics were RAM, hard disk, and CPU usage, as well as the time lag between the UAVs of each experiment with respect to a reference UAV in a single-UAV experiment. This last measurement allows us to evaluate how an increase in the resource consumption levels affects the real-time performance of our tests, and therefore the degree of scalability that can be supported. Notice that, when resource consumption is very high, execution is delayed with respect to the situation when there are sufficient resources, and threads do not have to wait for the scheduler to let them access CPU resources.

Regarding functionality, the application has shown to be fully stable after 4000 executions. The only problems detected occurred when more than 150 simulta-

neous UAVs where tested in the *i5 PC*, which is an issue related to the excessive resource usage, as detailed below.

Hard disk I/O operations slightly affect CPU usage when simulating a high number of UAVs (more than 100); this is due to log maintenance tasks performed by the simulation environment managed by SITL. ArduSim has been designed with this issue in mind, and thus provides two non-exclusive options. First, you can deactivate the SITL log, so disk usage loses relevance, or it can be kept active while running the simulator in root/administrator mode. In the latter case, I/O operations are performed on a virtual disk instead, which is faster than in the *i7 PC*'s SSD hard drive, and certainly much faster than in the *i5 PC*'s mechanical hard drive. In order to compare the results obtained with both PCs, all experiments were performed with the log turned off, and in root mode.

The SITL executable requires a very small amount of RAM, since it is a compilation of a controller firmware, and so it is designed to be executed with very few resources. For this reason, the RAM usage of the simulator is very small, even when simulating 256 UAVs simultaneously. The only circumstance where the simulator consumes a significant amount of RAM is when it is run as root, and a RAM drive is used to store the SITL logs. In this case, a maximum of 50 MiB per UAV is used in the RAM drive, which represents a global memory usage of 12800 MiB, and a minimum amount of recommended system memory of 16 GiB to simulate up to 256 UAVs simultaneously. However, with the SITL log disabled, and even when using the RAM drive, the overall memory size requirement is reduced to 1280 MiB, and so the recommended amount of memory to run ArduSim is reduced to only 6 GiB. If the execution is not performed as root, and therefore a RAM drive is not used, the memory consumption is further reduced to 1400 MiB both for the SITL instances and the simulator itself, and so a minimum of 4 GiB of RAM suffices.

4.2.1 CPU utilization

The CPU load is a critical factor affecting the scalability of ArduSim since the execution of each virtual UAV will be slowed down if the CPU cannot manage, in real time, all the necessary calculations. We carried out experiments for different numbers of UAVs while varying the CPU usage associated to the graphical environment and the computer used, and introducing as a synthetic load the transmission of coordination information between the UAVs. Each experiment, lasting 5 minutes, was repeated three times measuring CPU usage once a second, and then we took the average value.

4.2.1.1 Rendering quality overhead

Figure 4.18 shows the CPU usage with different numbers of UAVs, and when using four different rendering qualities in the *i7 PC* used for testing:

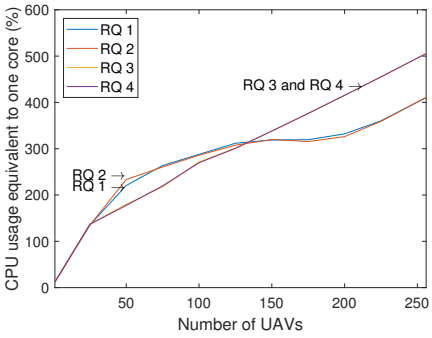


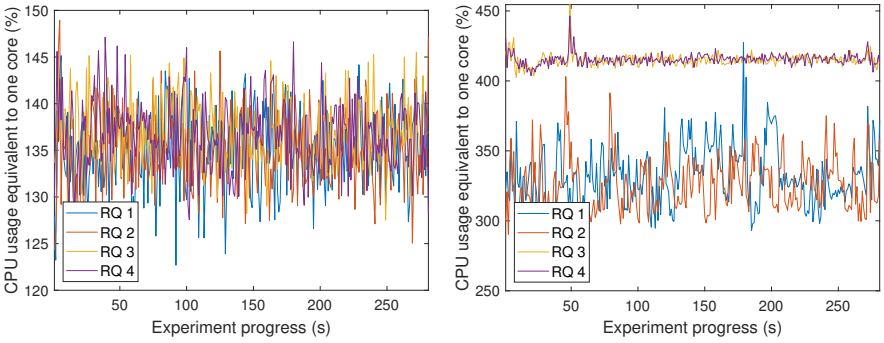
Figure 4.18: Rendering quality overhead *(i7 PC)*.

- *RQ 1*. Lowest level with maximum performance.
- *RQ 2*. Fonts smoothed. Font antialiasing enabled.
- *RQ 3*. Lines smoothed. Font antialiasing enabled, and lines with sub-pixel accuracy rendering.
- *RQ 4*. Same as *RQ 3*, and alpha blending optimized for quality.

We can see that only the use of line rendering with sub-pixel accuracy (i.e., *RQ 3* and *RQ 4*) is significant in terms of performance. In addition, in the *RQ 1* and *RQ 2* levels, we find that the processor's energy saving features apparently increase the CPU usage when the load is low, that is, with less than 200 UAVs, moving away from the theoretical line that would connect CPU usage with a single UAV and with 256 UAVs. This effect takes place since the processor goes into inactivity for short periods of time, and the execution of different threads overlaps in time. This effect is also observed in Figure 4.21, when additional CPU load is added by enabling inter-UAV communications. The maximum load with 256 UAVs is approximately 500%, when the maximum possible value is 800% (4 cores with Hyper-Threading can run 8 threads simultaneously).

Figure 4.19 shows the evolution of CPU usage during the experiment with 25 and 200 UAVs. In Figure 4.19a, the values oscillate significantly and randomly, since the processor is far from being saturated, and there are even time intervals during which the CPU is inactive. However, with 200 UAVs (see Figure 4.19b), we observe that, when line drawing with sub-pixel accuracy is activated, the CPU load increases and causes threads to start having to wait for execution, a situation that causes CPU usage to become more uniform.

Figure 4.20 shows the CPU usage with the 4 rendering quality levels, in this case for the *i5 PC*. It confirms that only two sets of rendering quality combinations are significant, differentiated by the use of line drawing with subpixel accuracy (*RQ*



(a) Results running 25 UAVs.

(b) Results running 200 UAVs.

Figure 4.19: CPU utilization when varying the rendering quality overhead *(i7PC)*.

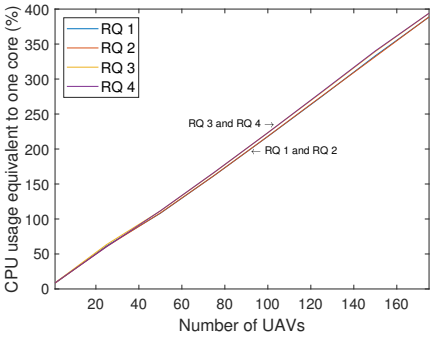


Figure 4.20: Rendering quality overhead *(i5PC)*.

1/RQ 2 and *RQ 3/RQ 4*), although the difference between both levels is very small. In this case, the system is not capable of simulating more than 175 UAVs without destabilizing, since the CPU usage reaches 400 %, the maximum possible one supported (4 cores can run 4 threads simultaneously). In addition, the time lag introduced when simulating 175 UAVs is excessive, which is why, in Figure 4.25, its magnitude is analyzed only with up to 150 UAVs. Notice that, since the CPU is less powerful, it shows a linear resource consumption increase with the number of UAVs, and it does not show the same effects detected in the i7 platform, associated to energy saving mechanisms.

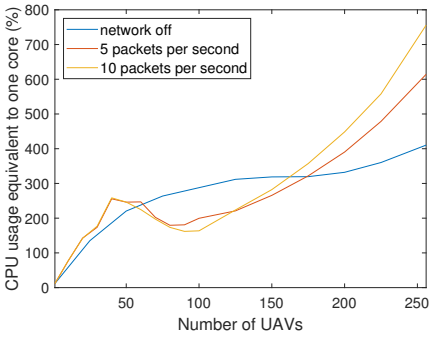


Figure 4.21: UAV-to-UAV communications overhead *(i7PC)*.

4.2.1.2 Communications overhead

ArduSim has been designed to develop and validate communication protocols between UAVs. Thus, the CPU usage analysis must also take into account the load that communications between UAVs introduces. For this purpose, two experiments have been designed on the *i7 PC* by varying the network load. In both cases, all the simulated UAVs transmit data packets at a constant rate, and follow an overlapping trajectory, meaning that nearly all the packets reach all the UAVs, a situation that we consider the most unfavorable for our analysis, since it is associated with a higher CPU load. The first experiment was done with a sending rate of 5 packets per second, while the second one was done with a rate of 10 packets per second. In both cases, the transmitted packet size is 705 bytes. Figure 4.21 shows that CPU usage grows faster with up to 40 UAVs. With more UAVs, the load causes the energy saving mechanisms of the i7 PC to lose significance, and the curves adjust better to the theoretical line connecting the results for 1 and 256 UAVs. However, the load introduced by the communications prevents the graph from being straight, which is consistent with the computational cost of sending and receiving data packets for the synthetic load used ($\mathcal{O}(2n^2)$, being n the number of UAVs).

4.2.2 Real-time constraints evaluation

Merely checking that the simulator is stable, and that the processor does not become saturated, is not enough to state that ArduSim correctly emulates the UAV flight in real-time. In fact, a very high or irregular CPU usage could introduce a global delay in the execution of the emulated UAVs, or even a differential delay between them, thereby affecting the scalability of the simulator. Therefore, the maximum number of UAVs that can run simultaneously on a given computer will depend on these factors.

To analyze the scalability of ArduSim, experiments were carried out on both the *i7 PC* and the *i5 PC*, with the rendering quality set to *RQ 1*, with a different number of UAVs, and measuring the time lag for each of the simulated UAVs regarding the simulation of a single UAV used as reference. In other words, considering the real-time performance of a single-UAV simulation, we analyze if the UAVs suffer any kind of lag among them, and also in regard to that single-UAV simulation. Each experiment was repeated 7 times, measuring the lag error every 5 meters throughout the followed path. We assessed the individual results of each repetition, or the overall set of results as a single group, depending on the analysis carried out. All the experiments were performed with the UAVs following a straight path trajectory from origin to destination for 5 minutes. All trajectories are overlapped, and UAV collision detection is deactivated. The flight altitude was set to 5 meters, and their speed was of 10 m/s; the default values of the simulator were used for the remaining parameters.

Regarding the single-UAV experiment used as reference, it was selected among the 7 repetitions as the one located nearest to the median value.

4.2.2.1 Scalability analysis under the i7 PC

Figure 4.22 shows a box-whisker plot with the time lag of the UAVs in regard to the reference UAV path, obtained when varying the number of UAVs. In addition to the median lag and the distribution of the lag values for all the simulated UAVs obtained along 7 experiments, it includes the mean lag value. When simulating 100 or less UAVs, the measurements shown are really close to the mean value; however, for a higher number of UAVs, data is more scattered, and the time lag increases. Similarly to previous results (see Figure 4.18), the processor's energy-saving mechanisms introduce a significant time lag for a number of UAVs between 150 and 225. The worst case is detected with 200 UAVs, with an average lag value of 0.45 seconds, and a maximum lag of 1.4 seconds. On the other hand, the dispersion of values with up to 100 UAVs is significantly lower than the one obtained with a higher number of UAVs.

Figure 4.23 shows the evolution of the mean value for the time lag corresponding to the 7 experiments performed with each number of UAVs. The time lag remains significantly constant throughout time for up to 100 UAVs (a mere increase of 0.5 seconds per simulated hour), and it increases linearly with more UAVs. As shown in Figure 4.22, the processor's energy-saving mechanisms introduce a significant time lag for a number of UAVs between 150 and 225, which makes the slope for 200 UAVs to be actually higher than the slope for 256 UAVs. We can conclude that the *i7 PC* allows us to simulate up to 100 UAVs while meeting soft real-time constraints.

Now, a detailed analysis of the experiment that produces the greatest time lag with regard to the reference UAV (i.e., with 200 UAVs) is presented. Figure 4.24a shows the set of time lag values obtained in the 7 experiments performed. From

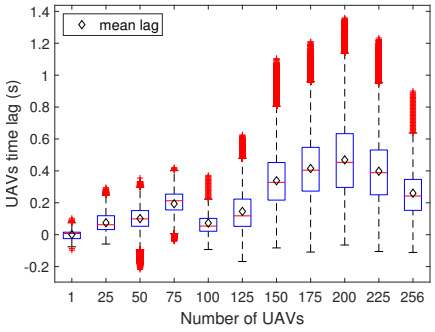


Figure 4.22: Time lag values for all the experiments *(i7PC)*.

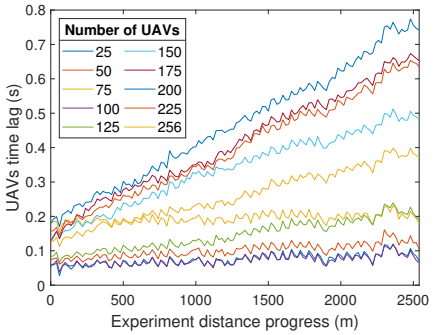
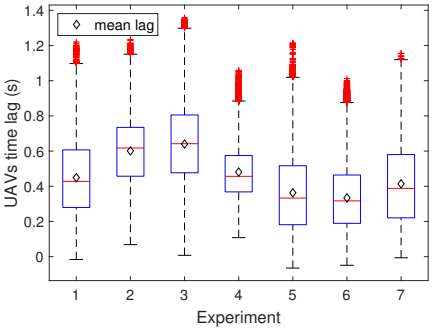
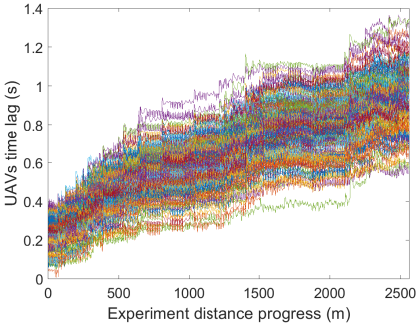


Figure 4.23: Time lag throughout the experiment *(i7 PC)*.

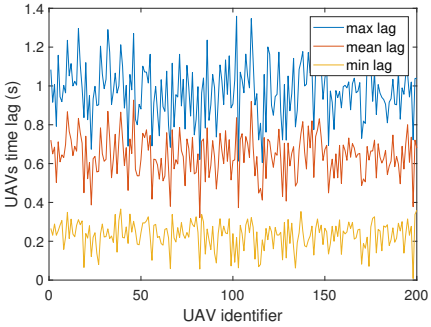
that figure we find that the worst case is test number 3, with a maximum time lag of 1.4 seconds, and a mean lag value of 0.62 seconds. Also, Figure 4.24b shows the evolution of the time lag of each UAV in that test. We can see that, after an initial warm-up period (first 700 meters), the time lag between UAVs stabilizes. In addition, this lag is always greater than zero and it increases throughout time, evidencing that UAVs suffer a delay with respect to the reference UAV. Figure 4.24c shows the average, minimum, and maximum lag for each UAV in that same test. Notice that UAV number 103 is the one with the highest time lag. In addition, the average lag for each UAV varies in a very small range of 0.6 seconds. Thus, we can state that, although the simulation does not meet strict real-time constraints, it can be considered to be correct as long as the absolute simulation time is not relevant for the protocol under development, since the simulation delay offset associated to the different UAVs remains similar.



(a) Time lag values of each experiment.



(b) Experiment 3 (worst case). Time lag of all the UAVs.



(c) Experiment 3 (worst case). Minimum, mean, and maximum time lag for each UAV.

Figure 4.24: Worst case analysis with 200 UAVs *(i7PC)*.

4.2.2.2 Scalability analysis under the i5 PC

The previous tests have been performed on a high-end desktop computer with very high performance. Thus, we consider adequate to complement our analysis by also checking the time lag associated to PCs with lower performance. In particular, this section details the results obtained with the *i5 PC* used for testing.

The results in Figure 4.25 are limited to 150 UAVs, since with more than 175 UAVs the simulation becomes unstable, and with 175 the time lag (14 seconds) is too high, something consistent with the results in Figure 4.20, where the CPU became saturated with 175 UAVs.

In the *i5 PC*, the time lag remains stable over time for up to 100 UAVs (see Figure 4.26). There is also a temporary reduction in CPU usage towards the end of the experiment, when the UAVs approach the last waypoint of the mission, just before landing. The initial lag has a bias deviation between 0 and 0.2 seconds

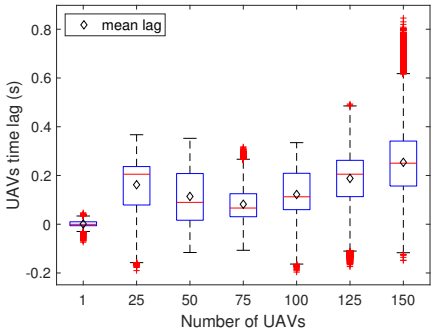


Figure 4.25: Time lag values of all the experiments *(i5PC)*.

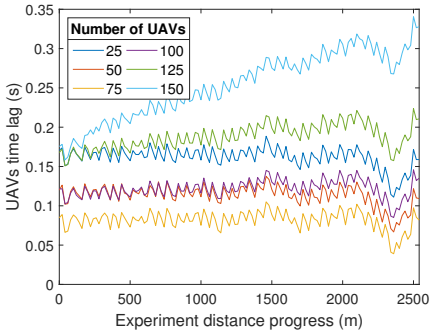
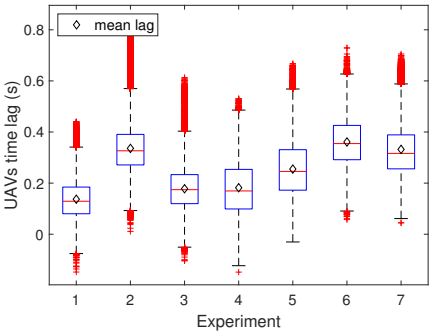


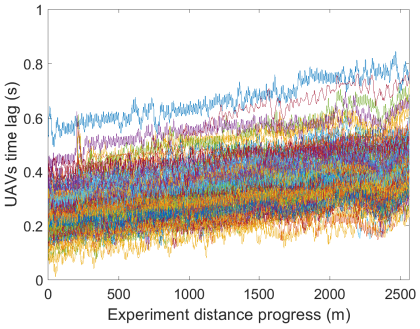
Figure 4.26: Time lag over experiment progress *(i5PC)*.

because the UAVs start their flight at different instants on each experiment. Notice that this issue does not affect the real-time execution, and is due to control loops included in the implementation.

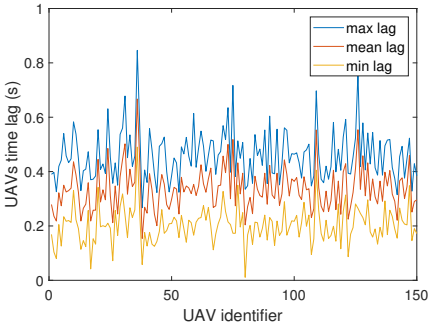
Similarly to the experiments made with the *i7 PC*, we include a study of the maximum temporal lag achieved (see Figure 4.27). First, in Figure 4.25, we can observe that the maximum time lag is of 0.85 seconds, and it corresponds to experiments with 150 UAVs. More specifically, it corresponds to the maximum lag of the second experiment (see Figure 4.27a). Figure 4.27b shows that the time lag between UAVs remains uniform throughout the test, similarly to Figure 4.24b. Finally, in this case, the maximum time lag detected is associated to UAV 36 in the experiment (see Figure 4.27c). There is greater variability in the average lag of each UAV due to a greater dispersion in the time lag of each UAV. This occurs because the processor is closer to saturation compared to the situation where the *i7 PC* is used.



(a) Time lag values of each experiment.



(b) Experiment 2 (worst case). Time lag of all the UAVs.



(c) Experiment 2 (worst case). Minimum, mean, and maximum time lag for each UAV.

Figure 4.27: Worst case analysis with 150 UAVs *(i5 PC)*.

4.2.2.3 Communications overhead analysis

This set of experiments was carried out with a load of 5 packets per second per UAV. This means that, with 200 UAVs, 1000 packets per second are sent, and so potentially 199,000 messages reception events per second can take place when broadcasting these packets. In such case, the associated CPU usage becomes high, as shown in Figure 4.21, which can negatively affect the scalability of the simulator, as the temporal offset of each UAV with respect to the reference may increase.

Figure 4.28 shows that the measured time lag with a number of UAVs between 150 and 225 is lower than the one depicted in Figure 4.22. This occurs because the processor is working at full capacity, without activating the energy saving features referred to earlier. However, with 256 UAVs, the lag is greater; in this case CPUs operate close to their saturation point (see Figure 4.21).

Similarly to previous cases, the evolution of the time lag throughout the ex-

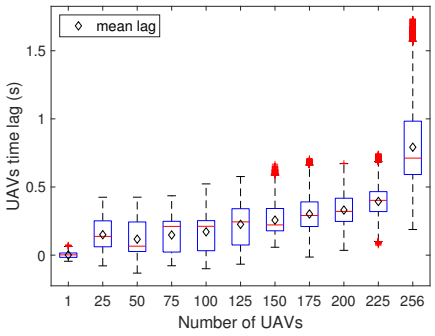


Figure 4.28: Time lag values of all the experiments at a sending ratio of 5 pps *(i7 PC)*.

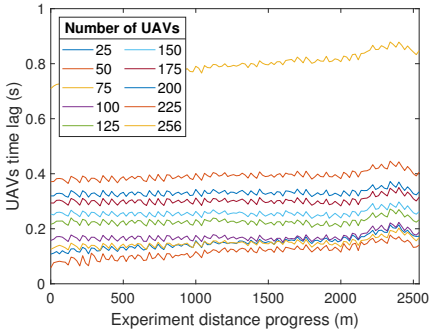


Figure 4.29: Time lag over experiment progress *(i7 PC, 5 pps)*.

periments has been studied. Figure 4.29 shows a peak of CPU usage towards the end of the test, when the UAVs reach the last waypoint and reduce their speed. It is also observed that the time lag remains stable over time with up to 175 UAVs, increasing linearly for higher number of UAVs. These results are better than those obtained before activating communications between UAVs, and occur because the CPUs are already working at their full capacity when reaching 90 UAVs (see Figure 4.21).

Another issue to be discussed is the accuracy with which the simulation of communications is able to detect the usage of the wireless medium (carrier sense), as well as the collision between data packets. Table 4.3 shows the results obtained when varying the number of UAVs while setting the transmission load to the value defined above (5 packets per second per UAV). It is observed that the results are somehow optimistic since, with 200 UAVs, 1000 packets sized 705 bytes are being transmitted at a speed of 6 Mbps (transmission time of $\approx$ 1 ms), which should saturate the medium. Although nearly all packets have to wait for the

Table 4.3: Percentage of packets that waited (carrier sense) and collided (collision detection).

	25	50	75	100	125	150	175	200	225	256
CS	0.84	3.40	4.02	4.25	21.30	60.01	79.29	92.07	94.92	94.82
CD	0.08	0.06	0.10	0.13	0.56	3.20	8.69	15.09	17.52	20.29

medium to become available, the collision rate remains lower than expected. To explain this phenomenon, we must take into account that the CPU of the *i7PC* can only run 8 threads at a time, while with 200 UAVs there are 800 threads / processes (one SITL process and three threads, *Listener*, *Talker*, and *Controller*, per UAV) that are competing for CPU time, which implies that there are 100 threads/processes that compete for being executed on a same core. On the other hand, the time slices provided to each thread by the Linux/Ubuntu operating system varies between 0.75 ms *(sysctl_ sched_ min_ granularity)* and 6 ms *(sysctl_ - sched_ latency)*, meaning that each thread can run about 13.3 times per second during 0.75 ms, or, what is the same, each *Talker* thread can send a packet once every 75 ms. Considering that, in the experiment performed, the transmission time of each packet is approximately 1 ms. We find that, in a worst-case scenario, it can only collide with packets sent by other *Talker* threads that are running when these have just left the processor, or will run in the next round. This behavior avoids that data packets collide with each other as often as expected, producing a packet collision rate lower than what would actually occur. Thus, this problem is inherent to the system itself, and can only be improved by using dedicated servers with a very high number of cores.

4.3 Summary

In this work we introduced ArduSim, a realistic simulation platform that allows operating with multiple UAVs simultaneously when performing planned missions, or when flying as a swarm. To date, no similar solution has been developed that offers similar characteristics, including the possibility to model inter-UAV communications using different channel models, as well as the way UAVs use the exchanged information to interact between them, paving the way for introducing a wide range of novel protocols. Through simulation, it becomes possible to analyze packet dissemination in DTNs, to develop new routing algorithms, or to propose new flight coordination mechanisms. Among its many benefits, our simulator allows validating the proposed solution beforehand, while porting that solution to real devices becomes straightforward, as the set of commands used is the same.

In addition to the simulation platform itself, we also modeled the WiFi com-

munications link between UAVs based on real experiments performed in the 5 GHz frequency band. In particular, we focused on the relationship between packet losses and distance when broadcasting data. The derived model was then integrated into the simulator as one of the wireless channel models available. To further improve the degree of realism of our experiments, we also modelled the wireless channel occupancy through a carrier sensing mechanism, and included the possibility of detecting collisions of data packets.

The stability of ArduSim has been correctly validated, with up to 4000 different successful executions, being stable with 256 UAVs in the *i7 PC*, and up to 175 UAVs in the *i5 PC*. We found that any mid-range or high-end computer is capable of simultaneously simulating a high number of UAVs (approximately 100) in near real-time, even when considering the overload introduced by the communications between UAVs.

Regarding scalability, we have verified that the simulation can be performed with up to 100 UAVs while meeting soft real-time constraints, and that the delay offset between them is uniform. In addition, ArduSim is able to run at least 150 UAVs when hard real-time is not required in a computer similar to the *i5 PC*, or even 225 with a high-end desktop computer *(i7 PC)*. We have also analyzed the influence of the rendering quality on the system load. We found that only the tracing of lines with sub-pixel quality has a significant effect on performance. Communications have a quadratic computational cost, so they also affect the system performance significantly. When configuring each UAV to transmit at a rate of 5 packets per second, the load affects real-time performance when having more than 225 UAVs in the *i7PC*. Also notice that, depending on the complexity of the UAV coordination protocol being developed, its impact on performance can also become non-negligible.

Chapter 5

Mission Based Collision Avoidance Protocol (MBCAP)

The adoption of UAVs to perform a multitude of tasks is raising concerns about privacy, security and flight safety [50], especially in urban environments where the consequences of any flight disruption are typically much more severe due to the risks of injuries for citizens. To address this issue, several efforts are taking place worldwide to make UAV flights safer. For instance, in Europe, U-space [48] is an initiative that aims at making UAV traffic management safer and more secure. In particular, U-space attempts to provide an appropriate interface with manned aviation and air traffic control so as to facilitate any kind of routine mission, in all classes of airspace, and even in congested environments like urban areas, so as to achieve the ambitious Single European Sky (SES) goal. The SESAR Joint Undertaking [60] was set up in order to manage this large scale effort, coordinating and concentrating all EU research and development activities focused on Air Traffic Management. This way, a wide range of drone missions that are currently being restricted will be possible thanks to a sustainable and robust European ecosystem that is globally interoperable.

Among the different areas where UAV flight safety is being considered, there is a particular area that has not yet been fully addressed: the development of sense & avoidance mechanisms to enable an UAV to become aware of its environment, allowing it to take evasive action if necessary [51]. In this chapter we focus on this problem by proposing the Mission Based Collision Avoidance Protocol (MBCAP), a collision avoidance solution that relies on wireless communications between nearby UAVs performing planned missions.

Experimental results using real UAVs, along with large-scale simulation experiments, validate the effectiveness of our proposed protocol, and evidence the low overhead introduced both in terms of channel occupation and mission delays.

5.1 Protocol overview

5.1.1 Introduction

MBCAP is a collision avoidance protocol applicable to UAVs following a planned mission in an autonomous manner, and issue not adequately addressed by the research community. To this aim, it relies on a cooperative sensing approach whereby multicopters broadcast their own location and predicted future locations. Upon receiving these data, receivers rely on them to decide if there is a collision risk (collision detection), and to avoid the collision if necessary (collision avoidance). The strategy is based on priorities, where a UAV has always a lower or higher priority than any other UAV it could meet during a flight. The high-priority UAV will be the first to resume its mission, and the low-priority UAV will wait the needed time to avoid the collision, only resuming its mission afterward.

Several technologies could be used to establish a communication link among the UAVs. ADS-B [1] could be a good solution, but it requires infrastructure, and uses proprietary technology and restricted frequencies. Our solution assumes the use of IEEE 802.11a wireless adapters operating in Ad-hoc mode, an open and cheap solution already available in the market.

Regarding the architecture of the protocol, it comprises three threads for each UAV, i.e. *Beaconing*, *Listener*, and *CollisionDetector*. The *Beaconing* thread periodically sends UDP broadcast datagrams with the current location of the multicopter, followed by a list of future predicted locations, including spatial and temporal coordinates. Such data is enough to detect collision risks with other UAVs. The *Listener* thread receives and stores the most up-to-date information received from other UAVs. Finally, the *CollisionDetector* thread periodically checks the gathered data, and compares the future predicted locations with the ones advertised in its own beacon to decide if there is a collision risk with another UAV. In that case, it stops the multicopter and relies on the protocol to address the risky situation. The high-priority UAV resumes the mission when the other multicopter is ready to be overtaken. The low-priority UAV resumes the mission once the other one is in a safe location. Furthermore, if the low-priority UAV stands still in the path of the high-priority UAV, before giving way it moves aside to let the UAV pass through its current location. This protocol has been mainly designed to avoid collisions between two UAVs, as the probability of more than two UAVs performing planned missions to meet each other all at once is very low. In case a third UAV detects a collision risk with any of the contending UAVs, it will stop and wait for the previous collision risk to be solved before executing the protocol.

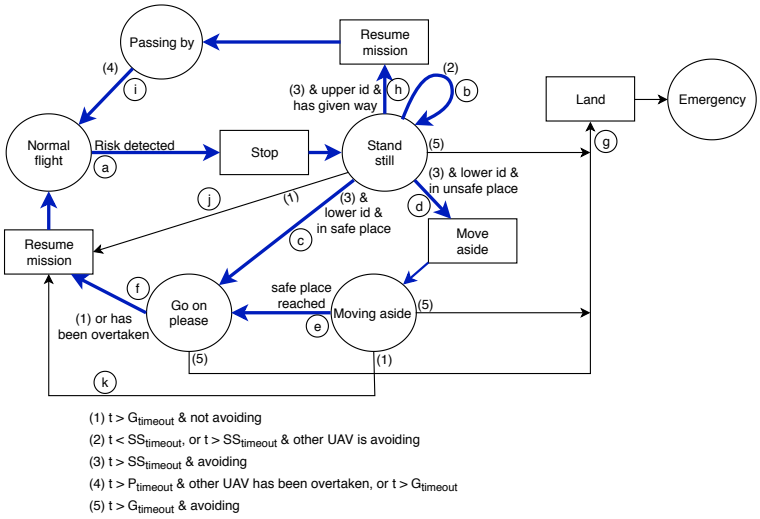


Figure 5.1: MBCAP finite state machine.

The collision avoidance strategy is based on priorities at the time of deciding which UAV can go on with the mission. For this purpose, all the multicopters must have a unique identifier (ID) which enables us to establish an ordered relation among them. We use the unique ID value provided by ArduSim, defining the high-priority UAV as the one with the higher ID value. This value is an unique random ID for each UAV when you run ArduSim as a simulation environment, and an unique ID based on the MAC address of the wireless adapter when you run ArduSim on a real multicopter.

5.1.2 Finite state machine

In this section we detail the finite state machine that regulates the behavior of MBCAP, which is implemented in the *Collision Detector* thread (see Figure 5.1). In the figure, the circles represent the machine states, the rectangles represent the commands sent to the flight controller to change the behavior of the UAV, and the arrows represent the transitions between states. The blue thick arrows are related to the most common scenario where only two UAVs are involved.

We will proceed to describe the most common situation addressed by MBCAP, where two UAVs meet and a collision risk is detected. Depending on the ID value, the UAVs can be in any of the following cases:

1. *Lower priority UAV*. It starts in the *Normal flight* state. When a collision risk is detected, it needs a few seconds to stop in the air and enter in the

Stand still state (transition *a*). Then, it will wait for a short time $SS_{timeout}$ (transition *b*) to ensure that the high-priority UAV has also reached the same state. When the other UAV informs that it is in the same state, it analyzes if it finds itself in the route the high-priority UAV was following. If not, it is safe to continue, and the UAV changes to the *Go on please* state (transition *c*), allowing the other UAV to continue. On the contrary, it calculates where to move aside, and switches to the corresponding state (transition *d*), moving until it reaches the target location, and changing to the *Go on please* state (transition *e*), as in the previous case. When the high-priority UAV moves beyond the area of conflict, the UAV resumes the mission (transition *f*) to exit the protocol, as the collision has been avoided.

2. *Higher priority UAV*. It also starts in the *Normal flight* state, and changes to the *Stand still* state (transition *a*) when a collision risk is detected. Then, it waits (transition *b*) for the same timeout and until the lower priority UAV gives it way, resuming the mission (transition *h*), and changing to the *Passing by* state. Afterward, during the overtaking process, the high-priority UAV approaches the low-priority UAV; the overtaking ends when the former detects that the distance between them is increasing. Immediately, it informs the low-priority UAV that it can continue with the mission, and it simultaneously switches to the *Normal flight* state (transition *i*).

We have implemented MBCAP so as to be resilient to unexpected situations, adding additional transitions and the *Emergency* state. Thus, if a UAV is in a state different from *Normal flight*, and a global timeout elapses, we can find two cases: (i) if the UAV is not receiving messages from the UAV it is contending with, which means that there is no risk of collision, the mission is resumed (transitions *f*, *i*, *j*, *k*); otherwise, (ii) the UAV lands (*emergency* state) if the other UAV is close enough and the protocol has failed. The *Passing by* state is a special case where the UAV resumes the mission instead of landing, because the low-priority UAV has moved aside, if necessary, and there is no collision risk.

When a third UAV detects a risk of collision with one of the UAVs that are in the process of solving a collision situation, the protocol causes it to stop (transition *a*), and to wait in the *Stand still* state (transition *b*) until the previous risky situation is solved. Afterward, the protocol is applied between the two UAVs in risk of collision.

5.1.3 Beacon content

MBCAP is a protocol where the decisions are taken considering the state information sent by the different UAVs using beacons; these beacons are periodically broadcasted using UDP datagrams.

The beacon transmitted in MBCAP (see Figure 5.2) includes the following fields:

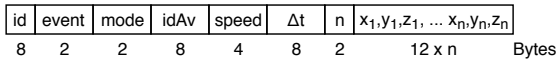


Figure 5.2: Periodic beacon content.

- *id*. Unique identifier of the sender UAV.
- *event*. Number of risky situations previously solved. The low-priority UAV resumes the mission when the high-priority UAV finishes the overtaking process and increases the value of this field.
- *mode*. Flight mode, equivalent to the current state in the finite state machine (see circles in Figure 5.1).
- *idAv*. Identifier of the neighbor UAV with which the UAV is avoiding a collision, if applicable.
- *speed*. Current ground speed (m/s).
- Δt. Time elapsed from the time the beacon information was generated until it has been transmitted; predicted future UAV locations are not recalculated for each beacon to avoid consuming excessive resources.
- *n*. Number of predicted future locations included in the beacon.
- *Predicted locations array*. 3D Universal Transverse Mercator (UTM) coordinates for predicted future locations.

The array containing the future locations sent by each UAV in a beacon includes different information depending on its state. In particular, it will include the following information:

- Moving at low speeds ($< 1m/s$). Only the current location is broadcasted.
- *Go on please* state. The current location and the location where the risk of collision was detected.
- *Moving aside* state. The current and the future locations towards the safe position the UAV is moving to.
- *Stand still* state. The current location and the set of waypoints not yet visited, conforming the information used to determine if the UAV should move aside to give way for a higher priority UAV, as detailed in section 5.1.5.
- *Normal flight* state. The current location and future locations, used to detect a risk of collision with other UAVs, typically 50 locations, which corresponds to a beacon of 634 Bytes.

5.1.4 Collision risk detection strategy

A significant difference between a UAV manually controlled and a UAV following a mission is the fact that we can predict where the latter will be in the future, as it tries to follow a predefined path.

The strategy adopted to detect a collision risk between two UAVs consists of predicting the future locations of the UAV given its current location, the waypoints it is moving towards (the remaining mission), and the current speed. UAVs broadcast their future locations and periodically compare the received locations with their own predicted locations. If they match in both space and time, a collision risk is detected, and the UAVs stop. A match in space happens when the horizontal distance between the two UAV predicted locations is lower than 20 meters, and the vertical distance is lower than 50 meters. On the other hand, a match on time happens when the two predictions are within the same half second. Furthermore, the UAVs do not stop until the distance between that risky location and the UAV itself is less than a predefined threshold distance (90 m by default). Such optimization attempts to avoid that UAVs stop when they are still far away, a situation that would unnecessarily extend the time required to solve the collision scenario.

MBCAP bases its collision avoidance effectiveness in predicting with enough accuracy the position that each UAV will have in the near future.

To check the accuracy of the mechanism used to predict future positions, several experiments were performed using a single virtual multicopter, and measuring the distance between predicted points (advertised in beacons) and the actual UAV location at the predicted time. Each experiment was repeated three times, and the most unfavorable results were considered. Notice that the global experiment time between runs varies by less than one second. The UAV had a programmed speed of 15 m/s, and followed a route composed by two perpendicular segments, thereby representing very unfavorable conditions since it is the maximum speed supported in mission-driven flights on common multicopters, and also because it is a very pronounced turn.

We developed three MBCAP alternatives depending on the strategy used to predict the future locations of the multicopter, each one improving upon the previous:

- MBCAP alternative 1. The future UAV positions are calculated using the route to be followed, and the current flight speed in m/s, according to the information provided by the flight controller.

- MBCAP alternative 2. Additionally, the current and predicted UAV positions are projected over the theoretical path that it should be following.

- MBCAP alternative 3. The current UAV acceleration a_i (m/s^2), based on

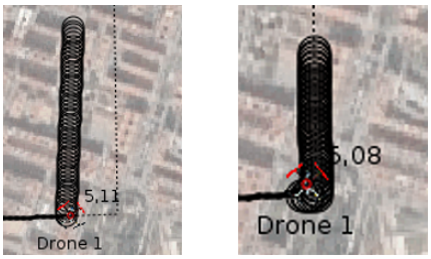


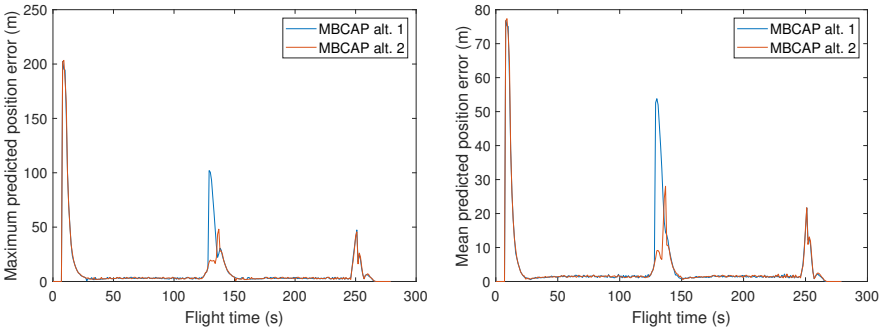
Figure 5.3: Curve error in MBCAP alt. 1.

the speed advertised by the flight controller, is also included in the calculation.

It must be taken into account that the controller determines that the UAV should move towards the next waypoint not at the time it actually reaches that waypoint, picking instead an earlier time instant, as shown in Figure 5.3 (left), which corresponds to MBCAP alternative 1, thereby allowing to achieve a smooth change of course. This is the standard behavior of any flight controller, and it introduces an error in the calculation of future positions that can be significant when the turn is too abrupt. MBCAP alternative 2 corrects this error by projecting future positions on the theoretical path to be followed by the UAV based on its current position, as shown in the second image.

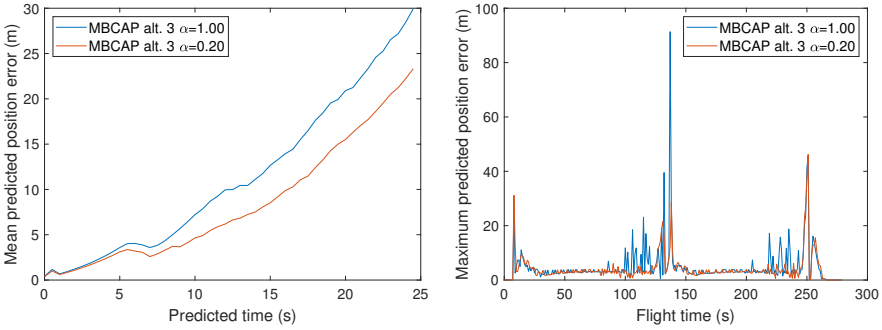
Figure 5.4a shows the maximum error on MBCAP alternatives 1 and 2, for the predicted positions embedded in beacons during the course of an experiment, while Figure 5.4b shows the mean value of the error. In both cases, only those positions that are at a distance that is less than or equal to the threshold below which the collision risk detection is applied (default 90 m) are considered. Both figures show a similar behavior, with a uniform maximum error between 3 and 4 meters when the UAV traverses each segment of the mission at a constant speed, and a much greater error when (i) starting to move, (ii) stopping, and (iii) taking the curve between two segments. As already mentioned, alternative 2 of MBCAP corrects the deviation corresponding to the curved path introduced by default by the flight controller. However, high error values still take place when the UAV experiences speed changes (in this case, the beginning and the end of the mission).

As already stated, MBCAP alternative 3 calculates the future positions of the UAV taking into account the current acceleration a_i. Such acceleration is estimated constant throughout the flight time included in the prediction. In addition, since we observed that the calculated acceleration varies significantly, we decided to apply a filter to the obtained value (see equation 5.1). Figure 5.5 allows us to compare the effect of the filter ($\alpha = 0.2$) against the results obtained when it does not apply ($\alpha = 1.0$). We observed that the filter significantly improves the quality of the prediction. Figure 5.5a shows the average error corresponding to all



(a) Maximum predicted location error. (b) Average predicted location error.

Figure 5.4: MBCAP alt. 1 vs. alt. 2: Predicted location error vs. flight time.



(a) Filter effect on each of the predicted locations.

(b) Filter effect: Maximum predicted location error.

Figure 5.5: MBCAP alt. 3: Filter effect.

beacons in each of the predicted instants, while Figure 5.5b shows the maximum error associated to each beacon. Hereafter, those figures showing mean values are not included since their behavior is very similar to the maximum value behaviour in all cases.

$$a = \begin{cases} 5 & \text{if } a > 5, \\ -5 & \text{if } a < -5, \\ 0 & \text{if } |a| < 0.1, \\ \alpha \cdot a_i - (1-\alpha) \cdot a_{i-1} & , \alpha \in]0,1] \text{ otherwise.} \end{cases} \tag{5.1}$$

Figure 5.6 compares the maximum error associated to each beacon on alternative 2 of the protocol with the maximum error on alternative 3 taking place when

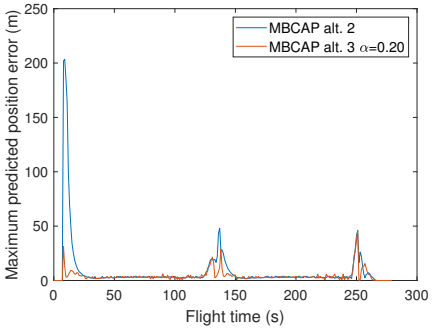


Figure 5.6: MBCAP alt. 2 vs. alt. 3: Maximum predicted location error.

$\alpha = 0.2$. The latter shows a drastic reduction in the estimation error associated to the three transitory situations, while the UAV adjusts its speed. In addition, the maximum error peaks now span a much shorter time interval, being of 2 seconds at startup, followed by two peaks of 6 and 5 seconds when making the turn, and of 13 seconds when landing, being these values clearly lower when compared to the ones achieved with alternative 2: 18, 26 and 15 seconds, respectively.

By default, beacons are sent 5 times per second, but the estimation of future UAV positions is made once every second. The only field updated on a later beacon based on the same information is the time elapsed since the beacon was generated (field Δt, see section 5.1.3). Taking into account that 25 seconds of flight time are included in each estimate, we could assume that the prediction error would be reduced when increasing the estimation frequency. However, we find that this does not occur. In Figure 5.7 we compare the maximum error associated to each beacon when updating the estimates once per second (default option), or when doing it each time a beacon is sent. As shown, having more frequent estimations does not cause the error to become lower, being that it even becomes worse in some cases, as occurs in the interval ranging from 95 to 130 seconds. The same effect is observed in all the implemented alternatives of the protocol, indicating that a calculation frequency of 1 Hz is sufficient, and it also implies a lower resource usage, being more convenient when considering that the protocol could be deployed in devices with reduced computation power.

The results obtained allow us to conclude that alternative 3 of the MBCAP protocol is the most accurate one, with a maximum error of 4 meters when estimating UAV positions under constant flight speeds, 31 meters for 2 seconds at the start of the mission, 28 meters for 2 seconds when turning 90 degrees, and between 27 and 46 meters for 3 seconds when braking, which occurs when reaching the last waypoint of the mission. On the other hand, the average error obtained is less than 2 meters at constant speed, up to 12 meters at the start of the mission, up to 13 meters at the turn, and up to 21 meters at the end of the mission. We con-

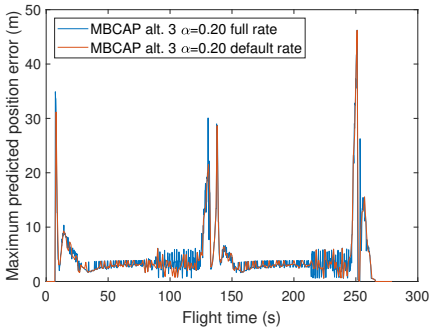


Figure 5.7: MBCAP alt. 3: Impact of predicted locations updating at full rate.

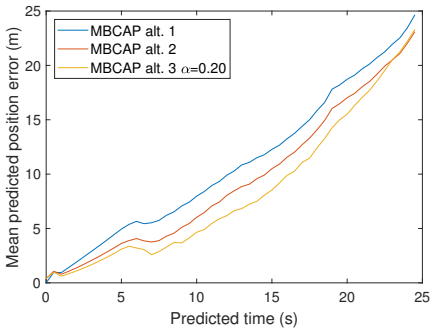


Figure 5.8: Average predicted location error for each of the predicted locations.

sider that the estimation error obtained is acceptable considering that: (i) most of the time the UAV will fly at a constant speed, thereby having a maximum error of 4 meters; (ii) the transient situation considered for the study is particularly unfavorable; and (iii) the duration of maximum error peaks is very small.

Figure 5.8 shows the average error of all beacons, for each of the predicted locations. It should be noted that, by default, the maximum distance at which collision chances are considered is 90 meters. Since the flight speed was of 15 m/s during most of the experiment, the predicted useful flight time under these conditions is 6 seconds. In Figure 5.8 we observed that, for up to 6 seconds of flight, alternative 3 is the one that offers the best results, and so we adopt and validate it as the final version of the protocol.

5.1.5 Collision avoidance strategy

The global idea behind the collision avoidance strategy is to define a right of way based on priorities among the UAVs. For this purpose, we use an unique number

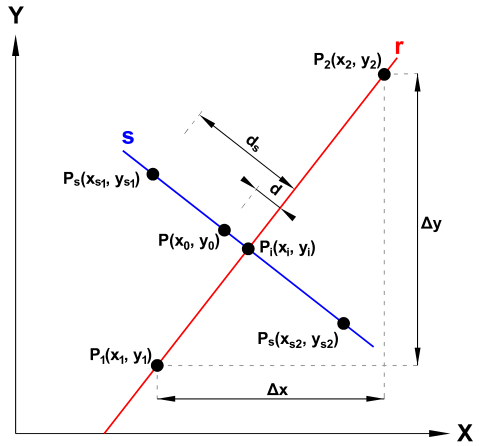


Figure 5.9: Safe location analysis.

(ID) based on the MAC address of the wireless adapter used on multicopters to communicate with each other. On the other hand, we use a random number provided by ArduSim when performing simulations. The UAV with a higher ID has right of way over the UAV with a lower ID.

Once a collision risk has been detected, both UAVs stop in the air and start the collision avoidance procedure. While they are in the *Stand still* state, they send their current location, and a list with the next waypoints of the mission they have to follow. Figure 5.9 shows how the low-priority UAV determines, based on such beacon information, whether it should move aside to allow the other UAV to pass by, which happens when it stands on the path the high-priority UAV must follow. This is achieved by determining if the distance between its current location $P(x_0, y_0)$ and each of the mission segments r (delimited by $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$), defined based on the locations advertised by the high-priority UAV, is higher than the threshold d_s. Notice that Δx and Δy refer to the relative increments on each axis according to the UTM coordinate system. Once the collision risk is detected, and to determine whether it is necessary for the lower-priority UAV to move to some specific location, it will follow these steps:

1) Determining the intersection $P_i(x_i, y_i)$ of line r (that contains the mission segment) with the perpendicular line s passing on point P:

$$P_i = (x_i, y_i) = \left(\frac{y_0 - y_1 + \frac{\Delta x}{\Delta y}x_0 + \frac{\Delta y}{\Delta x}x_1}{\frac{\Delta y}{\Delta x} + \frac{\Delta x}{\Delta y}}, y_1 + \frac{\Delta y}{\Delta x}(x_i - x_1)\right) \tag{5.2}$$

2) If P_i is within the mission segment, calculate the distance d between P and P_i.

3) It is only necessary to move aside from this mission segment if the intersection P_i is within the mission segment, and $d < d_s$. If d_{12} refers to the distance between P_1 and P_2, in other words, it is the length of the segment, then it is possible to calculate the coordinates for a safe location where to move (P_s) as follows:

$$P_s = \begin{cases} (x_{s_1}, y_{s_1}) = (x_i - \frac{d_s}{d_{12}}|\Delta y|, y_0 - \frac{\Delta x}{\Delta y}(x_s - x_0)) & \text{if } x_0 \leq x_i, \\ (x_{s_2}, y_{s_2}) = (x_i + \frac{d_s}{d_{12}}|\Delta y|, y_0 - \frac{\Delta x}{\Delta y}(x_s - x_0)) & \text{otherwise.} \end{cases} \quad (5.3)$$

If the low-priority UAV needs to move aside, it changes its location and gives way to the other UAV. Otherwise, it directly gives way. Then, the high-priority UAV starts the overtaking process, which finishes when (i) it goes beyond the collision risk region, and (ii) the distance to the low-priority UAV is increasing.

5.2 Protocol validation

Having introduced the MBCAP protocol, in this section we validate its correctness, taking as final version the proposed alternative 3. To this end, we performed a wide set of experiments using different scenarios to evaluate our solution. In these scenarios, UAVs approach each other in intersecting trajectories, considering different angles.

For our tests, we will refer to the low-priority UAV with the number 1, and to the high-priority UAV with number 2, meaning that the UAV 1 is the one moving aside whenever necessary.

The main metric used to validate MBCAP is the success ratio at avoiding collisions between UAVs, and the second metric is the flight time overhead introduced by the protocol while it is being applied; in other words, it is the difference between the time needed by the UAV to finish the mission when it has solved one or more collision risks, and the time needed to accomplish the mission when no other UAVs are present.

Each of the following experiments was repeated three times, considering the worst result in all cases:

- *Perpendicular crossing* (1). UAV 1 does not need to move aside.
- *Standard takeover* (2). Both UAVs follow a similar trajectory, although UAV 2 is approaching UAV 1 from behind at a higher speed. UAV 1 must move aside so that UAV 2 can takeover.
- *Face-to-face meeting* (3). This situation also requires UAV 1 to move aside in order to allow UAV 2 to pass without taking any risk.
- *Angled trajectories* (4). UAV 1 does not need to move aside.

Figure 5.10: Collision avoided on a crop field (scenario 6).

- *Angled trajectories, opposite directions* (5). Again, UAV 1 does not need to move aside.
- *Supervision of a crop field* (6). It simulates a real situation where a UAV approaches another UAV that is supervising a crop field sized 1500×900 m, going through it in a zig-zag fashion (see Figure 5.10), following lanes separated by 100 m. The two UAVs meet with perpendicular trajectories when UAV 2 is on its third pass over the crop field, and UAV 1 does not need to move aside.

During the experiments the flight speed was set to 10 m/s, except for scenario (2) where UAV 1 was flying at a lower speed (5 m/s), so that UAV 2 was able to takeover. The collision was avoided in all cases, and the time overhead (Δt) introduced by the protocol is shown in Table 5.1.

In general, we find that the flight time for the UAV with the highest priority was incremented between 19 and 29 seconds, while the UAV with the lowest priority experienced a time overhead ranging from 35 to 56 seconds, both cases being acceptable considering the current duration of standard multicopter batteries.

We also tested scenarios 1 and 2 with high wind speed (20 m/s) in order to verify if this factor affects the correctness and performance of the protocol. Table 5.2 shows that the overall flight time has no significant variation. Moreover, the time overhead introduced by the protocol is not affected by the wind speed.

Overall, we find that MBCAP avoids the collision in all cases, and the time overhead associated to the collision risk handling procedure is quite acceptable.

5.3 MBCAP-e: Enhanced Mission Based Collision Avoidance Protocol

In section 5.1 we propose MBCAP, a distributed protocol offering a cooperative sense & avoid solution. Experimental validation has confirmed that it is reliable, as

Table 5.1: MBCAP flight time overhead (min:sec).

Scenario	UAV	MBCAP: on			MBCAP: off	Δt
		Normal	Avoiding	Total	Total	
1	1	3:14	0:27	3:41	3:03	0:38
	2	2:57	0:24	3:21	3:03	0:18
2	1	3:41	0:47	4:28	3:35	0:53
	2	2:48	0:44	3:32	3:03	0:29
3	1	3:13	0:46	3:59	3:03	0:56
	2	2:48	0:44	3:32	3:03	0:29
4	1	3:14	0:26	3:40	3:03	0:37
	2	2:57	0:24	3:21	3:03	0:18
5	1	3:13	0:38	3:51	3:03	0:48
	2	2:50	0:36	3:26	3:03	0:23
6	1	10:20	0:26	10:46	10:11	0:35
	2	30:21	0:26	30:47	30:27	0:20

Table 5.2: Time overhead (min:sec) vs. wind.

Scenario	Wind orientation	UAV	MBCAP: on Total	MBCAP: off Total	Δt
1	Crosswind	1	3:41	3:03	0:38
	Headwind	2	3:31	3:13	0:18
1	Crosswind	1	3:41	3:03	0:38
	Tailwind	2	3:21	3:03	0:18
2	Headwind	1	4:28	3:35	0:53
	Headwind	2	3:41	3:12	0:29
2	Tailwind	1	4:26	3:33	0:53
	Tailwind	2	3:31	3:02	0:29

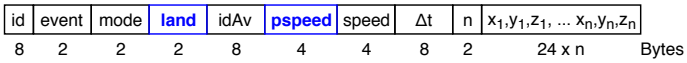


Figure 5.11: Periodic beacon content.

it is able to avoid collisions between two multicopters in all cases, and introducing only an acceptable overhead in terms of overall flight time. In addition, three alternatives were introduced for the mechanism predicting future UAV positions; experiments showed that the alternative 3, which considers the UAV acceleration on the prediction calculus, is the most accurate and adequate at predicting future positions.

In order to test and validate MBCAP under more complex situations we performed an extensive set of experiments with different numbers of UAVs, in a limited area, where more than two UAVs can meet at the same time. Results showed that the multicopters can collide under several circumstances, and also that some of them could be avoided by optimizing the protocol. In this section we propose MBCAP-e, an enhanced version of MBCAP which solves these problems. As the protocol has been already detailed in the previous sections, here we only detail the improvements made to the protocol.

5.3.1 Improvements overview

5.3.1.1 Beacon content

The data provided by the beacon sent by each flying multicopter showed insufficient to adequately detect collision risks, and to avoid the collision in some rare cases. The beacon transmitted in MBCAP-e (see Figure 5.11) adds the following fields:

- *land*. Whether the UAV started the landing phase. When a UAV reaches the end of the mission, it lands. MBCAP-e is not used while the UAV is landing because there are conditions under the landing procedure preventing it from being stopped.

- *pspeed*. Planned ground speed for the mission (m/s). This value is used in the collision avoidance strategy to assert if the high-priority UAV is close enough to a waypoint to increase the collision risk, as detailed later on section 5.3.1.3.

We have also reduced the number of predicted locations included in a beacon (see the following section), and at the same time, we have modified the format of the predicted locations, using *double* instead of *float* numbers, which increases the prediction accuracy.

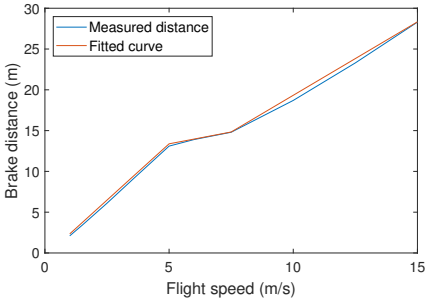


Figure 5.12: Measured brake distance vs. current flight speed.

5.3.1.2 Collision risk detection strategy

MBCAP-e includes several improvements over the protocol version detailed in the previous chapter. We now proceed to detail the most relevant one, which is related to the number of future predicted locations included in each beacon. The previous version of the protocol (MBCAP) sent 50 predicted locations within each beacon, which corresponds to 25 seconds in the future, considering to detect a possible collision risk only with locations that are less than 90 meters away from the current location of the multicopter. This configuration is prone to stop the UAVs too soon when their speeds differ significantly. In MBCAP-e we just send the necessary amount of locations to detect a collision risk considering the current speed of the UAV (see Equation 5.4).

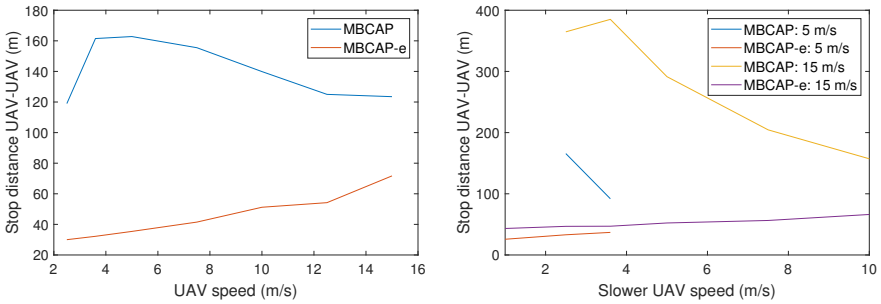
$$d = d_{GPS} + d_{brake} + d_{react} + d_{comm} \tag{5.4}$$

Where:

$d_{GPS} = 2.5\ m$
$d_{brake} = f(speed)$
$d_{react} = 1\ s \times speed$
$d_{comm} = 2\ s \times speed$

We consider that there is a collision risk if the distance d between the UAV and the location where a collision risk is detected is lower than the sum of: (i) the GPS error *(d_{GPS})*, (ii) the distance required to brake *(d_{brake})*, being that the latter depends on the current UAV speed (see Figure 5.12), (iii) the distance traveled between two collision risk checks *(d_{react})*, and (iv) the distance traveled throughout a predefined time when considering that some messages can be lost during transmission *(d_{comm})*.

Given the safety distance d, we calculate the total prediction time to be included in beacons as the safety distance divided by the current speed, and the



(a) UAVs flying at the same speed. (b) UAVs flying at different speeds.

Figure 5.13: Distance between UAVs after stopping, in a face-to-face meeting.

number of locations to include in the beacon as the total prediction time divided by the time elapsed between two predicted locations (500 ms in the default configuration). With the new configuration, the beacon only includes between 12 and 17 predicted locations depending on the speed, which represents a maximum of 9 seconds in the future, considerably lower than the original version of the protocol.

Figure 5.13a compares the distance between the UAVs when they stop due to a collision risk in the original (MBCAP), and the enhanced version of the protocol (MBCAP-e), both flying at the same speed. In the previous version, the UAVs stop too soon, causing the low-priority UAV to wait for a long time period until the high-priority UAV overtakes it. Furthermore, the distance between them increases as the speed goes down. On the other hand, with MBCAP-e, the distance is significantly lower, and it increases with speed. Figures 5.13b and 5.14 show similar results when the UAVs travel at different speeds. As stated before, the original version of the protocol is prone to stop the UAVs too soon when their speeds are quite different. With MBCAP, when the UAVs meet face-to-face, this distance could be up to 380 m, and when one of them overtakes the other from behind, it could be up to 260 m, making the process unnecessary slow. With MBCAP-e, the distance becomes almost independent of speed differences, and it is significantly lower.

In order to enhance the performance of the protocol, we have introduced additional improvements regarding the information included in the beacon:

- **Prediction window**. As referred above, the number of future locations sent is reduced from 50 floats to a number of doubles ranging from 12 to 17. This improvement reduces the size of the beacon from 634 Bytes to 328-448 Bytes, despite the fact that we have added two new fields, and reduces the CPU overhead while checking if there is a collision risk, thereby improving the overall quality of the prediction.

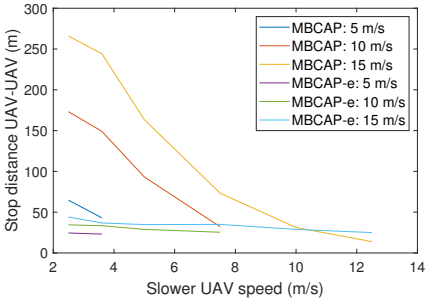
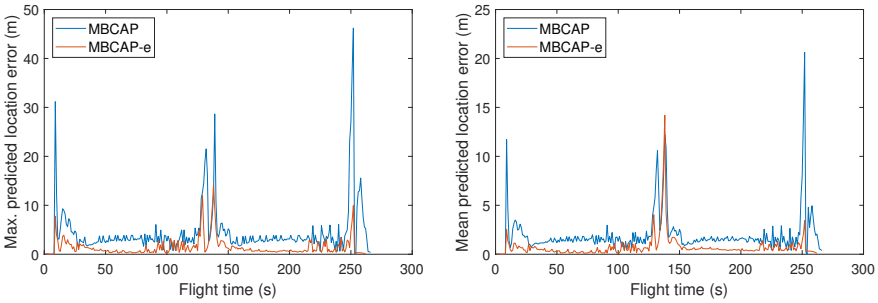


Figure 5.14: Distance between UAVs after stopping in a standard takeover, both flying at different speeds.



(a) Maximum predicted location error. (b) Average predicted location error.

Figure 5.15: MBCAP vs. MBCAP-e: Predicted location error vs. flight time.

- **Beacon renewal**. If the protocol state of the UAV changes, we immediately update the predicted locations in the beacon.
- **Location accuracy**. The predicted locations were originally sent as UTM coordinates in float numbers, and now, in MBCAP-e, they are sent as double numbers instead, which increases the precision when detecting possible collision risks.
- **Braking awareness**. Now, if the multicopter is braking ($a < -0.6$ m/s^2), only the current location is sent.

All of these changes have improved the overall quality of the prediction. To check the accuracy improvement of the mechanism used to predict future locations, we repeated the experiments detailed in section 5.1.4. Figure 5.15a compares the maximum predicted location error for the original protocol and MBCAP-e, and Figure 5.15b shows the mean error. Again, when the UAV accelerates at the

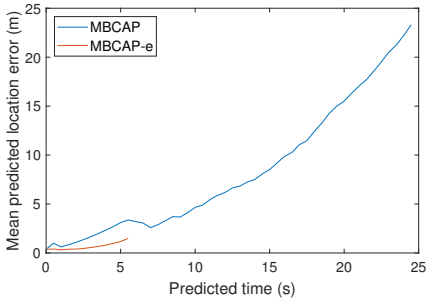


Figure 5.16: MBCAP vs. MBCAP-e: Average predicted location error for each of the predicted positions on the beacon.

beginning of the experiment, while it takes the curve (second 140), and when it brakes for landing, the prediction error is higher. The error when flying at constant speeds remains mostly uniform. In all cases, the error in MBCAP-e is significantly lower than in the original version of the protocol, with a uniform maximum error of 1 meter when the UAV traverses each segment of the mission at a constant speed. Figure 5.16 shows the average error for each of the predicted positions in the beacon along the whole experiment. We can see that we now send much fewer locations than before, and that the prediction quality increases, showing in general less error.

Additional enhancements have been included regarding the collision detection calculation. These changes, detailed below, improve the success rate at avoiding collisions, and reduce the time overhead introduced by the protocol (see section 5.3.2):

- **Risk detection during landing disabled**. We have disabled the protocol when one of the UAVs involved is landing because some landing procedures cannot be stopped, making it impossible to apply the protocol. This update required modifying transitions *f*, *i*, *j*, and *k* of the finite state machine (see Figure 5.17) to avoid detecting a possible collision risk when the other multicopter is in fact landing. Furthermore, we added the field *land* to the beacon (see Figure 5.11) to allow any UAV to analyze the flying state of other UAVs.

- **Risk detection over time**. To detect if there is a collision risk we check if the predicted locations match in space and time. Now we check if they match on time only if the speed of both UAVs is greater than 1 m/s, and both beacons send more locations in addition to the current location of the UAV. This is a more conservative approach, as we assume that a stopped, or almost stopped UAV, will maintain its location over time, and the approaching UAV

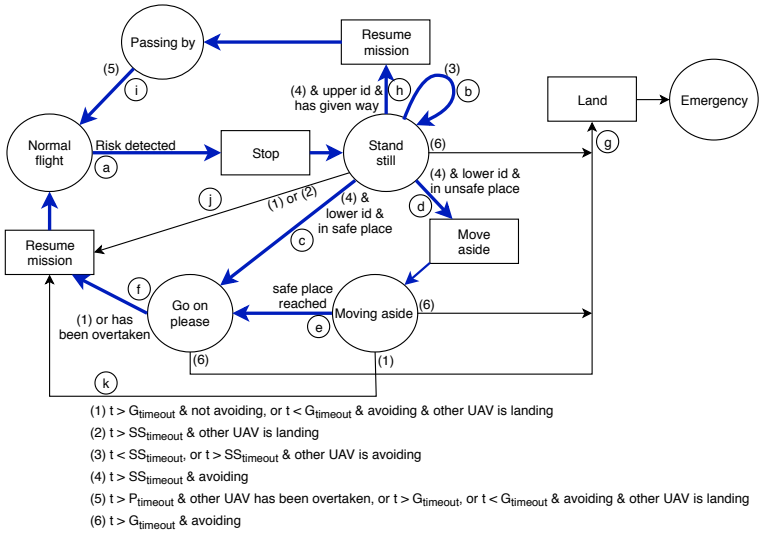


Figure 5.17: MBCAP-e finite state machine.

will detect a risk when any of the future locations that it reports in the beacon matches in space with the location of the UAV that is standing still.

- **Short-term position status**. If the other UAV is in the *Go on please* or the *Stand still* state, we only check if there is a collision risk with the current location of the UAV, ignoring the remaining locations included in the beacon. The first case includes the location where the UAV has detected the collision risk, and the second case includes a list of waypoints. None of these locations represent a position where the UAV will be on the short term, and so checking the existence of a possible collision risk at these locations would be inappropriate.

- **Fewer waypoints per beacon in the *Stand still* state**. In order to improve performance, we now include only the waypoints needed to report the path the UAV will follow for the next 400 meters, a distance that is greater than the maximum distance between UAVs when they stop to avoid the collision.

- **Risk check timeout**. This new timeout works as follows: Once a collision risk has been avoided, we wait four seconds before attending any other UAV informing that there is a collision risk with the current UAV (field *idAv* of the beacon). This approach solves a race condition in the protocol, due to its distributed nature, where the current UAV again detects a collision risk with

the other UAV because the latter keeps signaling that the previous collision situation is still in the process of being avoided.

- **Global timeout set to 120 seconds**. If a UAV is in an state different from *Normal flight* for more than 120 seconds, it must resume the mission if possible (see Figure 5.17), or land due to a deadlock condition associated to a protocol failure. This threshold is wide enough to consider the worst case, where the high-priority UAV (planned speed 1 m/s) could need 92 seconds to overtake the other UAV (planned speed 15 m/s). This situation could happen when the UAVs meet face-to-face.

5.3.1.3 Collision avoidance strategy

In order to improve the success ratio at avoiding collisions, and the overall performance of the protocol, several changes have been introduced in the collision avoidance strategy:

- **Reduced safety distance**. In the original version of MBCAP, when the low-priority UAV needs to move aside, it moves until the distance to the mission segment is of 20 meters. We have rationalized this distance as the sum of the probable GPS error of both UAVs, an error margin due to detected errors of the flight controller while trying to take a curve at a high speed, and an additional error margin due to slight movements of the UAV while standing still:

$$d_s = 2 \times GPS_{error} + curve_{error} + position_{error} \tag{5.5}$$

  Where:

$$GPS_{error} = 2.5\ m,\ curve_{error} = 1.5\ m,\ position_{error} = 1\ m$$

- **Overtaking end behavior**. In MBCAP-e, the overtaking process does not finish until the high-priority UAV is at least 20 meters (collision risk threshold) beyond the other UAV when the distance is increasing. As the safety distance is now lower than 20 meters, without this requirement the UAV would immediately detect another collision risk with the low-priority UAV, and the protocol would be triggered again.

- **Waypoint behavior**. When the high-priority UAV is close to a waypoint (see the UAV with an arrow indicating its direction in Figure 5.18), it performs a curve to approach that waypoint, but without actually reaching it, and without stopping at all. If a collision risk is detected, both UAVs stop, but the previous approach to detect if the low-priority UAV is far enough from the path the other UAV is about to follow is not good enough, as there is an offset (dotted line) between the real path and the theoretical mission

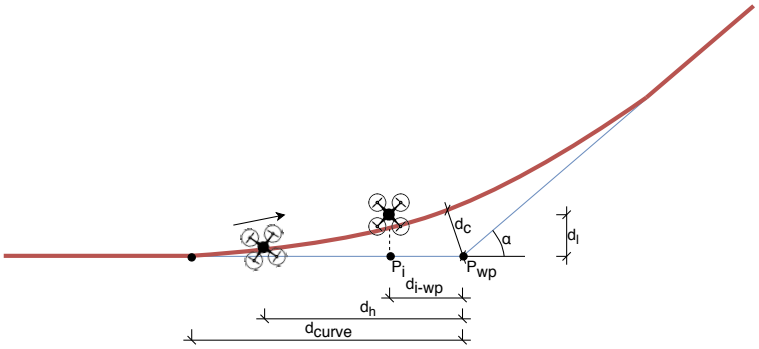


Figure 5.18: Safe location on curve analysis.

of the UAV. To consider this special case, we have calculated the function that represents the maximum distance d_c between the curve and the theoretical mission for different values of the planned speed and angle α between consecutive segments of the mission ($d_c = f(s, \alpha)$). If the high-priority UAV is close to a waypoint ($d_h < d_{curve}$), and the angle between segments of the mission is reduced so as to trace a curve ($\alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}]$); then, if the low-priority UAV is in the inner side of the corner defined by the mission segments of the other UAV, and its distance d_l to the mission segment is not safe ($d_l < d_c + d_s$), we assume that there is a collision risk, and the UAV must move to the other side of the mission segment to guarantee safety.

5.3.2 Protocol validation

Having introduced the improved version of the MBCAP protocol, in this section we validate its correctness. To this end, we run ArduSim with different roles (see chapter 4.1) to perform three different sets of experiments to analyze the behavior of the protocol under several circumstances: (i) UAVs approaching following straight trajectories under the most common scenarios, and with the presence of wind (ArduSim running as simulator), (ii) comparison of the results gathered in simulation with experiments using real multicopters (ArduSim running as simulator and also deployed on real UAVs), and (iii) analysis of the scalability and behavior of MBCAP when collision risks happen in scenarios with a large number of UAVs (ArduSim running as simulator).

Like in the previous chapter, for our tests we will refer to the low-priority UAV with the number 1, and to the high-priority UAV with number 2, meaning that the UAV 1 is the one moving aside whenever necessary.

Table 5.3: MBCAP flight time overhead (min:sec).

Scenario	UAV	MBCAP			MBCAP-e		
		on	off	Δt	on	off	Δt
1	1	3:41	3:03	0:38	3:29	2:59	0:30
	2	3:21	3:03	0:18	3:17	3:00	0:17
2	1	4:28	3:35	0:53	4:08	3:33	0:35
	2	3:32	3:03	0:29	3:23	2:59	0:24
3	1	3:59	3:03	0:56	3:41	2:59	0:42
	2	3:32	3:03	0:29	3:21	3:00	0:21
4	1	3:40	3:03	0:37	3:27	2:59	0:28
	2	3:21	3:03	0:18	3:16	3:01	0:15
5	1	3:51	3:03	0:48	3:34	2:59	0:35
	2	3:26	3:03	0:23	3:16	3:01	0:15
6	1	10:46	10:11	0:35	10:31	10:04	0:27
	2	30:47	30:27	0:20	30:33	30:17	0:16

The main metric used to validate MBCAP-e is the success ratio at avoiding collisions between UAVs, and the second metric is the flight time overhead introduced by this protocol.

5.3.2.1 Common scenarios and impact of the wind

The first set of experiments analyzes the most common scenarios where two UAVs approach each other following straight intersecting trajectories, and considering different angles, with the same setup detailed in section 5.2.

The collision was avoided in all cases, and the time overhead (Δt) introduced by the protocol is shown in Table 5.3, comparing the original version of the protocol with MBCAP-e. In general, we find that the flight time overhead introduced by MBCAP for UAV 1 is in the range between 35 and 56 seconds, and for UAV 2 it ranges from 18 to 29 seconds. On the other hand, MBCAP-e introduces an overhead of 27 to 42 seconds, and 15 to 24 seconds, respectively. The results show an overall gain for MBCAP-e in terms of flight time overhead, especially for UAV 1 (lower priority); such improvement is manly associated to having UAVs stop at a shorter distance between them.

We also tested if the presence of uniform wind affects the correctness and performance of the protocol (see Table 5.4). We observe that the flight time increases with headwind, as the UAV is no longer able to keep a ground speed of 10 m/s, and it has no significant variation when the UAV flies at 5 m/s, or if sidewind or tailwind are present. Moreover, the flight time overhead introduced by MBCAP and MBCAP-e is not significantly affected by the wind speed.

Table 5.4: MBCAP flight time overhead (min:sec) vs. wind.

Scenario	Wind	UAV	MBCAP	MBCAP-e		
			Δt	on	off	Δt
1	Crosswind	1	0:38	3:29	3:00	0:29
	Headwind	2	0:18	3:26	3:11	0:15
1	Crosswind	1	0:38	3:30	3:01	0:29
	Tailwind	2	0:18	3:15	2:58	0:17
2	Headwind	1	0:53	4:09	3:34	0:35
	Headwind	2	0:29	3:32	3:08	0:24
2	Tailwind	1	0:53	4:07	3:32	0:35
	Tailwind	2	0:29	3:22	2:58	0:24

(a) Quadcopter (b) Hexacopter

Figure 5.19: Multicopters used in real testbed.

Overall, we find that both versions of the protocol avoid the collision in all cases, being the flight time overhead introduced by MBCAP-e significantly lower than for the previous version.

5.3.2.2 Simulation vs. real testbed for common scenarios

The second set of experiments analyzes the effectiveness and performance of MBCAP-e when running on real multicopters. To this end, we deployed ArduSim with MBCAP-e enabled in a GRCQuad quadcopter from Quaternium [57] (see Figure 5.19a), and in a customized hexacopter (see Figure 5.19b), both capable of running ArduSim with the *multicopter* role in a Raspberry Pi 3B+ attached to them, and connected to the telemetry port of the flight controller through a serial port link (detailed instructions are available in the ArduSim repository). Experiments were performed for scenarios 1 to 5 from the previous section with similar missions, and then they were repeated without using MBCAP-e in order to measure the flight time overhead. In the experiments, the hexacopter had higher priority than

the quadcopter. Finally, the experiments with and without MBCAP-e were repeated in simulation to compare both results. As an example, Figure 5.20 depicts a Google Earth 3D view that shows the path followed by the real multicopters with a red line, the path of the virtual high-priority UAV with a blue line, and the route of the virtual low-priority UAV with a black line. The green arrows indicate the direction the UAVs are moving towards before detecting the collision risk, also marked with a green circle. We can observe that the paths followed in simulation and in real experiments are quite similar, indirectly validating the accuracy of ArduSim at simulating the physical properties of an actual UAV.

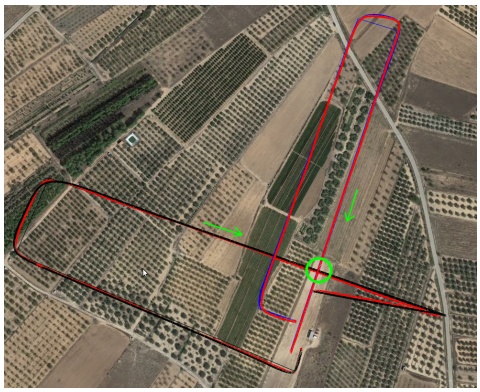


Figure 5.20: Simulation vs. reality in a perpendicular crossing (scenario 1).

The collisions were avoided in all cases, both in simulation and in real experiments. Table 5.5 shows that, in general, the flight time overhead remains similar in both environments, with the exception of scenarios 1 and 5, where the presence of gusty wind slightly increased the time necessary to complete the process. A video showing these experiments is also available online[1].

5.3.2.3 Scalability analysis

In the previous sections we have confirmed that MBCAP-e behaves better than its previous version. The protocol always avoids collisions whenever two UAVs meet in the air following a straight line from different directions, and the flight time overhead is bounded and low enough considering the battery capacity of current multicopters. Moreover, we have shown that MBCAP can easily be deployed on real multicopters thanks to ArduSim's capabilities. In this section, we analyze how the protocol behaves when the risk of collision increases, and the UAVs trace

[1] https://youtu.be/xHnMuMOd9C0

Table 5.5: MBCAP-e flight time overhead (min:sec). Simulation vs. real testbed.

Scenario	UAV	Simulation			Real testbed		
		on	off	Δt	on	off	Δt
1	1: Quadcopter	3:53	3:25	0:28	3:56	3:21	0:35
	2: Hexacopter	3:10	2:53	0:17	3:15	2:59	0:16
2	1: Quadcopter	4:10	3:38	0:32	4:12	3:36	0:36
	2: Hexacopter	3:57	3:33	0:24	4:08	3:43	0:25
3	1: Quadcopter	3:39	2:56	0:43	3:39	2:56	0:43
	2: Hexacopter	3:20	2:56	0:24	3:18	2:59	0:19
4	1: Quadcopter	3:30	3:04	0:26	3:29	3:03	0:26
	2: Hexacopter	3:20	3:02	0:18	3:29	3:03	0:26
5	1: Quadcopter	3:31	2:54	0:37	3:49	2:55	0:54
	2: Hexacopter	3:14	2:58	0:16	3:23	2:59	0:24

curves along their path, by simulating a large number of UAVs in a bounded area. A video that summarizes some of these experiments is also available online[2].

Experimental setup

MBCAP was tested on a squared area of 5×5 km, deploying 25, 50, 75, and 100 UAVs on four different scenarios. Each scenario consists on a new random deployment location for each UAV, and each UAV is assigned a new random mission based on the Gauss-Markov Mobility pattern [23] included in OMNeT++ [53]. Each experiment was repeated three times, taking the mean value. Moreover, the flight time and the traveled distance were measured with MBCAP, MBCAP-e, and without applying the protocol at all, in order to determine its overhead and performance. When the protocol is not used, the mean flight time was of 1 hour and 4 minutes, the mean traveled distance per UAV was of 36.9 kilometers, and the mean number of collisions was of 6.5, 16.5, 45.5, and 84.25 when deploying 25, 50, 75, and 100 UAVs, respectively. Also, we found that the selected scenario does not significantly affect the experimental results.

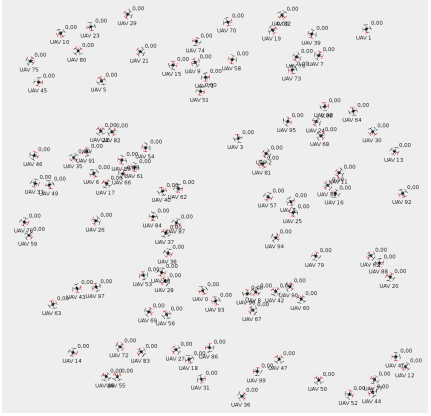
We use Algorithm 4 to get a random location and heading for each UAV included in our experiments. Initially, all the UAVs are randomly located inside the target area; then, if needed, their initial location is adjusted so that the distance between them is greater than or equal to the minimum distance specified for the experiment (100 meters). The minimum distance between UAVs has an upper limit to assure that they can fit inside the area ($d_{min} < \Delta x/\sqrt{n}$, and $d_{min} < \Delta y/\sqrt{n}$). Figure 5.21a shows the results gathered for a scenario with 100 UAVs randomly deployed.

[2]https://youtu.be/bEdcsPX1hXY

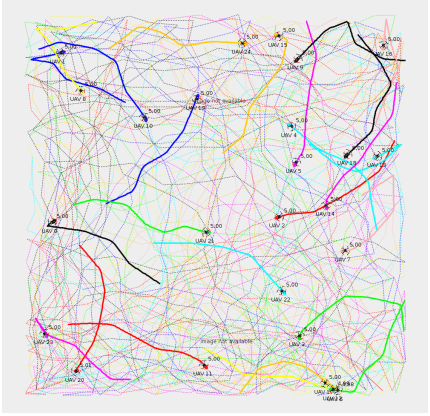
Algorithm 4 RanLoc returns a random initial location and a random heading for n UAVs.

Require: $area = \Delta x \times \Delta y = 5 \times 5$ km, n UAVs (25, 50, 75, or 100), area center location, $d_{min} = 100$ minimum distance between UAVs in meters.
Ensure: $A = \{(P_1, \beta_1), (P_2, \beta_2), ...(P_n, \beta_n)\}$, where P_i are locations, and β_i are headings.

$A = \emptyset$
for $i \leftarrow 1$ **to** n **do**
    $P_i = random\ location$
    $\beta_i = random\ heading$
    $A = A \cup (P_i, \beta_i)$
end for
$success = false$
while $\neg success$ **do**
    $success = true$
    for $i \leftarrow 1 \wedge success$ **to** n **do**
        for $j \leftarrow i + 1 \wedge success$ **to** n **do**
            if $distance\ P_i\ to\ P_j < d_{min}$ **then**
                $P_i = random\ location$
                $success = false$
            end if
        end for
    end for
end while
return A



(a) 100 UAVs randomly deployed.



(b) 25 random Gauss-Markov missions.

Figure 5.21: Experiment setup in an area of 5×5 km.

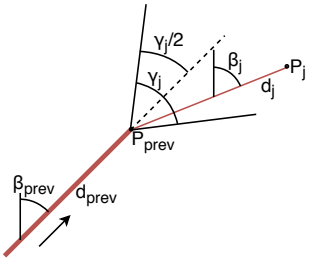


Figure 5.22: Gauss-Markov mobility model calculations.

The main objective of this set of experiments is to force the UAVs to meet several times in the air, not only when they are flying following a straight line, but also when they are performing a curve trajectory close to a waypoint. To this end, we designed long experiments where the UAVs are changing their direction along almost one fourth of the mission length so as to create a highly unfavourable scenario. We used Algorithm 5 (see also Figure 5.22) to get all the waypoints of the mission, starting from the initial location previously calculated. The length of each segment of the mission was randomly obtained in a range that varies from 250 to 500 meters. Moreover, the maximum length should be lower than half the side of the area were UAVs are deployed to guarantee that the algorithm can go on. Global parameter α represents the linearity of the path followed by the UAVs, varying from 0 (Brownian movement) to 1 (linear motion). We set α to 0.75, which makes the mission significantly linear (see Figure 5.21b), changing to a Brownian movement when the UAV is too close to the limits of the area, with the aim of allowing it to bounce inward. Finally, we set the number of waypoints of the mission to 100, which is equivalent to 99 segments having a mean length of 375 meters.

Global results

Table 5.6 compares the results gathered with MBCAP, and with MBCAP-e, when varying the number of UAVs in the area. We show the mean value for all the experiments, finding that MBCAP-e significantly outperforms MBCAP. We now proceed to analyze the performance metrics included in table 5.6.

- **Collisions expected**. Represents the mean number of collisions detected between UAVs in a single experiment, when the protocol is not in use. This value allows us to determine the success ratio of the protocol at detecting and avoiding collisions.
- **Risks detected**. Represents the mean number of times the collision-avoidance protocol is enforced throughout the experiment. Figure 5.23a shows

Algorithm 5 RanMission gets a random mission for n UAVs.

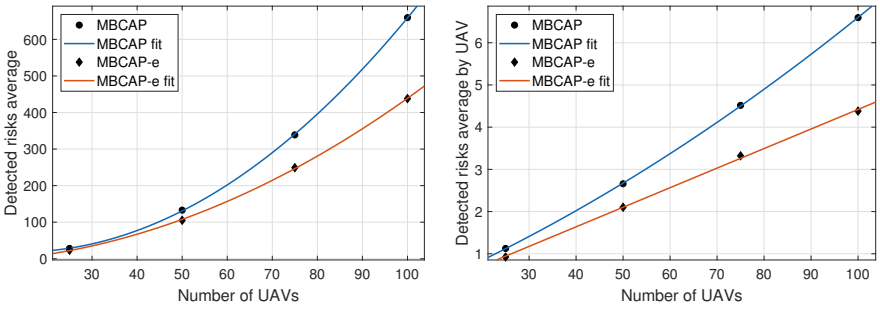
Require: $area = \Delta x \times \Delta y = 5 \times 5$ km, n UAVs (25, 50, 75, or 100), location of the area center, $[d_{min}, d_{max}] = [250, 500]$ mission segment length range in meters, mobility linearity of $\alpha = 0.75$, $numWPs = 100$ waypoints of the mission, $A = \{(P_1, \beta_1), (P_2, \beta_2), ...(P_n, \beta_n)\}$ starting location and heading of the n UAVs.

Ensure: $M = \{mission_1, mission_2, ...mission_n\}$, where $mission_i$ is the sequence of waypoints that comprises the mission of the UAV i.

$M = \emptyset$
for $i \leftarrow 1$ **to** n **do**
  $P_{prev} = P_i$
  $mission_{i_1} = P_{prev}$
  $\beta_{prev} = \beta_i$
  for $j \leftarrow 2$ **to** $numWPs$ **do**
    $\alpha' = \alpha$
    $\gamma_j = 2\pi\,(1 - \alpha')$
    $d_j = random\ length \in [d_{min}, d_{max}]$
    $\beta_j = random\ heading \in [\beta_{prev} - \gamma_j/2, \beta_{prev} + \gamma_j/2]$
    $P_j = P_{prev} + f(d_j, \beta_j)$
    if $P_j \notin area$ **then**
      do
        $\alpha' = 0$
        $\gamma_j = 2\pi\,(1 - \alpha')$
        $\beta_j = random\ heading \in [\beta_{prev} - \gamma_j/2, \beta_{prev} + \gamma_j/2]$
        $P_j = P_{prev} + f(d_j, \beta_j)$
      while $P_j \notin area$
    end if
    $mission_{i_j} = P_j$
    $P_{prev} = P_j$
    $\beta_{prev} = \beta_j$
  end for
  $M = M \cup mission_i$
end for
return M

Table 5.6: MBCAP vs. MBCAP-e: Collision avoidance performance (mean value by experiment).

	MBCAP				MBCAP-e			
Number of UAVs	25	50	75	100	25	50	75	100
Collisions expected	6.5	16.5	45.5	84.25	6.5	16.5	45.5	84.25
Risks detected	28.17	132.92	338.75	659.42	23.08	105.08	249.08	438
Soft collisions ($d < 5m$)	0.58	1.83	3.42	10.08	0.08	0.08	0.58	1.5
Hard collisions ($d < 4m$)	0.58	1.67	2.58	8.42	0.08	0.08	0.58	1.08
Deadlocks avoided	0.08	1.67	4.33	10.83	0	0.33	0.25	0.58
Deadlock failures	0.33	3.42	8.75	21.42	0	0	0	0



(a) Total risks vs. number of UAVs: $\mathcal{O}(n^2)$.

(b) Risks by UAV vs. number of UAVs: $\mathcal{O}(n)$.

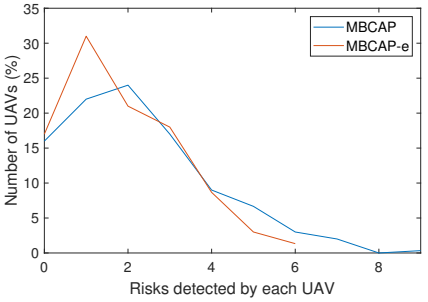
Figure 5.23: MBCAP vs. MBCAP-e: Average risks detected during an experiment.

that the mean number of risky situations detected along an experiment increases with the number of UAVs present in the area as $\mathcal{O}(n^2)$, and it also shows the precision increment in the collision detection strategy that is achieved with MBCAP-e, where the latter is able to prevent UAVs from stopping unnecessarily in many situations. On the other hand, Figure 5.23b shows that the mean number of dangerous situations detected by a UAV in a single experiment increases with the number of UAVs as $\mathcal{O}(n)$. Finally, table 5.6 and Figure 5.24 show that, in general, the UAVs detect less risky situations when adopting MBCAP-e, for any number of UAVs, e.g. 66.4% for 100 UAVs.

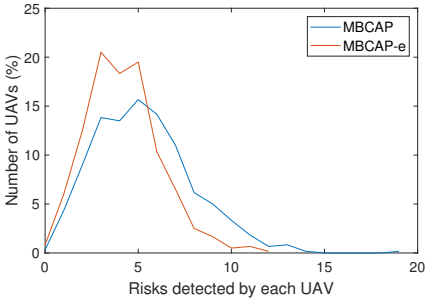
- **Soft collisions ($d < 5m$)**. Mean number of possible collisions taking place during an experiment. We consider that a simulated collision has happened when two UAVs are located at a distance lower than 5 meters. The typical GPS error on multicopters is 2.5 meters, and so we consider it a very unfavourable situation, occurring in those cases where the GPS error bias of both UAVs is exactly the opposite. MBCAP-e highly improves the collision avoidance ratio with respect to MBCAP, e.g., from 88.04% to 98.22%, for the worst-case experiment (100 UAVs).

- **Hard collisions ($d < 4m$)**. Represents the same metric, but with a more realistic threshold. In this case, we only consider that a collision has happened if the UAVs are closer than 4 meters. As expected, the success ratio is higher (98.92%), but this is only detected in experiments with 100 UAVs, as in other cases the number of collisions is too low to compare.

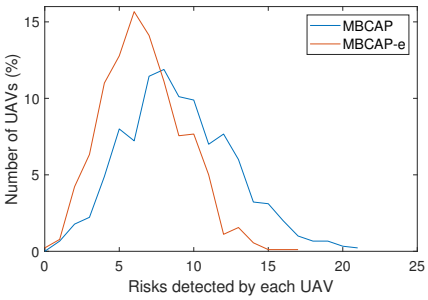
- **Deadlocks avoided**. Represents the number of situations where a UAV surpasses the global timeout when waiting for other UAVs to solve another



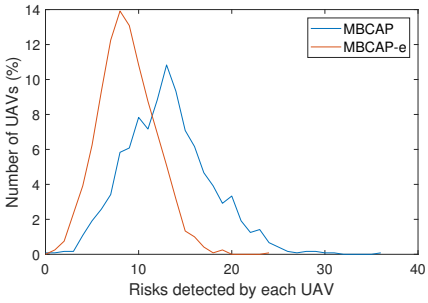
(a) Experiment with 25 UAVs.



(b) Experiment with 50 UAVs.



(c) Experiment with 75 UAVs.



(d) Experiment with 100 UAVs.

Figure 5.24: MBCAP vs. MBCAP-e: Distribution of UAVs given the risks detected by each one.

collision risk situation, but it resumes its mission (transitions *f*, *i*, *j*, or *k* in Figure 5.17). Somehow the protocol has failed, and the UAV has been waiting for an excessive time because it is trying to solve a collision with a UAV that has already moved out of the contending area; in this situation, the protocol is able to detect that the UAV is no longer present and that the risk has gone, allowing the waiting UAV to go on with its mission. With MBCAP-e, the UAVs are in this situation only in extremely rare cases, i.e. 0.07% for 100 UAVs in these specially unfavourable scenarios.

- **Deadlock failures**. Represents the number of situations where a UAV surpasses the global timeout while waiting for other UAVs to solve another collision risk situation, and it is not safe for it to go on with the mission (transition *g* in Figure 5.17). If a low-priority UAV is blocked in a state for too long, and the high-priority UAV is already present in the conflict area, it should not continue with the mission, because resuming it could cause a collision. MBCAP showed an undesirable behavior, failing in many

Table 5.7: MBCAP vs. MBCAP-e: Performance comparison (mean value by experiment).

		Reference	MBCAP	MBCAP-e
Flight time (s)	Min.	3618	3691	3688
	Mean	3848	4154	4006
	Max.	4111	5511	4457
	Max. overhead	-	1120	553
Flight length (m)	Min.	34893	34916	34918
	Mean	36898	36949	36933
	Max.	39194	39258	39253
	Max. overhead	-	229	127
Mean speed (m/s)		9.59	8.9	9.22

encounters, while with MBCAP-e no UAV needed to land.

To gain further insight on the protocol performance, we analyzed in detail the few collisions detected, finding that the collision risks between two UAVs were always avoided, meaning that collisions always happened when three or more UAVs met in the same area, and at the same time; in particular, problems only occur when a third UAV stops in the path that the high-priority UAV is following while overtaking the low-priority UAV. We can consider this case a possible but improbable situation.

Regarding the behavior of the low-priority UAV, when it stops it can stand still while being overtaken, or it could move aside to allow the other UAV to go on. With MBCAP it needed to move aside in 22.8% of the cases, while with MBCAP-e this occurred for 28.3% of the cases. This increment is due to the *Waypoint behavior* improvement included in the collision avoidance strategy (see section 5.3.1.3 and Figure 5.18), as it forces the low-priority UAV to move further away from the path the high-priority UAV has to follow.

Up to this point, we have compared the success ratio of MBCAP and MBCAP-e. Now we analyze the flight time overhead of both versions of the protocol. Table 5.7 shows the mean flight time and path length for a UAV using MBCAP, MBCAP-e, and without using the protocol. We can observe that, with MBCAP-e, a UAV needs to travel for additional 35 meters on average, and consumes a mean extra time of 158 seconds at avoiding collisions, while with MBCAP it needs 51 extra meters and 306 extra seconds, respectively. The mean speed during flight is also higher with MBCAP-e, showing that MBCAP-e is significantly more efficient at avoiding collisions than the previous version. Figure 5.25 shows the mean time overhead for a UAV during the whole experiment, given the number of times that UAV needed to stop to avoid a collision. We can observe that the slope of the line is nearly constant, independently of the number of UAVs included in the

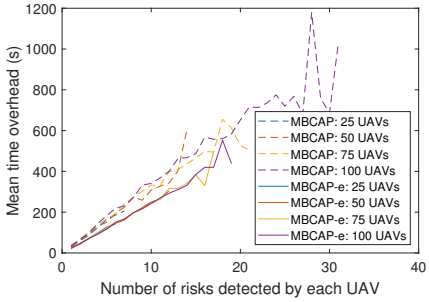


Figure 5.25: MBCAP vs. MBCAP-e: Global time overhead given the risks detected by each UAV.

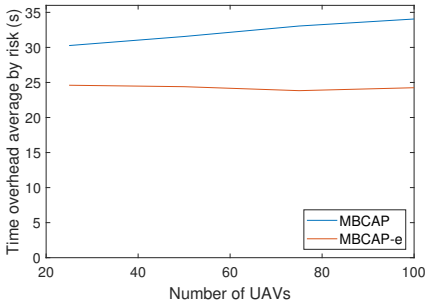


Figure 5.26: MBCAP vs. MBCAP-e: Time overhead by risk detected vs. number of UAVs.

experiment. Also, we find that its slope is lower with MBCAP-e, as the UAV requires less time to avoid each collision. We also find that, the more UAVs are present in a same experiment, the more collision risks per UAV are detected, and that with MBCAP a UAV detects more collision risks than with MBCAP-e. Finally, Figure 5.26 shows the mean time needed for a UAV to avoid a single collision depending on the number of UAVs flying around in the same experiment. It is clear that, with MBCAP-e, UAVs require less time to avoid a collision, with a mean value lower than 25 seconds, while with the previous version it needed more than 30 seconds.

Overall, experiments have shown that MBCAP-e adds a significant improvement over the earlier version of this protocol when performing large scale simulations in a congested airspace. Also, we have found that the time overhead introduced by the protocol remains quite low (mean overhead of 25 seconds per risky situation solved).

5.4 Summary

As new mission-based applications for multicopters emerge, the number of UAVs flying simultaneously also increases, and the risk of collision between them becomes higher. In addition, there are currently no collision avoidance protocols developed for UAVs from different owners when performing planned missions.

In this chapter we propose MBCAP, a collision avoidance protocol for multirotor UAVs performing planned missions by relying on a cooperative sense & avoid approach, and MBCAP-e, an enhanced version of MBCAP. Experimental results showed that MBCAP-e is able to avoid collisions between two UAVs in all cases, and with a success ratio of 98.22% in highly crowded environments (100 UAVs scenario). Experiments using real UAVs evidenced the resemblance between the simulated and the real-life performance of MBCAP-e. In addition, we found the flight time overhead introduced by the protocol to be quite low and well bounded, considering the current lifespan of multicopter batteries. Overall, the effectiveness, reliability, and efficiency of MBCAP-e proved to be considerably higher when compared to its previous version (MBCAP), making it suitable for deployment in real multicopters.

Chapter 6

Mission-based UAV Swarm Coordination Protocol (MUSCOP)

Although there are already some solutions for the automation of UAV swarm flights, in certain situations automatic guidance can be required. Examples of such situations may include applications for large-scale agriculture in search of pests or weeds [19, 2], wild life recordings [5], or border surveillance [13], among others. In these specific cases, the different UAVs that make up the swarm must allow the coordination of multiple UAVs when carrying out the mission. Such mission must be planned beforehand. Then, the communications between UAVs should enable near-real-time responsiveness to maintain the consistency of the swarm.

In this chapter, we propose the MUSCOP protocol which provides UAV coordination to maintain the desired flight formation when carrying out planned missions. MUSCOP uses a centralized approach where the master UAV synchronizes all slave UAVs each time they reach an intermediate point in the mission.

6.1 Protocol overview

6.1.1 Introduction

MUSCOP aims at keeping a stable flight formation while performing a planned mission. To design the protocol, we rely on a master-slave model where the master UAV synchronizes all the slave multicopters every time they reach a waypoint in a mission. Furthermore, before beginning the mission, all the slaves receive

a mission brief having waypoint coordinates modified so as to agree with their position offset with regard to the swarm leader. This way, each UAV can move from one waypoint to the following one according to its own mission when the master UAV sends the corresponding command. Our solution keeps the flight formation consistent throughout the entire flight mission.

The messages used to synchronize UAVs in the swarm are transmitted from the master to the slaves, and from the slaves to the master, meaning that only two threads per UAV (*Talker Thread* and *Listener Thread*) are required for message handling. The *Listener Thread* thread implements the protocol's finite state machine, and reacts as soon as the required messages are received from other UAVs, sending commands to the flight controller to dynamically control the behavior of the multicopter.

6.1.2 Finite state machine

Figure 6.1 shows the finite state machine that rules the behavior of the master and slave UAVs. The circles represent the states, the curved arrows represent the messages sent and received, and the straight lines represent the transitions between states. The curved arrows above the states are the messages sent by the UAV *(Talker Thread)*, while the arrows below the states represent the messages received from other UAVs *(Listener Thread)*.

Letters "M" and "S" refer to the master and the slave UAVs. Before the UAVs take off, the UAV located in the centre of the flight formation ("C") becomes the master UAV, and the remaining UAVs take the role of slaves, being identified as "NC". Notice that, by setting the UAV in the middle of the formation as the master, we are able to optimize communications, as the messages are sent from master to slaves and vice-versa, and we need to minimize the distance between sender and receiver to mitigate the impact of channel losses.

All the UAVs begin in the *Start* state. The slaves send to the master a *hello* message to inform it about their presence, so it can know the number of UAVs that will integrate the swarm flight formation. When the user is informed that all the UAVs have successfully connected to the master, he presses a button that allows switching to the *Setup* state, thereby starting the initial configuration procedure. First, the master UAV decides the location of each UAV in the flight formation. Then, it computes a modified mission brief for each specific UAV, according to their position in the swarm, and sends the message *data* to the different slaves, which includes the personalized mission each of them will have to follow. Afterward, the slaves inform the master *(dataAck)* that they have received the message. Only when all the slaves have the required data does the master UAV switch to the *Ready to fly* state, and broadcasts a *readyToFly* message to force the remaining UAVs to switch to that same state, and send back the corresponding acknowledgement *(readyToFlyAck)*. When all the UAVs are ready to fly, the master UAV starts the take off (*Taking off* state), and stops sending the *readyToFly* message, forcing the

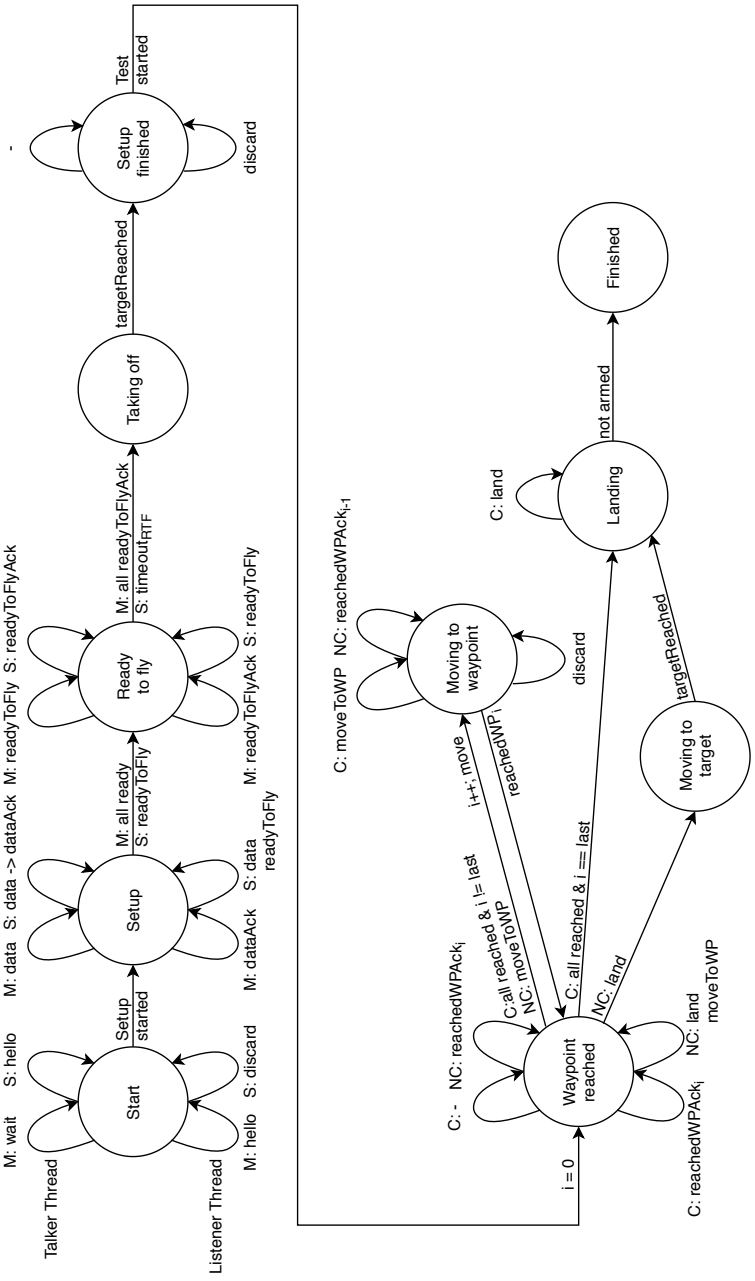


Figure 6.1: MUSCOP protocol finite state machine.

slaves to also take off. The setup process finishes when all the UAVs reach their respective location in the flight formation, switching to the *Setup finished* state.

The flight coordination protocol begins when the user presses a button to start the flight. The initial UAV location during flight is considered as the first waypoint of the mission, so all the UAVs begin in the *Waypoint reached* state. When the master UAV receives the *reachedWPAck* message from all the slaves, it starts to move to the next waypoint (*Moving to waypoint* state), and forces the slaves to also move to the next waypoint through message *moveToWP*. All the UAVs remain in the *Moving to waypoint* state until they reach the next waypoint. During that process, the master UAV is continuously sending a command to move to the next waypoint, while the slave UAVs return an acknowledgement of having reached the previous waypoint earlier on. This redundant behavior increases the reliability of the protocol, as the messages sent among the UAVs could be lost due to the distance or the presence of noise in the communications channel. In addition, we find that the proposed protocol is characterized by a very low occupancy of the wireless medium, as these messages are really short (6 and 14 bytes, respectively). When all the UAVs reach the last waypoint of the mission, the master UAV lands in its current location, and broadcasts the *land* message, including its current location, to force slaves to start the landing procedure. Before landing, the slaves move closer to the master. This way they maintain the same formation used during the flight while reducing the distance between them (i.e. 5 meters) to make sure the swarm landing area remains small. Otherwise, they would land on a much wider area, possibly on unexpected places, making it difficult to recover these UAVs afterward.

6.1.3 Message format

Figure 6.2 shows the format of the messages transmitted from the master to the slave UAVs, and vice-versa. In this section we detail their content and purpose. All the messages start with the *type* field, which identifies the type of message, and they are sent periodically (period of 200 ms) taking into account that they can be lost due to an unreliable communications channel. Furthermore, the communications rely on UDP broadcasting to be reached by all the UAVs within range while guaranteeing the lowest possible network overhead.

Message *hello* (1) is sent from the slaves to the master when they are turned on. This allows the master to detect their presence and their intention to take part in the swarm. The *id* represents a unique identifier for the sender UAV, and it is included in all the messages transmitted towards a specific UAV. On a real UAV, it is based on the MAC address of the communications adapter, as it is also unique, while in simulation it is a unique number provided by ArduSim. The current location is also included to allow the master UAV to organize the mission that must be followed by each UAV.

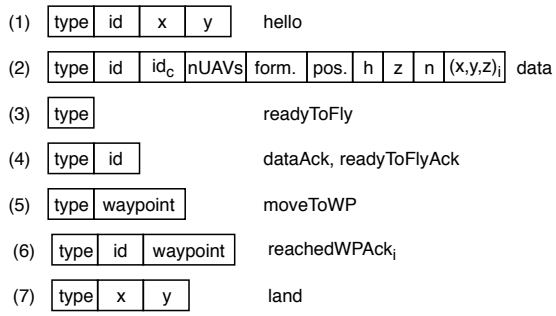


Figure 6.2: MUSCOP protocol message types.

During the setup phase, the master UAV defines the relative location of each UAV in the flight formation, and then it calculates the mission they will follow. The UAV in the center of the formation will follow the original planned mission, while the remaining UAVs will follow a modified version of that mission that includes a constant offset so as to maintain a same relative position in the swarm formation throughout the flight. During the flight, the UAV located at the center will be the master, a strategical choice which allows us to optimize master-slave communications. The *data* (2) message includes the following fields:

- *id*. Identifier of the target UAV.
- id_c. Identifier of the UAV that will be in the center of the flight formation. When id_c matches *id*, the target multicopter will become the master UAV during the flight, sending coordination messages to the remaining UAVs. Otherwise, the multicopter will become a slave, and will accept commands from the UAV with this identifier.
- *nUAVs*. Number of UAVs that will take part on the flight formation. This is a parameter required by the master UAV to know when all the UAVs have reached a specific waypoint.
- *form*. Type of formation, selected between the options provided by ArduSim: linear, matrix, and circular, among others.
- *pos*. Position of the target UAV in the flight formation. Beside the previous two fields, this one allows each UAV to know its relative location in the flight formation.
- *h*. Heading of the UAV swarm, fixed during the whole flight. Before landing, the multicopters surrounding the center UAV approach it, conforming a more compact version of the flight formation in order to land in a reduced area,

making it easier to collect the multicopters afterward. This field is needed for each UAV to calculate its location in the landing formation.

- *z*. Altitude over the ground for the take-off step.
- *n*. Number of waypoints of the mission included in the message.
- $(x,y,z)_i$. Coordinates for all the waypoints included in the message. The maximum number of waypoints that can fit in the message is 58, considering the maximum payload of a UDP datagram over standard Ethernet.

The master UAV sends the *readyToFly* message (3) when all the slaves have received the *data* message, and makes them ready to take-off.

Messages *dataAck* and *readyToFlyAck* (4) are used by slaves to inform the master UAV that the corresponding message has already been received. The former points out that the UAV has received the mission to be followed, and the later that it is ready to take-off. As the master needs to know when all of them have received these messages, they must include the identifier of the sender UAV.

Message *moveToWP* (5) allows the master UAV to synchronize the formation. When it notices that all the slaves have arrived to the current waypoint, it sends this command to make them all move towards the next waypoint. The slaves use the $reachedWPAck_i$ message (6) to inform the master UAV that they have reached the waypoint defined.

Finally, the *land* command (7) is sent by the master UAV, including its location, which allows slaves to determine where to land, adopting a more compact version of the flight formation.

6.2 Data sources and error assessment

Once the MUSCOP protocol was introduced, we now proceed to validate it. To this purpose, we will first provide details about the used data sources. Then, we will describe the procedure adopted to determine the errors associated with our tests.

6.2.1 Data source

The MUSCOP protocol needs a planned mission to be used as data source, so that the master UAV can guide the swarm along the defined route. In this sense, for our tests we plan different missions in such a way that the complexity of the missions is increasing. For this, we fixed the total distance for all missions to 1840 meters, defining a set of missions where the UAVs move towards the northeast direction with an increasing number of steps. In figure 6.3 there are a few mission examples where the same total distance to be traveled is maintained, but having

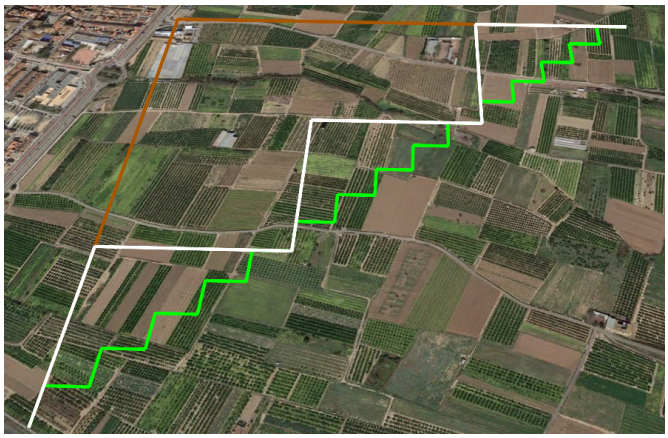


Figure 6.3: Samples of test missions with 2, 6, and 30 waypoints.

different mission complexity levels: 2 waypoints (brown), 6 waypoints (white), and 30 waypoints (green), respectively.

Once the missions were obtained and used as input, different simulations were performed. ArduSim generates a set of data for each UAV in the simulation. The information generated by ArduSim allows us to obtain UTM coordinates, speed, acceleration, height, heading, and time per UAV along the experiment. In our evaluation set, the location recorded at the beginning of the experiment is taken as the origin. Then, we use the time interval during which the flight data is available for all the UAVs in the simulation, and then proceed to perform the interpolation for the simulated data set at fixed time steps.

We now proceed to describe the methodology used to calculate the error associated to the different UAVs when flying as a swarm using MUSCOP. In general, we measure the time delay produced between the master UAV and the slave UAVs for the different formations tested: linear, matrix and circular. To achieve this, we first determined the general formation error.

Figure 6.4 shows the general error for an array formation. The point marked with "X" (blue color) represents the theoretical position on the formation calculated for each of the UAVs, and centered on the master UAV. The yellow line represents the calculated error corresponding to the position of the slave UAV (black color) and the theoretical one. The black arrows represent the direction of the movement for each of the UAVs in the swarm. It is noteworthy that all the UAVs move in the same direction along straight lines, and thus the offset for the entire journey will in general be parallel to the mission segments. Below we provide more details about the procedure adopted for the calculation of the position and time offsets for the UAVs in the swarm.

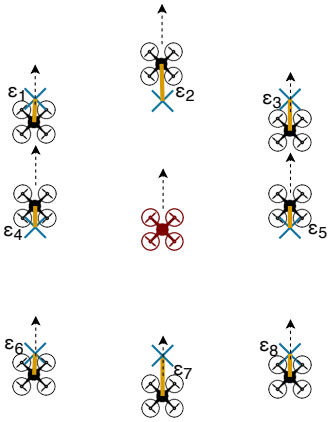


Figure 6.4: Distance offset error for the matrix formation.

6.2.2 Error analysis

The global swarm formation error (ε) refers to the theoretical location of all the slave UAVs, where the individual positions in the formation are defined based on the current location of the master UAV $\overrightarrow{P_{M_i}}$, and taking as reference a constant heading for the formation.

$$\overrightarrow{P_{M_i}} = (P_{x_i}, P_{y_i}) \tag{6.1}$$

In equation 6.1, i represents the current time step for a real position on x and y axes, respectively. $\overrightarrow{P_{k_i}}$ represents the theoretical position of the slave UAV k calculated for time instant i.

$$\overrightarrow{P_{k_i}} = \overrightarrow{P_{M_i}} + \overrightarrow{\Delta_k} \tag{6.2}$$

where:
$\overrightarrow{P_{k_i}} = (x_{k_i}, y_{k_i})$
$\overrightarrow{\Delta_k} = (\Delta x_k, \Delta y_k)$
$x_{k_i} = P_{x_i} + \Delta x_k$
$y_{k_i} = P_{y_i} + \Delta y_k$
$\Delta x_k = \mathit{offset}_{x_k} \cdot \cos(h) + \mathit{offset}_{y_k} \cdot \sin(h)$
$\Delta y_k = \mathit{offset}_{y_k} \cdot \cos(h) - \mathit{offset}_{x_k} \cdot \sin(h)$

As the heading h of the flight formation and the relative offset of a slave k with respect to the master UAV remain constant to achieve swarm cohesion, the theoretical position of the slave k is calculated adding a constant $\overrightarrow{\Delta_k}$ value to the current master location.

The actual position of a specific UAV is $\overrightarrow{P'_{k_i}} = (x'_{k_i}, y'_{k_i})$ on the period of time i. Notice that ${x'}_{k_i}$ is the real value obtained by the simulator in the x axis, and ${y'}_{k_i}$ is the real value obtained in the y axis, for each slave UAV at time instant i for the given interpolated data set.

The equation to calculate the error or distance offset of the slaves with regard to the master UAV is defined as:

$$\varepsilon_{k_i} = \sqrt{(x_{k_i} - {x'}_{k_i})^2 + (y_{k_i} - {x'}_{k_i})^2} \tag{6.3}$$

Finally, we calculate the time offset as $\varepsilon_{k_i}/v_{k_i}$, where v_{k_i} is the current speed of UAV k.

6.3 Protocol validation

Once the data source and the methodology used to calculate the errors have been defined, we proceeded to evaluate our proposed protocol. To achieve this, we conducted an extensive set of experiments using different numbers of UAVs, different distances between UAVs, and different formations. In this sense, we will divide our evaluation set in 3 parts: (i) impact of MUSCOP and mission complexity on flying time; (ii) impact of channel losses on swarm cohesion for the given formation; (iii) impact of using different inter-UAV distances in the swarm formation; and finally (iv) scalability analysis. Please notice that, in all the simulations carried out, we have only evaluated performance when the swarm formations are in the flight phase, discarding from our traces the take-off and landing phases. In an illustrative video[1] we run three experiments with different flight formations on the ArduSim simulator. Below we provide details about the results obtained.

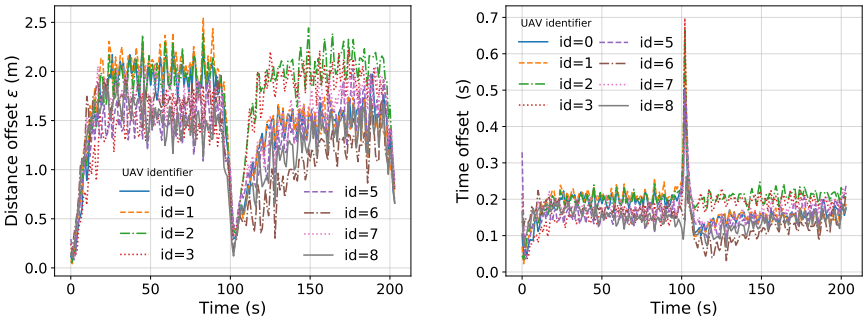
6.3.1 Impact of MUSCOP and mission complexity

In this section, we seek to evaluate the time and distance offsets produced by our protocol, checking the performance achieved in the presence of an ideal lossless channel. For this purpose we use the default configuration of the ArduSim simulator, keeping the speed constant at 10 *m/s*, using the linear formation for 9 UAVs, and varying the complexity of the mission. The total distance travelled by each UAV is 1840 meters.

6.3.1.1 Time series analysis

In this first evaluation, we analyze the behavior of a single simulation using nine UAVs and a linear formation. The target flight speed is set to 10 m/s, and we

[1] `https://youtu.be/VLMsbL5B6tA`



(a) Formation distance offset. (ε). (b) Formation time offset.

Figure 6.5: Evaluation using the linear formation with 9 UAVs (ideal channel).

adopt a distance between UAVs of 50 meters (our default value). The total number of waypoints in the mission is just 2.

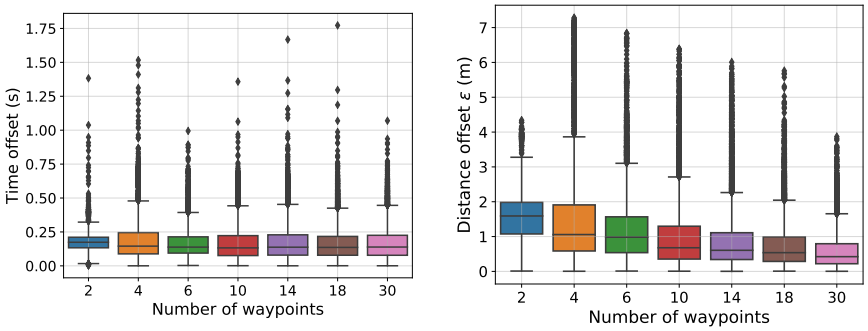
Figure 6.5 shows the formation error measured as position offset, and the corresponding delay offset as a function of time for a single experiment. At time 105 seconds it shows the offset between slaves and master UAV assessed when the flight coordination is taking place, when the UAV switches from the first to the second waypoint of the mission. Note that the distance offset error in the swarm formation remains low despite the time offset increases for that point. This is due to the deceleration of all UAVs, and the synchronization time overhead when arriving at this coordination point.

In general, Figure 6.5 (a) shows that the average error obtained for this type of formation is lower than 2 meters. Concerning the time offset, Figure 6.5 (b) shows that the average error is of 0.16s, which coarsely corresponds to the synchronization time requirements of our protocol (∼200 ms).

6.3.1.2 Impact of mission complexity

Once the behavior for a single simulation was analyzed, we proceeded to perform multiple simulations to evaluate the behavior when the complexity of the mission increases. The complexity of the mission studied was defined by varying the total number of waypoints while maintaining the overall mission length, as explained in section 6.2.1. In particular, we defined missions having 2, 4, 6, 10, 14, 18, and 30 waypoints, respectively. In total, five simulations were made for each mission type, and the speed and the separation between UAV was kept constant, adopting the values defined earlier.

Figure 6.6 shows the results obtained when we vary the complexity of the mission. In general, Figure 6.6a shows that the time offset maintains an approximate average value of 0.20s. Also, maximum atypical values are shown, which vary



(a) Time offset.

(b) Distance offset.

Figure 6.6: UAVs offset on a swarm of 9 UAVs (ideal channel).

Table 6.1: Overall simulation statistics (ideal channel).

Formation	Distance offset ε (m)			Time offset (s)		
	Mean	Max.	Std.	Mean	Max.	Std.
Linear	1.0865	7.2788	0.9077	0.1749	1.7724	0.1278
Matrix	1.0496	6.6886	0.8564	0.1720	1.3376	0.1209
Circular	1.1624	8.2147	0.9314	0.1854	1.6323	0.1266

according to the mission. Concerning the distance offset error, it becomes evident that, in general, it tends to decrease as the mission becomes more complex. Such behavior is shown in Figure 6.6b. To understand the reasons underlying this phenomenon, notice that distances between the coordination points decrease as the mission complexity increases, meaning there is less time for UAVs to accelerate and reach maximum speed values.

Table 6.1 shows the average, maximum, and standard deviation values for each flight formation evaluated. In particular, it shows that the matrix formation appears to achieve slightly better performance than the other formations, but the overall results are equivalent, as the distance between UAVs does not affect the communications link quality (ideal channel). In general, we conclude that, under an ideal channel, the number of waypoints and the complexity of the mission defined for the swarm is relevant for the distance offset error when the UAVs do not have enough time to reach maximum speed, but irrelevant otherwise. Table 6.1 also shows that the average time offset is close to the update interval adopted by our protocol regarding message broadcasting (messages are transmitted every 200 ms).

Table 6.2: Flight time overhead.

Number of Waypoints	Reference mission (s)	Linear formation (s)	Δt (s)
2	205.33	206.33	1.00
4	226.00	228.33	2.33
6	247.67	250.00	2.33
10	288.00	293.00	5.00
14	326.00	334.33	8.33
18	364.33	374.00	9.67
30	466.00	479.33	13.33

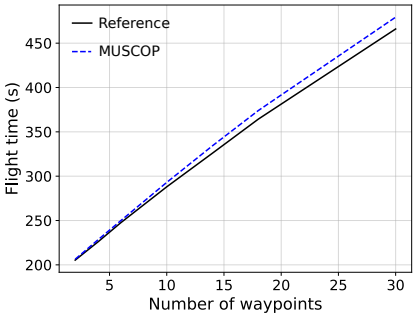


Figure 6.7: Flight time overhead using the linear formation with 9 UAVs (ideal channel).

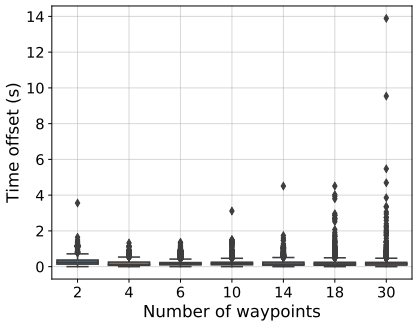
6.3.1.3 MUSCOP time overhead

In this section we analyze the time overhead introduced by our protocol. We measured the flight time when varying the number of waypoints (see Table 6.2), and we repeated the experiments five times, taking the mean value. Then, we compared the flight time with the value obtained when the UAV follows the same mission automatically *(Reference mission)*, obtaining the time overhead for the whole flight Δt. We found that the protocol only adds 0.55 seconds to the flight time for each waypoint crossed.

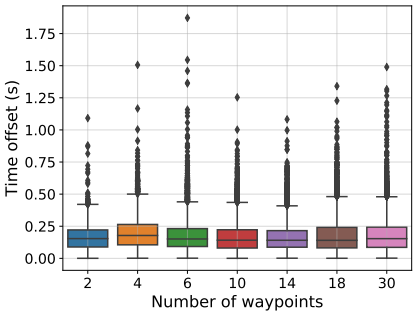
Figure 6.7 shows that the flight time increases linearly with the number of waypoints, and that the total time added to the flight remains very low.

6.3.2 Impact of channel losses

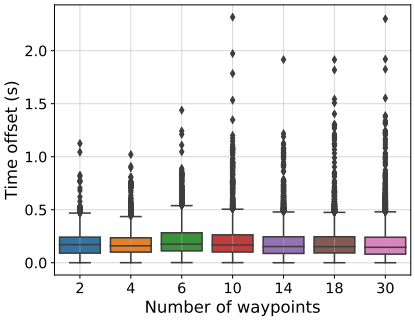
In the previous section we evaluated the performance under ideal channel conditions to determine the impact of the mission complexity and the protocol time



(a) Linear formation.



(b) Matrix formation.

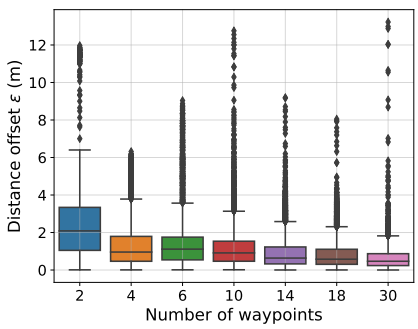


(c) Circular formation.

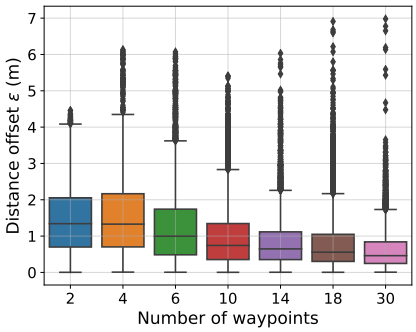
Figure 6.8: Time offset on a swarm of 9 UAVs (lossy channel).

overhead. We now proceed to evaluate our protocol under a more realistic lossy communications channel based on IEEE 802.11 technology, and where the maximum communications distance is of about 1300 meters (channel conditions based on real experiments, as detailed in section 4.1.3). The idea is to maintain the same experimental conditions as in the previous section, and assess the impact of channel losses on performance. Specifically, we use nine UAVs for each simulation. The distance between UAVs was set to 50 meters, and the speed was maintained at 10 m/s. Figures 6.8 and 6.9 show the time and distance offset error for each of the formations evaluated.

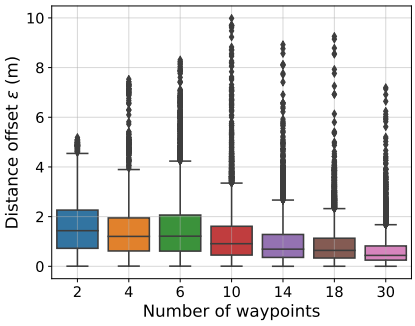
Figure 6.8 shows that, in general, the time offset does not vary significantly when comparing the different formations. However, compared to the results obtained under ideal channel conditions, we can now find outlier values for the delay which are quite higher than for the previous experiments. Besides, these values increase as the complexity of the mission also increases. Specifically, in the linear formation, we can see delay values that are much higher than for the rest of the



(a) Linear formation.



(b) Matrix formation.



(c) Circular formation.

Figure 6.9: Distance offset on a swarm of 9 UAVs (lossy channel).

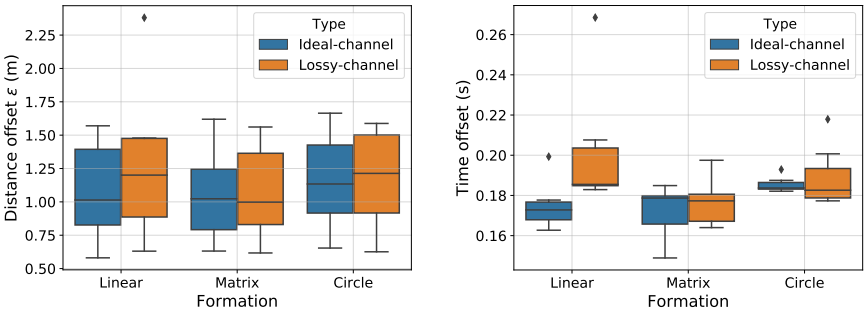
formations. This occurs because, with 9 UAVs, the maximum distance from the farthest UAV to the leader is of about 200 meters.

Figure 6.9 shows that the average error in distance, measured as position offset differences, tends to decrease as the complexity of the mission increases. Specifically, the matrix formation introduces lower errors with respect to the linear and circular formations. In the case of linear training, we can see that it is associated with higher errors; this is due to the higher distances regarding inter-UAV communications, and whose details will be analyzed later.

Table 6.3 shows the mean, maximum, and standard deviation values for each of the evaluated formations under a lossy channel. We can see that, compared to the ideal channel conditions presented earlier, it shows a slight increase in all its values. Figure 6.10 shows the behavior of our proposed protocol using an ideal channel and a lossy channel for each of the formations analyzed, and using the average values obtained from the different missions (with 2, 4, 6, 10, 14, 18, and 30 waypoints). In general, it becomes evident that the time differences between

Table 6.3: Overall simulation statistics (lossy channel).

Formation	Distance offset ε (m)			Time offset (s)		
	Mean	Max.	Std.	Mean	Max.	Std.
Linear	1.2766	13.223	1.1753	0.2020	13.8883	0.1882
Matrix	1.0807	6.9097	0.8625	0.1763	1.8721	0.2667
Circular	1.1804	9.982	0.9920	0.1889	2.3161	0.1370



(a) Formation distance offset (ε).

(b) Formation time offset.

Figure 6.10: Performance comparison between ideal and lossy channel conditions for different formations.

ideal channel and lossy channel conditions are in fact minimal, which validates our MUSCOP protocol under realistic scenarios.

6.3.3 Impact of varying the inter-UAV distances

In the previous analysis, it was shown that the inter-UAV distance affects the formation error in terms of both delay and position offsets. In this section, we adopt the linear formation to provide additional insight into the impact of varying the inter-UAV distances, and thus the channel losses, on the flight formation cohesion. To this end, we vary the distance towards the leader UAV so that it ranges from 100 to 1000 meters. In particular, our flight formation is now limited to 3 UAVs (1 leader plus 2 slaves), so that the distance from all the slaves to the leader is exactly the same. The number of mission waypoints used for this evaluation is 14, and 5 independent simulations were run for each distance measured.

Figure 6.11 shows the time offset when varying the distance between UAVs. In general, it is observed that the mean time offset increases as the separation distance increases. This is expectable since, in a lossy channel, large distances will difficult the synchronization between UAVs, thus becoming a critical problem. It

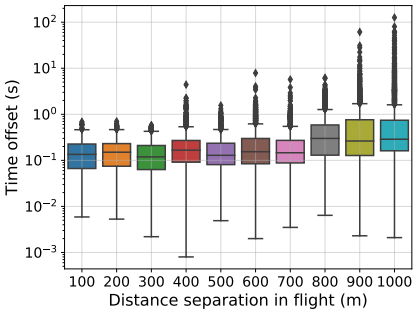


Figure 6.11: Time offset for the linear formation when varying the inter-UAV distance.

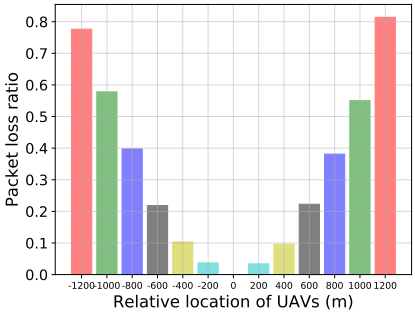


Figure 6.12: Packet loss ratio values at different distances.

is also worth mentioning that, for up to 300 meters between contiguous UAVs, the time offset does not exceed 1 second, which is a quite acceptable value.

6.3.3.1 Packet loss ratio assessment

In the previous evaluation, it was observed that, at long distances, a greater time offset is introduced. Next, we proceed to evaluate the message loss ratio that occurs when using our protocol in the context of a larger swarm. For this purpose, a linear formation with 13 UAVs was used in such a way that each UAV is separated from its neighbors by a distance of 200 meters; hence, the UAVs located at the edges are 1200 meters away from the leader UAV. Several simulations were made, and the mean values were taken. The total number of waypoints for the mission was again equal to 14.

Figure 6.12 shows the packet loss ratio measured at different distances. In general, we can see that the UAVs that are at a greater distance have higher loss

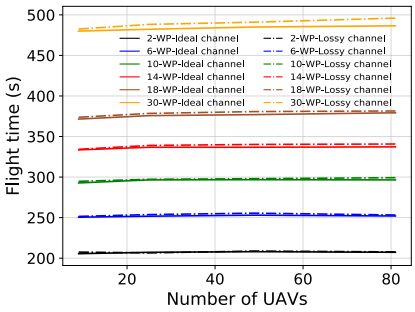


Figure 6.13: Scalability analysis when varying the mission size and the number of UAVs.

levels, as expected, a ratio that even exceeds 80% for the UAVs located in the periphery. These results evidence that ArduSim induces a very realistic model for UAV communications.

6.3.4 Scalability analysis

Finally, we wanted to determine the impact of increasing the number of UAVs and the complexity of the mission in performance. For this purpose, we used the matrix flight formation, and we varied the number of UAVs, testing with 9, 25, 49, and 81 UAVs. For each number of UAVs, several simulations were carried out, having different mission complexity levels: 2, 6, 10, 14, 18, and 30 waypoints. To maintain a high degree of similarity between the different formations tested, we maintained a same distance of 106 meters for the distance between the furthest away UAVs in the matrix and the leader.

Figure 6.13 shows the scalability results when varying the number of UAVs under both ideal and lossy channels. In general, the figure shows that the number of waypoints (WP) is the factor having the greatest impact. In particular, we find that complex missions having up to 30 waypoints make the overall flight time grow beyond 450 seconds. It is also observed that the actual number of UAVs does not have much impact, which means that our solution is highly scalable by efficiently synchronizing all the UAVs in the swarm. In addition, we can also see that the differences between both channel types is minimal, which means that our proposed protocol has a high resilience to packet loss.

To gain further insight on how mission complexity affects the overall flight time, we analyzed the resulting mobility pattern in more detail, finding that more complex missions introduce more acceleration and deceleration events, which causes the average flight speed to become lower, thereby increasing the total time associated to a mission.

6.4 Summary

In this chapter we focus on applications that require the use of UAV swarms to undertake some preplanned missions. To this aim we proposed MUSCOP, a novel protocol able to adequately synchronize all UAVs in a swarm throughout all the steps involved in the flight.

Experimental validation using the ArduSim simulation platform has shown that MUSCOP is effective at maintaining the swarm cohesion for different formations tested, and under different experimental conditions, being highly resilient to channel losses, and able to seamlessly scale to a large number of UAVs without a significant performance penalty. In fact, tests have shown that the complexity of the mission is the main parameter affecting the overall flight times, a factor that is independent of the number of UAVs involved. Nonetheless, the flight time only increases by 0.55 seconds due to the required synchronization on each waypoint when testing with an ideal communications channel, being only slightly higher in the presence of channel losses.

Chapter 7

FollowMe protocol

Technically, a swarm is a group of UAVs powered by artificial intelligence algorithms. UAVs in a swarm communicate with each other while they are in flight, and can dynamically respond to changing conditions autonomously. This chapter focuses on those applications where drone guidance must be manual, not following a pre-planned mission. In this particular case, the different UAVs that make up the swarm have to dynamically adjust their routes in order to follow the master UAV acting as the leader of the swarm. Such a solution may be required in scenarios such as search & rescue [66, 3], fire tracking [65], or the monitoring of disaster areas. In these cases, the pilot must respond to visual stimuli in real-time, and adapt the UAV course accordingly. Our focus on UAV swarms also addresses situations where, in addition to manual guidance, there is a need to carry multiple items or sensors that go beyond the lifting capacity of a single UAV. An example would be a rescue scenario where different UAVs carry food, water, medicine, or shelter. Thus, our proposed application becomes very useful in these situations, by allowing the pilot to control the leader UAV following the usual manual procedures, while seamlessly dragging along the rest of the UAVs conforming the swarm.

One of the main problems to address when creating swarms is communications reliability; whenever a cluster leader is present, such master UAV must maintain an almost real-time synchronization with slave UAVs. The distance separating neighboring UAVs must also maintain consistency to avoid collision problems. Finally, the swarms can also experience a slight lag between the different UAVs, an issue associated to distance and communication disruption, which at times difficults synchronization throughout the whole process.

In this chapter, we propose the FollowMe protocol to define and maintain the

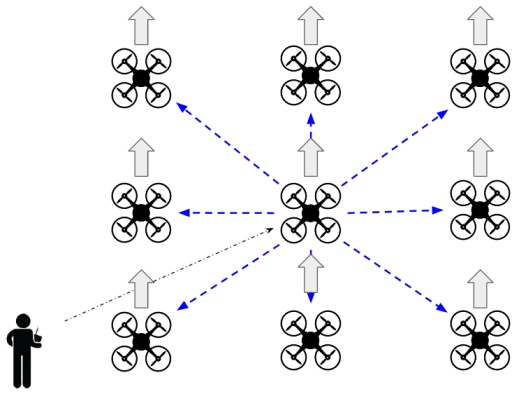


Figure 7.1: FollowMe protocol operation using a matrix formation.

formation of UAVs in a swarm in the specific case where a real pilot controls the swarm leader, and the other UAVs must follow it in real time.

7.1 Protocol overview

7.1.1 Introduction

The FollowMe protocol adopts a master-slave model, where a single UAV is manually controlled by a pilot (master), and several UAVs follow the former automatically (slaves). Figure 7.1 shows the scheme of the FollowMe protocol operation using a matrix formation.

The multicopter acting as swarm leader (master) requires two threads to communicate with the slaves. The *MasterTalker* thread allows commands to be sent to the slaves, and the *MasterListener* thread gathers state information from them. When running the protocol in simulations, an additional thread *(MasterRemote)* is used to simulate the input of a remote control, using the trace of a real flight performed with a quadcopter model GRCQuad from Quaternium [57]. This thread is in charge of controlling the virtual multicopter the same way the remote control does in a real deployment of the protocol.

On the other hand, the slaves receive commands through the *SlaveListener* thread, and send feedback about the progress of the protocol to the master UAV through the *SlaveTalker* thread. In order to achieve the fastest possible response to the commands received from the master multicopter, each slave UAV is controlled from the *SlaveListener* thread, reacting immediately each time a new command is received.

7.1.2 Finite state machine

Figure 7.2 represents the finite state machine that rules the behavior of the master and slave UAVs. The upper and lower lines show the progress of the protocol in the master and the slave multicopters, respectively. Above each step, a curve line shows the action performed by the talker thread on the UAV, while the lower line shows the message the UAV is waiting for at each particular state.

The master multicopter starts by waiting until the flight controller is ready to accept commands *(UAVs ready)* in the *Start* state. Then, it waits for the slaves to signal their presence (message *ID*) in the *Wait slaves* state. When the master detects all the slaves, the user can press the setup button *(Setup started)*, going to the *Wait takeoff* state. The *TakeOff* message is issued with enough information to let the slaves know their location in the flight formation. The master UAV is ready to fly when all the slaves reach their formation location (message *Ready* received from all of them). The master UAV starts the flight when the user presses the start button *(Test started)*, and the pilot performs manually the takeoff. Once it reaches enough altitude, it changes to the *Follow me* state, and then it starts to issue messages *(Coordinates)* that allow the slaves to follow the leader flight pattern. Finally, the flight ends by changing to the *Landing* state, sending the slaves the *Land* message to finish their flight. In simulation, this happens when the real recorded trace used to feed the master UAV finishes, and in a real multicopter when the user changes the flight mode to *Land*. The master UAV only waits for messages from the slaves during the initial steps. Since the master-slave model gives this multicopter full control over the rest of the UAVs while they are flying, the listener thread becomes useless when reaching the *Ready* state, and so it is terminated.

Regarding the slave UAVs, they also start by waiting in the *Start* state until the flight controller is ready to receive commands *(UAVs ready)*. Then, they remain in the *Hello* state, signaling their presence with their own message *ID* until the takeoff command (*TakeOff* message) is received from the master UAV. All slaves remain silent until they reach the target location in the formation with respect to the master UAV, and then they switch to the *Wait master* state, meaning that they are ready to follow the master (*Ready* message). When a UAV receives commands to follow the master (*Coordinates* message), it changes to the *Following* state, going after the master UAV until the landing command (*Land* message) is received. Notice that the talker thread is not needed once the UAV starts to follow the master (*Following* state), and it is terminated, similarly to the listener thread, when the land command is received.

7.1.3 Message format

Several messages are needed to coordinate the behavior of the master and the slave multicopters. Figure 7.3 shows the format of these messages. Fields *type* and *ID*

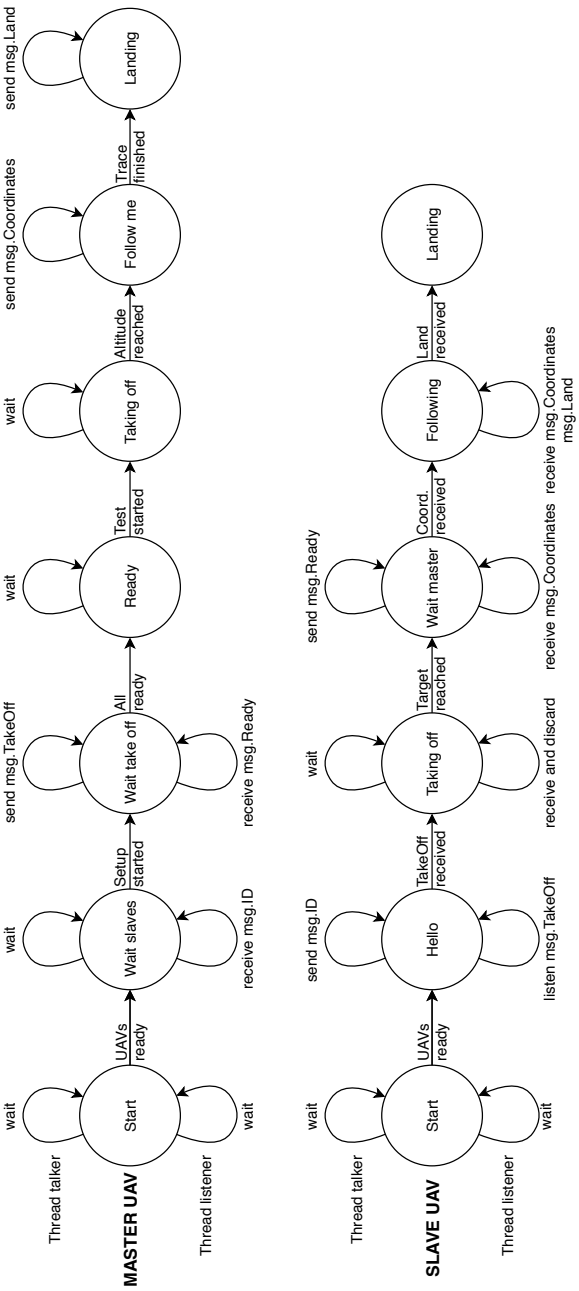


Figure 7.2: FollowMe protocol finite state machine.

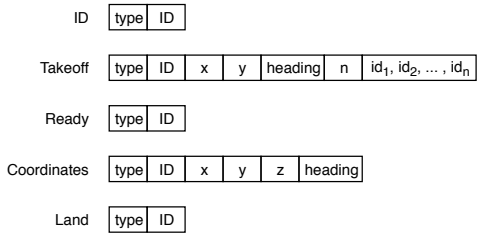


Figure 7.3: FollowMe message types.

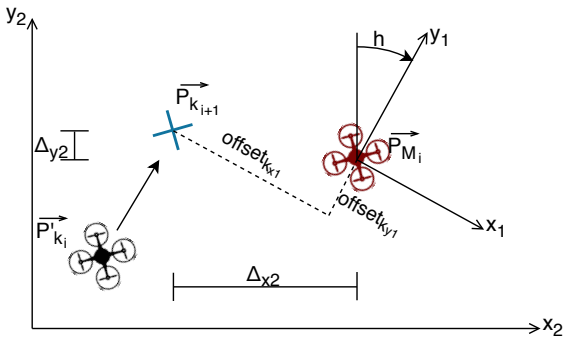


Figure 7.4: FollowMe protocol: target location calculation.

are common to all messages, and represent the message type, and the identifier of the sender UAV, respectively.

The message *ID* is used by slave UAVs to inform the master UAV about their presence, in order to join the swarm at the beginning of the process.

When all the slaves are detected by the master UAV, and the user starts the setup step of the simulation, the *TakeOff* message is issued by the master UAV. It adds its current coordinates and heading using the Universal Transverse Mercator (UTM) coordinate system, as well as an ordered list of the UAV identifiers of the slaves that were previously detected. This information allows a slave k to know its theoretical position in the flight formation $\overrightarrow{P_{k_{i+1}}}$, besides the current location of the master $\overrightarrow{P_{M_i}}$. Then, the offset between the target location of the UAV and the master (see Figure 7.4) is calculated considering the current heading of the master h, and the slave takes off and moves to the designated location in the formation $\overrightarrow{P_{k_{i+1}}}$.

Message *Ready* is sent by the slaves to the master UAV when the takeoff process finishes. The master UAV cannot start the flight until all the slaves have taken-off and are ready at their expected relative positions, surrounding its current location. Only then can the manually controlled flight start.

Once the master UAV reaches the same altitude as the rest of the UAVs in the swarm, it periodically broadcasts message *Coordinates* during its flight, which includes the current 3D location and heading of the master. Each time a slave receives this message, it calculates a new target location $\overrightarrow{P_{k_{i+1}}}$, and issues a command to the flight controller to move to the designated location in the formation.

$$\overrightarrow{P_{k_{i+1}}} = \overrightarrow{P_{M_i}} + \overrightarrow{\Delta_{k_i}} \tag{7.1}$$

where

$\overrightarrow{\Delta_{k_i}} = (\Delta_{x2}, \Delta_{y2})$

$\Delta_{x2} = \mathit{offset}_{k_{x1}} \cdot \cos(h) + \mathit{offset}_{k_{y1}} \cdot \sin(h)$

$\Delta_{y2} = \mathit{offset}_{k_{y1}} \cdot \cos(h) - \mathit{offset}_{k_{x1}} \cdot \sin(h)$

On each iteration i, the offset of a slave k with respect to the master UAV in frame 1 remains constant to achieve swarm cohesion, while the target coordinates of that slave ($\overrightarrow{P_{k_{i+1}}}$) at instant $t+1$ are calculated considering the current location of the master UAV ($\overrightarrow{P_{M_i}}$) at instant i, and a translation relative to the master UAV ($\overrightarrow{\Delta_{k_i}}$), calculated with the theoretical relative location of the UAV towards the master $\left(\mathit{offset}_{k_1}\right)$, and the current heading *(h)* of the master. The slave will need time to reach the target location at instant $i+1$, as calculated for the current location of the master at instant i, and so a slight delay between the master and its slaves is expected.

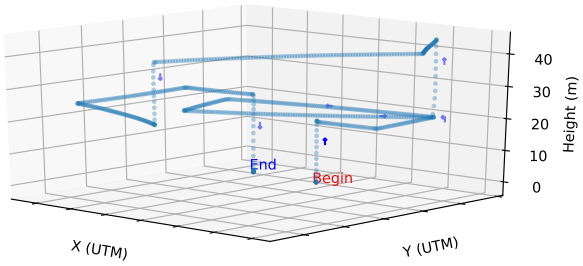
Finally, message *Land* is sent by the master to all the slaves at the end of the experiment to force them to land in a coordinated manner.

7.2 Data sources and error assessment

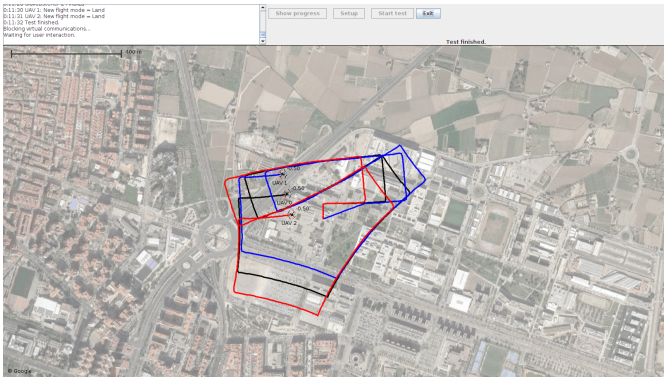
Once the FollowMe protocol was introduced, we now proceed to validate its correctness. To this end, we will first provide details about the source data used, as well as the process followed to achieve data synchronization. Then, we will describe the method followed to determine the errors associated to our tests. Next, we present the details of the data source used, and the methodology for estimating the error.

7.2.1 Data source

For the proposed protocol to come into operation, a real data source is needed so that the master drone can guide the swarm. To this purpose, a UAV assembled by our own research group was used, which allowed us to capture the commands issued by the pilot along the flight path. The captured dataset stores approximately 11 flight minutes. The UAV was piloted in manual mode, making several turns in order to obtain a more realistic data set. Figure 7.5 (a) shows the trajectory of



(a) 3D view.



(b) Upper view using ArduSim.

Figure 7.5: Real-time data source.

the data source used in the simulation, and Figure 7.5 (b) shows the actual flight trajectory in a real map.

As complementary information, to gain further insight into the flight pattern generated, Figure 7.6 shows how different variables have evolved throughout time. It is noteworthy that, in Figure 7.6 (top and down), there are several variations regarding speed and acceleration, respectively. This is due to the different turns the pilot made when generating the trace. Figure 7.6 (left) also shows the three changes of altitude taking place in our reference dataset.

Once the dataset was obtained and used as input, different simulations were performed. Notice that ArduSim generates a set of data for each drone in the simulation. The information generated by ArduSim allows us to obtain UTM coordinates, height, speed, acceleration, heading, and time. In our evaluation set, the location registered at the beginning of the experiment is taken as the origin. Then, we use the range of time for which flight data is available for all the drones

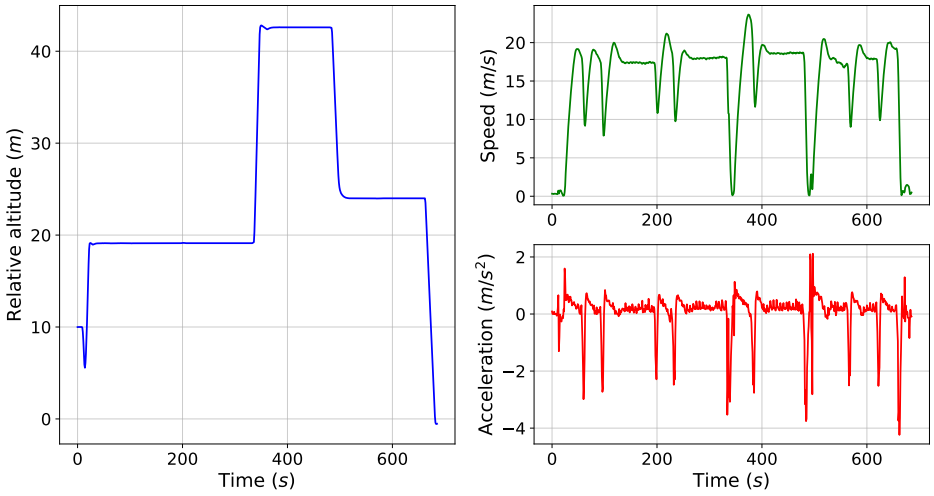


Figure 7.6: Data source variables

in the simulation, and afterward we proceed to perform the interpolation for the simulated dataset at regular time intervals.

7.2.2 Error analysis

Having obtained a simulated and synchronized dataset, we now proceed to describe the methodology for calculating the error. We measured two types of errors: (i) the swarm formation error (σ), and (ii) the global error (ε). Figure 7.7 illustrates the error regarding the stability of the formation (left), as well as the global error (right). The point marked as X (light blue) represents the calculated theoretical value for the different UAV positions, and the yellow line represents the calculated errors regarding the positions of the slave UAVs (black), and the master UAV (red colour). Below we provide details about the calculations we made.

The formation error (σ) refers to the theoretical UAV layout centered on the mean location for all the slave UAVs $\overline{m_i}$, where the formation is built around $\overline{m_i}$ taking as reference the heading for the master UAV.

$$\overline{m_i} = (\overline{m_{x_i}}, \overline{m_{y_i}}), \text{ where } \begin{cases} \overline{m_{x_i}} = \frac{\sum_{k=1}^{n} x_k}{n} \\ \overline{m_{y_i}} = \frac{\sum_{k=1}^{n} y_k}{n} \end{cases} \tag{7.2}$$

In equation 7.2, i represents the current time step, k represents the different slave UAVs, and n is the total number of slave UAVs.

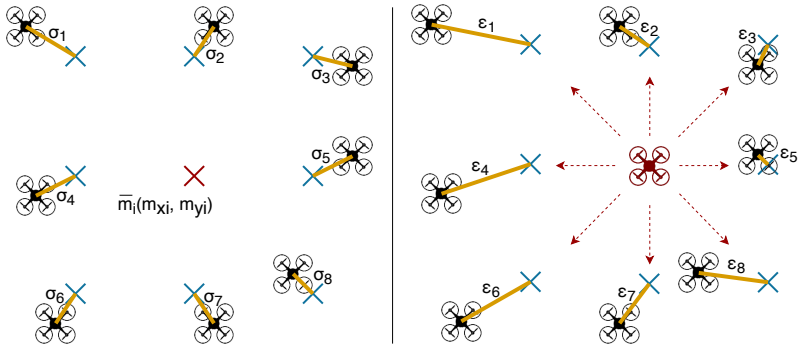


Figure 7.7: FollowMe protocol: Formation stability error (left), and global error (right).

The theoretical position of a slave UAV $\overrightarrow{P_{k_i}}$ calculated for time instant i is obtained with $\overrightarrow{\Delta_{k_i}}$ from equation 7.1, and equation 7.3:

$$\overrightarrow{P_{k_i}} = (x_{k_i}, y_{k_i}), \text{ where } \begin{cases} x_{k_i} = \overline{m_{x_i}} + \Delta_{x2} \\ y_{k_i} = \overline{m_{y_i}} + \Delta_{y2} \end{cases} \tag{7.3}$$

The actual position of a specific UAV is $\overrightarrow{P'_{k_i}} = (x'_{k_i}, y'_{k_i})$ on the period of time i. Notice that x'_{k_i} is the real value obtained by the simulator in the x axis, and y'_{k_i} is the real value obtained in the y axis, for each slave UAV.

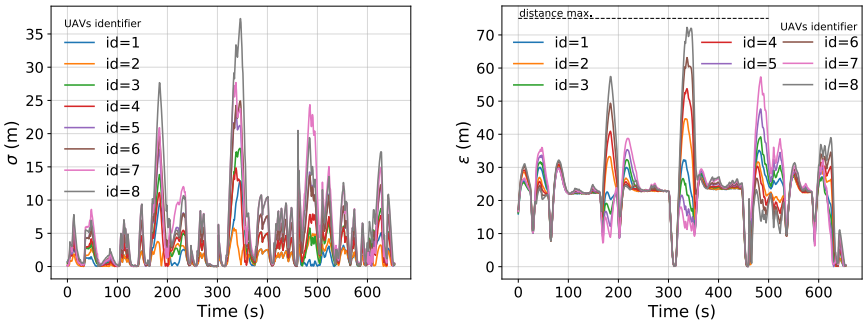
Finally, the equation to calculate the error of a single UAV in the swarm layout is defined as:

$$\sigma_{k_i} = \sqrt{(x_{k_i} - x'_{k_i})^2 + (y_{k_i} - x'_{k_i})^2} \tag{7.4}$$

To calculate the global swarm layout error (ε), we replace the mean location of the slaves $\overline{m_i}$ with the current location of the master UAV $\overrightarrow{P_{M_i}}$ in equation 7.3. Finally, ε is calculated the same way as σ through equation 7.4.

7.3 Protocol validation

Once the data source and the methodology used to calculate the errors were defined, we proceeded to evaluate our proposed protocol. To achieve this, we conducted an extensive set of experiments using different numbers of UAVs, different distances between UAVs, different formations, and different update intervals of the information sent from the master UAV. In this sense, we will describe our evaluation set in 4 parts: i) Swarm with linear formation, ii) swarm with matrix formation, iii) swarm with varying communication settings, and finally iv) swarm



(a) Formation error (σ). (b) Global swarm layout error (ε).

Figure 7.8: Error on a swarm of 9 UAVs using a linear formation, and with a separation between neighbors of 75m.

comparing the different proposed formations. Below we provide details about the results obtained.

7.3.1 Swarm with linear formation

We used the default settings of the ArduSim simulation platform to evaluate two different linear approaches: i) formation evaluation using nine UAVs, and ii) formation evaluation using different numbers of UAVs.

7.3.1.1 Formation evaluation using nine UAVs

In this first evaluation, we launched nine drones with the speed set to 15 m/s, and a separation distance between multicopters equal to 75 meters. Several simulations were made, and the average of the whole set of all the tests was taken. Figure 7.8 shows both the stability error in the swarm formation (a), and the global error (b), for each of the slaves of the formation. Figure 7.8a shows that, most of the time, the values do not exceed the average obtained of 3.85m, while the behavior in (b) is very similar, being that, at specific times, some values may approach the maximum distance threshold established for the simulation. The global average error is 22.61 meters. We also get this value while flying at constant speed, due to a delay of about 4 seconds between master and slaves. That effect appears because the slaves calculate their target location accounting for the current location of the master drone, instead of its location in a near future.

In general, it is observed that many of these high error values occur when there are changes in speed, and the UAVs take several turns along their path, thus representing worst-case conditions. Also, both figures 7.8 (a) and (b) show that the UAV IDs that are the furthest away from the master UAV tend to experience

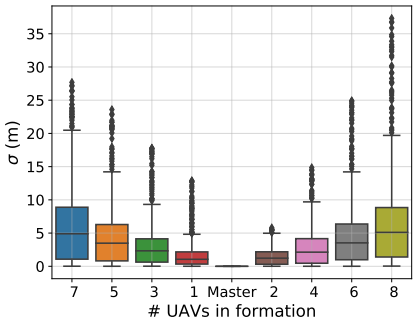


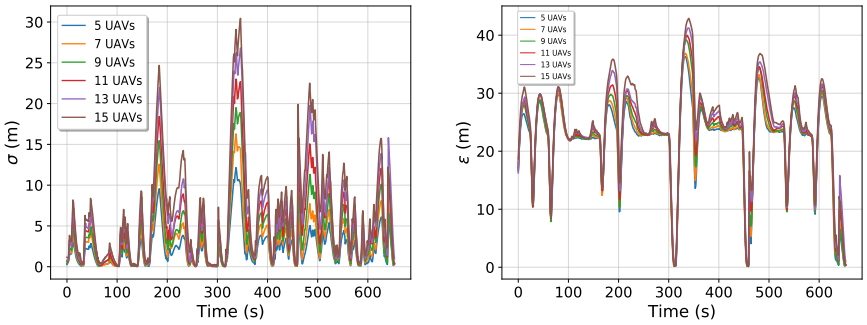
Figure 7.9: Box and whisker plot of the formation error (σ) of the 9 UAVs of the swarm.

higher errors. For this reason, we decided to evaluate the behavior of each of the slave drones in more detail. Figure 7.9 shows the linear formation that the drones perform in the simulation according to the UAV ID. The leading (master) drone received ID=0. It is worth noticing that, the greater the distance towards the master, the bigger the error values become. This is due to small fluctuations in the heading parameter of the master UAV.

7.3.1.2 Formation evaluation using different numbers of UAVs

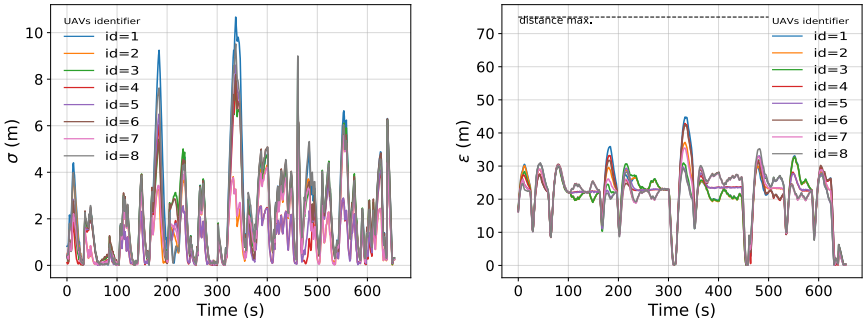
In the second analysis, we wanted to compare formation and global errors for different numbers of drones. The same separation distance and default configurations established in the previous evaluation were used. Seven simulations were performed with 3, 5, 7, 9, 11, 13, and 15 drones, respectively. Figure 7.10 shows the formation and global errors for these numbers of UAVs. Figure 7.10 (a) evidences that, as the number of UAVs increases, the formation error tends to grow as well. Figure 7.10 (b) shows that most of the route has an average error of 22.84 meters, although higher values (above 40 meters) can take place in simulations with 13, and 15 UAVs.

Regarding the evaluations discussed above, we find that our developed protocol is capable of achieving the desired flight pattern. In terms of swarm formation errors, they are due to the following factors: i) Slave drones calculate their target position accounting for the current location of the master drone, taking a few seconds to reach that new position (up 4 seconds), which introduces a delay; ii) It was evidenced that the greater is the distance from the slave drones to the master, the higher the error becomes, an issue that is mostly due to small fluctuations in the heading parameter of the master UAV.



(a) Mean formation error (σ). (b) Mean global swarm layout error (ε).

Figure 7.10: Mean error for all the UAVs in a swarm of n drones using a linear formation, and a neighbor separation of 75m.



(a) Formation error (σ) formation. (b) Global swarm layout error (ε).

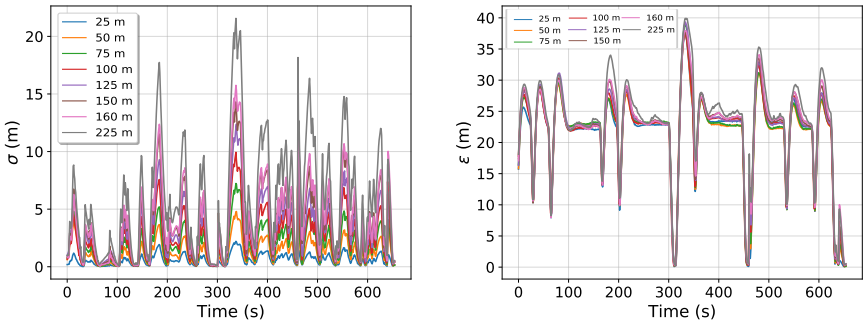
Figure 7.11: Error on a swarm of 9 UAVs using a matrix formation, and for an inter-UAV separation of 75 m.

7.3.2 Swarm with matrix formation

In this section, we evaluate the swarm performance when adopting a matrix layout for the swarm. To this aim we use three approaches: Evaluation of (i) formation errors when simulating nine UAVs, (ii) Matrix formation error when adopting different distances between UAVs, and (iii) using different numbers of UAVs.

7.3.2.1 Formation evaluation using nine UAVs

We used the same configuration described in the linear formation, but changing the swarm layout to a matrix formation. Figure 7.11 shows the results regarding



(a) Formation error (σ). (b) Global swarm layout error (ε).

Figure 7.12: Error on a swarm of 9 drones using a matrix formation, varying the distance of separation between drones.

distance errors in the swarm. In general, the results show that, for the matrix formation, the errors are much lower than those previously obtained for the linear formation. In particular, Figure 7.11 (a) shows that the errors are less than 10 meters for most part of the route traveled in the simulation. Specifically, the average error obtained with this formation is 1.83 m. Figure 7.11 (b) shows that the global distance error is far from the distance between UAVs we have set, having an average of 21.67 meters throughout the whole path.

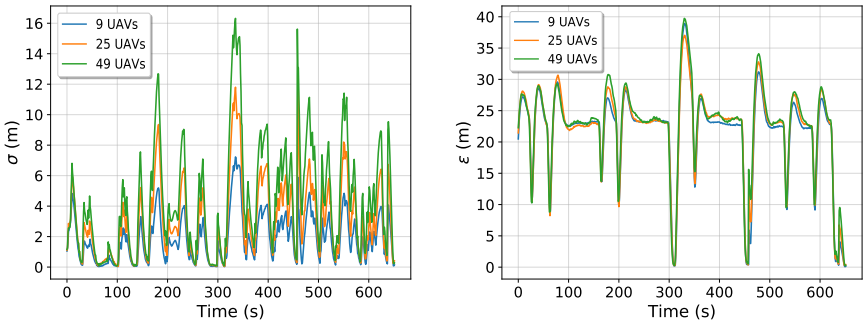
7.3.2.2 Evaluating swarm cohesion when varying the inter-UAV distances

After the previous analysis, we proceeded to evaluate the error by varying the distance between UAVs, and using the same configuration of 9 drones. In this test we used several separation distances: 25, 50, 75, 100, 125, 150, 160, and 225 meters. We consider the maximum distance of the master drone equivalent to the maximum in the linear formation previously analyzed.

Figure 7.12 (a) shows that the formation error magnitude is directly related to the separation distance, but, at the distances evaluated, values do not exceed 14 meters in general. Figure 7.12 (b) shows that the behavior does not vary much with distance, and the average value of the error obtained is 21.82 meters.

7.3.2.3 Formation evaluation using different number of UAVs

In the previous results it has been proven that, for larger inter-UAV distances, the formation error grows, being the matrix swarm more stable. Starting from this point, we now want to check the swarm performance for a matrix layout when



(a) Mean formation error (σ). (b) Mean global swarm layout error (ε).

Figure 7.13: Mean error for all the UAVs in a swarm of n drones using a matrix formation and an inter-UAV distance of 75 m.

having 9, 25, and 49 UAVs, while maintaining the same distance between them (75 meters).

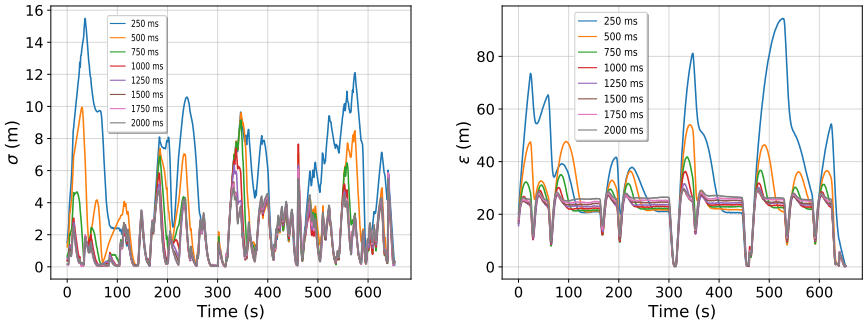
In Figure 7.13 (a) we observe that the errors in this formation do not exceed 18m, becoming higher when a matrix formation having 49 UAVs is used. In Figure 7.13 (b) the variability detected can be considered small, and the average error found is 21.58 m.

In general, experimental results so far show that the matrix formation approach is more effective in reducing formation errors compared to the linear formation pattern. The reason is that the matrix pattern allows minimizing the distance between the master and the different slaves, provided that the master occupies a central position with respect to the slaves. In addition, since the slave drones are closer to the master, the delays associated to the transmission and processing of information are lower than those obtained in the linear formation.

7.3.3 Formation error when varying the position refresh period

In this part, we want to evaluate how our protocol behaves if we vary the position update period in terms of communications. For this analysis we used the matrix formation. Different simulations were performed using nine UAVs with a distance between them of 75 m. The refresh time values evaluated were: 250ms, 500ms, 750ms, 1000ms, 1250ms, 1500ms, 1750ms, and 2000ms.

Figure 7.14 shows the results obtained. In general we observe that, the shorter the update period, the higher the evaluation error becomes. In particular, Figure 7.14 (a) shows that, with a refresh period of 1000ms, the performance is optimal. Figure 7.14 (b) shows that, when using update values between 1000ms and 1500ms, the error tends to decrease in those peaks that correspond to situations where the



(a) Mean formation error (σ). (b) Mean global swarm layout error (ε).

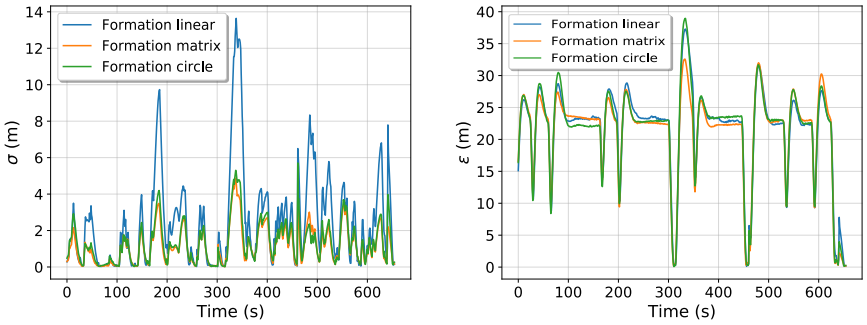
Figure 7.14: Mean error for all the UAVs in a swarm of 9 drones using a matrix formation, varying the network refresh period.

Table 7.1: Errors values using different refresh periods.

Refresh period (ms)	$\overline{\sigma}$ (m)			$\overline{\varepsilon}$ (m)		
	Mean	Max.	Std.	Mean	Max.	Std.
250	5.31	8.60	2.19	37.85	44.80	5.02
500	3.07	5.01	1.22	26.57	30.87	2.89
750	2.18	3.09	0.70	22.74	25.45	1.97
1000	1.84	2.52	0.54	21.70	23.87	1.60
1250	1.75	2.30	0.47	21.67	23.61	1.50
1500	1.70	2.15	0.43	22.06	23.87	1.44
1754	1.74	2.18	0.43	22.74	24.54	1.46
2000	1.77	2.21	0.43	23.43	25.42	1.47

UAVs slow down.

Table 7.1 shows the results of the average formation errors obtained, evidencing the maximum values in this evaluation.The results show that having the master UAV announcing its current position more than once per second is counterproductive, and is prone to increase errors. Also, it becomes evident that refresh rates of 1250 ms allow us to optimize performance, having a maximum general error value of 23.61, and a mean value of 21.67. Overall, we find that, beyond this value, the longer it takes for slave UAVs to receive the information, the longer it will take for them to react and place themselves in their expected location in the formation. So, in general, the error tends to be greater.



(a) Mean formation error (σ). (b) Mean global swarm layout error (ε).

Figure 7.15: Mean error for all the UAVs in a swarm of 9 drones at a separation distance of 50 m, varying the formation type.

7.3.4 Formation error for linear, matrix, and circular formations

Finally, we decided to assess the effectiveness of our protocol when comparing the three formations: linear, matrix and circular. In addition, the collision control system of ArduSim was enabled. The idea is to check if, at a considerable distance, our proposal presents UAV collision problems. The previous validation tests do not include results for the circular formation because the behavior along each experiment was quite similar to the matrix formation results. The distance adopted for this set of simulations was 50 meters. Thus, the parameters established for the collision system were: the distance threshold was set to 10 meters, and the altitude threshold to 20 meters.

Figure 7.15 (a) and (b) show the results obtained for both these types of errors. In general, a smaller distance error is observed when the circular and matrix formations were used, compared with the linear formation, where higher error values are expected. As explained before, the results obtained for matrix and circular formations are quite similar.

Table 7.2 shows the results obtained for the formation and global swarm layout errors. The results obtained favor swarms adopting a matrix formation, as distances towards the master UAV are minimized. We found that the matrix approach was more effective in reducing formation errors compared to linear or circular patterns.

Table 7.2: Errors values for the three formations.

Formation	$\overline{\sigma}$ (m)			$\overline{\varepsilon}$ (m)		
	Mean	Max.	Std.	Mean	Max.	Std.
Linear	2.48	4.42	1.18	21.76	24.96	1.90
Matrix	1.20	1.60	0.35	21.31	22.70	1.06
Circular	1.34	1.77	0.31	21.67	23.30	1.14

7.4 Summary

In this chapter we proposed the FollowMe protocol, a solution to coordinate UAV swarms where the leader UAV is manually controlled by a pilot, and the remaining UAVs (slaves) must attempt to follow its mobility pattern in real time.

To validate and assess the performance of the proposed protocol, we first recorded a real trace of control inputs when a real pilot was controlling a UAV. Then, this trace was used as input to ArduSim, our multi-UAV simulation platform.

Our results have shown that the developed protocol is able to achieve the desired functionality, as expected. Regarding the swarm formation errors, we found that these errors are mainly due to two causes: first, the lag between the master and the followers, expectable due to the different delays involved in transmission and information processing, which can be of up to 4 seconds; and second, due to the fluctuations associated to the heading parameter for the master UAV, which where amplified for the slave UAVs, especially those located far away from it.

In terms of position updates, we also found that having the master UAV advertise its current position more than once a second is counterproductive, being prone to increase errors.

In terms of swarm formations, we found that the matrix approach was more effective at reducing formation errors when compared to linear or circular formation patterns.

Chapter 8

Conclusions, Publications and Future Work

Low cost multicopters are spreading around the world due to many new applications not even envisioned before, and new UAV coordination protocols are needed to satisfy the requirements of these applications. In this thesis we proposed ArduSim as a multi-UAV simulation platform that allows developers to design and implement such protocols, porting them to real multicopters in a straightforward manner, reducing the time and money needed during the research phase. Once a protocol is tested and validated on real multicopters, it can be implemented in dedicated hardware. We also propose three protocols to coordinate the flight of UAVs, all of them developed and validated with ArduSim.

Below we briefly summarize the most relevant contributions of this thesis:

- **ArduSim Simulation Platform**. We proposed a platform to design and implement novel protocols to coordinate the flight of multicopters. ArduSim can run up to 100 UAVs in soft real-time in a high-end computer, which enables to test a wide range of novel protocols that would otherwise be hard to validate without assembling a large number of multicopters.

  ArduSim also includes a realistic WiFi model based on real experiments performed in the 5 GHz frequency band, where the packet losses are directly related to the distance between UAVs. We also modeled the wireless channel occupancy through a carrier sensing mechanism, and included the possibility of detecting collisions of data packets.

- **MBCAP Protocol**. We presented a collision avoidance solution for multicopters that are following independent planned missions. The UAVs broadcast their current and predicted future locations, calculated considering the path they must follow. A collision risk is detected if there is a match between the UAV future locations and those included in the messages received from other UAVs. In such case the UAVs stop, and the low-priority UAV waits while the high-priority UAV goes beyond the risk area, resuming its own mission afterwards.

  To validate our proposal we performed tests with real multicopters performing independent missions, and we performed a thorough set of simulation tests in highly crowded environments (100 UAVs) with a success ratio of 98.22%. We also found that the flight time overhead introduced by the protocol is quite low and well bounded.

- **MUSCOP Protocol**. Some applications can be optimized through cooperation, parallelizing a sensoring task, like large-scale agriculture in search of pests or weeds. To this end, we proposed a protocol where a master UAV synchronizes the slave UAVs each time they reach an intermediate point in the mission, maintaining the formation cohesion for different formations tested. Results showed that MUSCOP is highly resilient to channel losses, and it is able to seamlessly scale to a large number of UAVs without a significant performance penalty.

- **FollowMe Protocol**. Similarly to the former protocol, FollowMe can speedup tasks that need manual guidance, like searching for missing persons in wide areas. The proposed solution is also based in the master-slave model, where the slaves mimic in real-time the movements of the leader that is manually guided by a pilot using a standard remote control.

  To validate our proposal, we recorded a real trace of a flight of a multicopter, and we used it as input for the leader UAV in ArduSim. We measured two type of errors: first, the swarm formation error, i.e., the relative error of the slaves in the formation, and second, the global error of all the UAVs in relation to the theoretical formation centered in the leader UAV. We found that having the master UAV advertise its current location more than once a second is counterproductive, and that the errors are mainly due to two causes: first, the delay involved in the transmission from master to slaves and the processing of messages; and second, detected fluctuations in the heading parameter for the master UAV, that where amplified for the slaves, being the effect increased for the UAVs located far away from the former.

Having accomplished all our objectives, the original goal of this thesis has been achieved, and so this dissertation can now be concluded. The next section

enumerates the publications related to this thesis. The last section of this chapter refers to some open issues for the future.

Publications

This section lists the publications that have been produced as a result of this thesis, as well as some other collaborations and related publications we published during this time.

International Journals

- Fabra, F., Calafate, C. T., Cano, J. C., & Manzoni, P. (2018). ArduSim: Accurate and real-time multicopter simulation. Simulation Modelling Practice and Theory, 2018. **I.F. 2018: 2.426; JCR: Q2 Category**.

- Fabra, F., Zamora, W., Masanet, J., Calafate, C. T., Cano, J. C., & Manzoni, P. (2019). Automatic system supporting multicopter swarms with manual guidance. Computers & Electrical Engineering, 2019. **I.F. 2018: 2.189; JCR: Q2 Category**.

- Fabra, F., Zamora, W., Sangüesa, J., Calafate, C. T., Cano, J. C., & Manzoni, P. (2019). A Distributed Approach for Collision Avoidance between Multirotor UAVs Following Planned Missions. Sensors, 2019. **I.F. 2018: 3.031; JCR: Q2 Category**.

- Fabra, F., Zamora, W., Reyes, P., Sangüesa, J., Calafate, C. T., Cano, J. C. & Manzoni, P. (2019). MUSCOP: Mission-based UAV Swarm Coordination Protocol. Submitted to Computer Communications, 2019. **I.F. 2018: 2.766; JCR: Q2 Category**.

- Fabra, F., Sangüesa, J., Zamora, W., Calafate, C. T., Cano, J. C. & Manzoni, P. (2019). UAV deployment: challenges, applications, and collaborative solutions based on wireless communications. Submitted to IEEE Vehicular Technology Magazine, 2019. **I.F. 2018: 6.145; JCR: Q1 Category**.

International Conferences

- Fabra, F., Calafate, C. T., Cano, J. C., & Manzoni, P. (2017, January). A methodology for measuring UAV-to-UAV communications performance. In 14th IEEE Consumer Communications and Networking Conference (CCNC), 2017 (pp. 280-286). IEEE. **Core B**.

- Fabra, F., Calafate, C. T., Cano, J. C., & Manzoni, P. (2017, June). On the impact of inter-UAV communications interference in the 2.4 GHz band. In

13th International Wireless Communications and Mobile Computing Conference (IWCMC), 2017 (pp. 945-950). IEEE. **Core B**.

- Fabra, F., Calafate, C. T., Cano, J. C., & Manzoni, P. (2018, April). A Collision Avoidance Solution for UAVs Following Planned Missions. In IEEE Wireless Communications and Networking Conference (WCNC), 2018 (pp. 55-60). IEEE. **GRIN-SCIE class 2**.

- Fabra, F., Calafate, C. T., Cano, J. C., & Manzoni, P. (2018, May). MBCAP: Mission Based Collision Avoidance Protocol for UAVs. In 32nd IEEE International Conference on Advanced Information Networking and Applications (AINA), 2018 (pp. 579-586). IEEE. **Core B**.

- Fabra, F., Zamora, W., Reyes, P., Calafate, C. T., Cano, J. C., Manzoni, P. & Hernández, E. (2019, Jul). An UAV Swarm Coordination Protocol Supporting Planned Missions. In 28th International Conference on Computer Communications and Networks (ICCCN), 2019 (pp. 1-9). IEEE. **Core A**.

National Conferences (Spain)

- Fabra, F., Calafate, C. T., Cano, J. C., & Manzoni, P. (2018, September). MBCAP: Protocolo de Evitación de Colisiones para Vehículos Aéreos No Tripulados. In XXIX Jornadas de Paralelismo, 2018.

- Fabra, F., Reyes, P., Calafate, C. T., Cano, J. C., Manzoni, P. & Zamora, W. (2019, September). Protocolo de coordinación de enjambres de VANTs para misiones planificadas. In XXX Jornadas de Paralelismo, 2019.

Future Work

In this thesis we proposed ArduSim as a simulation platform for the development of new UAV coordination protocols. Regarding this simulator, we propose the following tasks as future work:

- **UAV-to-UAV simulated communications**. We included a model based on WiFi ad-hoc networks, using the 802.11a protocol, and specifically with 5 dBi antennas. As this setup is not a general usage case, we plan to extend the application by integrating new communication models based on different types of antennas and wireless technologies. We will also use flight traces obtained with ArduSim to analyze in more detail the performance of wireless communications in OMNeT++, and thus develop more realistic communication models. Studies about the influence of different factors in the communications link quality between UAVs have not been completed, as the results in the 2.4 GHz band were masked by the interference caused by

remote controls. We plan to repeat all the experiments in the 5 GHz band to analyze the relative influence of such factors.

- **Sensors integration**. We also plan to implement interfaces to allow the developer to use virtual sensors (CO2, ozone, LIDAR, etc.). This functionality will help to test new protocols that require sensors before deploying them in real multicopters.

- **Swarm protocols safety**. UAVs are prone to collisions while they takeoff or land in a small area, and so any swarm protocol should automate the takeoff and landing of all the UAVs in a safe manner. We plan to integrate in ArduSim a protocol to achieve a safe takeoff prior to applying any swarm-based protocol, like *MUSCOP* or *FollowMe*, and another protocol to bring slave UAVs closer to the leader before landing, using an integrated camera to land with high accuracy in a specific place, and avoiding collisions.

Regarding the MBCAP protocol, we found that collisions only take place with MBCAP-e when multiple UAVs are implied in a conflicting situation, and one of them remains stopped along the path of the high-priority UAV while taking over, a situation associated to the priorities adopted for the collision avoidance strategy. As future work, we plan to study alternative collision avoidance strategies based on safety areas, like artificial potential fields, and roundabout-like manoeuvres [20], to further enhance the effectiveness of MBCAP-e at avoiding collisions, and to reduce the time overhead involved in collision avoidance manoeuvres.

Additionally, we proposed two swarm coordination protocols: *MUSCOP* and *FollowMe*. As future work, we plan to validate both protocols in a testbed using real UAVs. We also plan to reduce the swarm formation error in the *FollowMe* protocol by introducing predictions regarding the future location of the leader UAV.

Acronyms

ArduSim Arducopter Simulator . . . 12
GCS Ground Control System . . . 1
SES Single European Sky . . . 3
EU European Union . . . 3
MAV Micro Aerial Vehicle . . . 8
MUAV Mini Unmanned Aerial Vehicle . . . 8
LASE Low Altitude, Short Endurance . . . 8
LALE Low Altitude, Large Endurance . . . 8
MALE Medium Altitude, Large Endurance . . . 8
HALE High Altitude, Large Endurance . . . 8
LOS Line Of Sight . . . 8
VTOL Vertical Take Off and Landing . . . 8
ESC Electronic Speed Controller . . . 9
PWM Pulse Width Modulation . . . 10
PPM Pulse Position Modulation . . . 10
SDDL Scalable Data Delivery Layer . . . 15
GS Ground Station . . . 15
DDDAS Dynamic Data-Driven Application System . . . 16
MAVLink Micro Air Vehicle Link . . . 10
TCP Transport Control Protocol . . . 12
ns-2 The Network Simulator . . . 11
OMNeT++ Objective Modular Network Testbed in C++ . . . 11

MBCAP Mission Based Collision Avoidance Protocol 14
MUSCOP Mission-based UAV Swarm Coordination Protocol 16
DTN Delay-Tolerant Network 1
LIDAR Laser Imaging Detection and Ranging 14
API Application Programming Interface 22
SITL Software In The Loop 23
ACK acknowledgment 26
PIC Pilot in Command 29
UTM Universal Transverse Mercator 30
CSMA/CA Carrier Sense Multiple Access with Collision Avoidance 36
FIFO First In First Out 37
ESC Electronic Speed Controller 9
UAV Unmanned Aerial Vehicle 1

Bibliography

[1] F. A. Administration. *Automatic Dependent Surveillance-Broadcast (ADS-B)*. https://www.faa.gov/nextgen/programs/adsb/, accessed 6/3/2019 (cited on pp. 13, 19, 62).

[2] D. Albani, J. IJsselmuiden, R. Haken, and V. Trianni. "Monitoring and mapping with robot swarms for agricultural applications". In: *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. Aug. 2017, pp. 1–6. DOI: 10.1109/AVSS.2017.8078478 (cited on pp. 3, 95).

[3] 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 (cited on pp. 3, 113).

[4] M. Aljehani and M. Inoue. "A swarm of computational clouds as multiple ground control stations of multi-UAV". In: *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*. Oct. 2017, pp. 1–2. DOI: 10.1109/GCCE.2017.8229200 (cited on p. 17).

[5] K. Anderson and K. J. Gaston. "Lightweight unmanned aerial vehicles will revolutionize spatial ecology". In: *Frontiers in Ecology and the Environment* 11.3 (2013), pp. 138–146 (cited on pp. 3, 95).

[6] I. Bayezit and B. Fidan. "Distributed cohesive motion control of flight vehicle formations". In: *IEEE Transactions on Industrial Electronics* 60.12 (2013), pp. 5763–5772 (cited on p. 16).

[7] I. Bekmezci, O. K. Sahingoz, and Şamil Temel. "Flying Ad-Hoc Networks (FANETs): A survey". In: *Ad Hoc Networks* 11.3 (2013), pp. 1254 –1270. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2012.12.004 (cited on p. 15).

[8] Y. Ben-Asher, M. Feldman, S. Feldman, and P. Gurfil. "IFAS: Interactive flexible ad hoc simulator". In: *Simulation Modelling Practice and Theory* 15.7 (2007), pp. 817 –830. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2007.04.004 (cited on p. 11).

[9] Y. Chai and K. c. Cao. "Distributed UAV formation control with two-hop relay protocol". In: *2017 36th Chinese Control Conference (CCC)*. July 2017, pp. 8707–8712. DOI: 10.23919/ChiCC.2017.8028739 (cited on p. 1).

[10] C. Chen, Q. Zhu, and C. Wang. "Rejection Methods for Nakagami-m Fading Simulation". In: *International Conference on Internet Technology and Applications- iTAP'11*. Wuhan, China, 2011. ISBN: 9781424472550 (cited on p. 27).

[11] P. H. Chen and C. Y. Lee. "UAVNet: An Efficient Obstacel Detection Model for UAV with Autonomous Flight". In: *2018 International Conference on Intelligent Autonomous Systems, ICoIAS 2018* (2018), pp. 217–220. DOI: 10.1109/ICoIAS.2018.8494201 (cited on p. 14).

[12] L. Ciarletta, A. Guenard, Y. Presse, V. Galtier, Y. Q. Song, J. C. Ponsart, S. Aberkane, and D. Theilliol. "Simulation and platform tools to develop safe flock of UAVs: a CPS application-driven research". In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. May 2014, pp. 95–102. DOI: 10.1109/ICUAS.2014.6842244 (cited on p. 12).

[13] E. Commission. *Autonomous swarm of heterogeneous RObots for BORDER surveillance*. https://cordis.europa.eu/project/rcn/209949/factsheet/en. Accessed: 2019-01-30 (cited on pp. 3, 95).

[14] K. Dalamagkidis. "Classification of UAVs". In: Jan. 2015, pp. 83–91. ISBN: 978-90-481-9706-4. DOI: 10.1007/978-90-481-9707-1_94 (cited on p. 7).

[15] F. Darema and M. Rotea. "Dynamic Data-driven Applications Systems". In: *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. SC '06. Tampa, Florida: ACM, 2006. ISBN: 0-7695-2700-0. DOI: 10.1145/1188455.1188458 (cited on p. 16).

[16] DJI. *DJI Worldwide leader in Drones/Quadcopters*. https://www.dji.com. Accessed: 2019-04-3 (cited on p. 9).

[17] X. Dong, Y. Zhou, Z. Ren, and Y. Zhong. "Time-varying formation control for unmanned aerial vehicles with switching interaction topologies". In: *Control Engineering Practice* 46 (2016), pp. 26 –36. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2015.10.001 (cited on p. 16).

[18] Dronethusiast. *Drone Flight Simulator – Analysis & Comparison*. http://www.dronethusiast.com/drone-flight-simulator/, accessed 15/06/2017 (cited on p. 11).

[19] B. S. Faiçal, G. Pessin, G. P. R. Filho, A. C.P.L. F. Carvalho, G. Furquim, and J. Ueyama. "Fine-Tuning of UAV Control Rules for Spraying Pesticides on Crop Fields". In: *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*. Nov. 2014, pp. 527–533. DOI: 10.1109/ICTAI.2014.85 (cited on pp. 3, 95).

[20] E. Ferrera, A. Alcántara, J. Capitán, A. Rodríguez Castaño, P. Marrón, and A. Ollero. "Decentralized 3D Collision Avoidance for Multiple UAVs in Outdoor Environments". In: *Sensors* 18 (Nov. 2018), p. 4101. DOI: 10.3390/s18124101 (cited on p. 135).

[21] S. G Gupta, M. Ghonge, and P. Jawandhiya. "Review of Unmanned Aircraft System (UAS)". In: *International Journal of Advanced Research in Computer Engineering & Technology* 9 (Apr. 2013) (cited on p. 7).

[22] R. Garcia and L. Barnes. "Multi-UAV Simulator Utilizing X-Plane". In: *Journal of Intelligent and Robotic Systems* 57.1 (Oct. 2009), p. 393. ISSN: 1573-0409. DOI: 10.1007/s10846-009-9372-4 (cited on p. 11).

[23] *Gauss-Markov mobility*. https://doc.omnetpp.org/inet/api-current/neddoc/inet.mobility.single.GaussMarkovMobility.html. Accessed: 2019-01-30 (cited on p. 86).

[24] L. Ghouti. "Mobility prediction in mobile ad hoc networks using neural learning machines". In: *Simulation Modelling Practice and Theory* 66 (2016), pp. 104 –121. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2016.03.001 (cited on p. 1).

[25] C. Giannini, A. A. Shaaban, C. Buratti, and R. Verdone. "Delay Tolerant Networking for smart city through drones". In: *Proceedings of the Interna-*

tional Symposium on Wireless Communication Systems. Vol. 2016-Octob. Poznan, Poland, 2016, pp. 603–607. ISBN: 9781509020614. DOI: 10.1109/ISWCS.2016.7600975 (cited on p. 1).

[26] N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, and R. Moore. "Google Earth Engine: Planetary-scale geospatial analysis for everyone". In: *Remote Sensing of Environment* 202 (2017), pp. 18 –27. ISSN: 0034-4257 (cited on p. 23).

[27] J. Holt, S. Biaz, L. Yilmaz, and C. A. Aji. "A symbiotic simulation architecture for evaluating UAVs collision avoidance techniques". In: *Journal of Simulation* 8.1 (Feb. 2014), pp. 64–75. ISSN: 1747-7786. DOI: 10.1057/jos.2013.5 (cited on p. 11).

[28] T. C. Hong, K. Kang, K. Lim, and J. Y. Ahn. "Network architecture for control and non-payload communication of UAV". In: *2016 International Conference on Information and Communication Technology Convergence (ICTC)*. Oct. 2016, pp. 762–764. DOI: 10.1109/ICTC.2016.7763289 (cited on p. 19).

[29] A. Y. Javaid, W. Sun, and M. Alam. "UAVSim: A simulation testbed for unmanned aerial vehicle network cyber security analysis". In: *2013 IEEE Globecom Workshops (GC Wkshps)*. Dec. 2013, pp. 1432–1436. DOI: 10.1109/GLOCOMW.2013.6825196 (cited on p. 12).

[30] X. Jinwu, L. Yang, and L. Zhangping. "Flight safety measurements of UAVs in congested airspace". In: *Chinese Journal of Aeronautics* 29 (Aug. 2016). DOI: 10.1016/j.cja.2016.08.017 (cited on p. 13).

[31] E. W. Justh and P. S. Krishnaprasad. *A simple control law for UAV formation flying*. Tech. rep. College Park, Maryland 20740, EE. UU.: Maryland University Institute for Systems Research, 2002 (cited on p. 15).

[32] S. Kang, M. Aldwairi, and K.-I. Kim. "A survey on network simulators in three-dimensional wireless ad hoc and sensor networks". In: *International Journal of Distributed Sensor Networks* 12.9 (2016), p. 1550147716664740. DOI: 10.1177/1550147716664740. eprint: https://doi.org/10.1177/1550147716664740 (cited on p. 11).

[33] B. Kate, J. Waterman, K. Dantu, and M. Welsh. "Simbeeotic: A simulator and testbed for micro-aerial vehicle swarm experiments". In: *Proceedings of the 11th international conference on Information Processing in Sensor*

Networks - IPSN'12. Beijing, China, 2012, pp. 49–60. ISBN: 9781450312271. DOI: 10.1145/2185677.2185685 (cited on p. 12).

[34] M. A. Khan, H. Hasbullah, and B. Nazir. "Recent open source wireless sensor network supporting simulators: A performance comparison". In: *2014 International Conference on Computer, Communications, and Control Technology (I4CT)*. Sept. 2014, pp. 324–328. DOI: 10.1109/I4CT.2014.6914198 (cited on p. 11).

[35] A. Khvilivitzky. *Visual collision avoidance system for unmanned aerial vehicles*. US Patent 5,581,250. Dec. 1996 (cited on p. 13).

[36] B. Kim, K. Kim, B. Roh, and H. Choi. "A new routing protocol for UAV relayed tactical mobile ad hoc networks". In: *2018 Wireless Telecommunications Symposium (WTS)*. Apr. 2018, pp. 1–4. DOI: 10.1109/WTS.2018.8363941 (cited on p. 20).

[37] H. Kim and J. Ben-Othman. "A Collision-Free Surveillance System Using Smart UAVs in Multi Domain IoT". In: *IEEE Communications Letters* 22.12 (2018), pp. 2587–2590. ISSN: 15582558. DOI: 10.1109/LCOMM.2018.2875477 (cited on p. 14).

[38] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar. "Towards a swarm of agile micro quadrotors". In: *Autonomous Robots* 35.4 (Nov. 2013), pp. 287–300. ISSN: 1573-7527. DOI: 10.1007/s10514-013-9349-9 (cited on p. 15).

[39] J. Lee, K. Kim, S. Yoo, A. Y. Chung, J. Y. Lee, S. J. Park, and H. Kim. "Constructing a reliable and fast recoverable network for drones". In: *2016 IEEE International Conference on Communications, ICC 2016*. Kuala Lumpur, Malaysia, 2016, pp. 0–5. ISBN: 9781479966646. DOI: 10.1109/ICC.2016.7511317 (cited on p. 1).

[40] R. L. Lidowski, B. E. Mullins, and R. O. Baldwin. "A novel communications protocol using geographic routing for swarming UAVs performing a Search Mission". In: *2009 IEEE International Conference on Pervasive Computing and Communications*. Mar. 2009, pp. 1–7. DOI: 10.1109/PERCOM.2009.4912764 (cited on p. 15).

[41] Q. Lin, X. Wang, and Y. Wang. "Cooperative Formation and Obstacle Avoidance Algorithm for Multi-UAV System in 3D Environment". In: *Chinese Control Conference, CCC* 2018-July (2018), pp. 6943–6948. ISSN: 21612927. DOI: 10.23919/ChiCC.2018.8483113 (cited on p. 13).

[42] Z. Liu and A. G. Foina. "An autonomous quadrotor avoiding a helicopter in low-altitude flights". In: *IEEE Aerospace and Electronic Systems Magazine* 31.9 (Sept. 2016), pp. 30–39. ISSN: 0885-8985. DOI: 10.1109/MAES.2016.150131 (cited on p. 13).

[43] L. Ma. "Cooperative target tracking using a fleet of UAVs with collision and obstacle avoidance". In: *2018 22nd International Conference on System Theory, Control and Computing, ICSTCC 2018 - Proceedings* (2018), pp. 652–658. DOI: 10.1109/ICSTCC.2018.8540717 (cited on p. 14).

[44] I. Mahjri, A. Dhraief, and A. Belghith. "A Review on Collision Avoidance Systems for Unmanned Aerial Vehicles". In: *5th International Workshop, Nets4Cars/Nets4Trains 2013*. Vol. 7865. Villeneuve d'Ascq, France, 2013. ISBN: 978-3-642-37973-4 (cited on p. 13).

[45] R. R. McCune and G. R. Madey. "Swarm Control of UAVs for Cooperative Hunting with DDDAS". In: *Procedia Computer Science* 18 (2013). 2013 International Conference on Computational Science, pp. 2537 –2544. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2013.05.436 (cited on p. 16).

[46] L. Meier. *Pixhawk*. http://pixhawk.org/. accessed 2019-04-03 (cited on p. 9).

[47] L. Meier and QGroundControl. *MAVLink Micro Air Vehicle Communication Protocol*. http://qgroundcontrol.org/mavlink/start. Accessed: 2019-01-30 (cited on pp. 10, 22).

[48] M.I.C, E.A.S.A, and C.A.A. *Warsaw Declaration: Drones as a leverage for jobs and new business opportunities*. https://ec.europa.eu/transport/sites/transport/files/drones-warsaw-declaration.pdf, accessed 4/3/2019. Nov. 2016 (cited on pp. 2, 61).

[49] J. Modares, N. Mastronarde, and K. Dantu. "UB-ANC Emulator: An Emulation Framework for Multi-Agent Drone Networks". In: *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. San Francisco, USA, 2016, pp. 252–258. ISBN: 8750141007 (cited on p. 12).

[50] N. Mohamed, J. Al-Jaroodi, I. Jawhar, A. Idries, and F. Mohammed. "Unmanned aerial vehicles applications in future smart cities". In: *Technolog-*

ical Forecasting and Social Change (2018). ISSN: 0040-1625. DOI: https://doi.org/10.1016/j.techfore.2018.05.004 (cited on pp. 2, 61).

[51] F. Mohammed, A. Idries, N. Mohamed, J. Al-Jaroodi, and I. Jawhar. "UAVs for smart cities: Opportunities and challenges". In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. May 2014, pp. 267–273 (cited on pp. 3, 61).

[52] *NS-2 The Network Simulator*. http://nsnam.sourceforge.net/wiki/index.php/Main_Page. Accessed: 2019-01-30 (cited on p. 23).

[53] *OMNeT++ Discrete Event Simulator*. https://omnetpp.org/. Accessed: 2019-01-30 (cited on pp. 23, 86).

[54] R. C. Palat, A. Annamalau, and J. R. Reed. "Cooperative relaying for ad-hoc ground networks using swarm UAVs". In: *MILCOM 2005 - 2005 IEEE Military Communications Conference*. Oct. 2005, 1588–1594 Vol. 3. DOI: 10.1109/MILCOM.2005.1605902 (cited on p. 16).

[55] S. Park, H. Kim, K. Kim, and H. Kim. "Drone Formation Algorithm on 3D Space for a Drone-based Network Infrastructure". In: *IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. Valencia, Spain, 2016. ISBN: 9781509032549 (cited on p. 1).

[56] R. Purta, M. Dobski, A. Jaworski, and G. Madey. "A Testbed for Investigating the UAV Swarm Command and Control Problem Using DDDAS". In: *Procedia Computer Science* 18 (2013). 2013 International Conference on Computational Science, pp. 2018 –2027. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2013.05.371 (cited on p. 16).

[57] *Quaternium, home of the longest flight time hybrid drone*. http://www.quaternium.com/. Accessed: 2019-01-30 (cited on pp. 84, 114).

[58] R. Rahimi, F. Abdollahi, and K. Naqshi. "Time-varying formation control of a collaborative heterogeneous multi agent system". In: *Robotics and Autonomous Systems* 62.12 (2014), pp. 1799 –1805. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2014.07.005 (cited on p. 16).

[59] A. Ray and D. De. "An energy efficient sensor movement approach using multi-parameter reverse glowworm swarm optimization algorithm in mobile wireless sensor network". In: *Simulation Modelling Practice and Theory* 62

(2016), pp. 117 –136. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2016.01.007 (cited on p. 1).

[60] *SESAR Joint Undertaking*. https://www.sesarju.eu/, accessed 4/3/2019 (cited on pp. 3, 61).

[61] A. Singh. *Drone Collision Avoidance*. https://create.arduino.cc/projecthub/anshulsingh163/drone-collision-avoidance-system-0b6002, accessed 11/03/2019 (cited on p. 13).

[62] A. Sivakumar and C. K.-Y. Tan. "UAV Swarm Coordination Using Cooperative Control for Establishing a Wireless Communications Backbone". In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 3 - Volume 3*. AAMAS '10. Toronto, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 1157–1164. ISBN: 0-98265-713-7 (cited on p. 15).

[63] B. J. O. de Souza and M. Endler. "Coordinating movement within swarms of UAVs through mobile networks". In: *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. Mar. 2015, pp. 154–159. DOI: 10.1109/PERCOMW.2015.7134011 (cited on p. 15).

[64] A. D. Team. *SITL Simulator (Software in the Loop)*. http://ardupilot.org/ardupilot/. accessed 2019-04-03 (cited on p. 9).

[65] A. Viguria, I. Maza, and A. Ollero. "Distributed Service-Based Cooperation in Aerial/Ground Robot Teams Applied to Fire Detection and Extinguishing Missions". In: *Advanced Robotics* 24.1-2 (2010), pp. 1–23. DOI: 10.1163/016918609X12585524300339 (cited on pp. 3, 113).

[66] P. Vincent and I. 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 (cited on pp. 3, 113).

[67] M. Wang, H. Voos, and D. Su. "Robust Online Obstacle Detection and Tracking for Collision-Free Navigation of Multirotor UAVs in Complex Environments". In: *2018 15th International Conference on Control, Automation, Robotics and Vision, ICARCV 2018* (2018), pp. 1228–1234. DOI: 10.1109/ICARCV.2018.8581330 (cited on p. 14).

[68] A. Watts, V. Ambrosia, and E. A. Hinkley. "Unmanned Aircraft Systems in Remote Sensing and Scientific Research: Classification and Considerations of Use". In: *RS* 4 (Dec. 2012), pp. 1671–1692. DOI: 10.3390/rs4061671 (cited on p. 7).

[69] Z. Zhao and T. Braun. "Topology Control and Mobility Strategy for UAV Ad-hoc Networks: A Survey". In: *Joint ERCIM eMobility and MobiSense Workshop*. Citeseer. 2012, pp. 27–32 (cited on p. 15).

[70] X. Zhou, X. Yu, and X. Peng. "UAV Collision Avoidance Based on Varying Cells Strategy". In: *IEEE Transactions on Aerospace and Electronic Systems* PP.c (2018), p. 1. ISSN: 15579603. DOI: 10.1109/TAES.2018.2875556 (cited on p. 14).