Computers and Electrical Engineering 74 (2019) 413–428

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Automatic system supporting multicopter swarms with manual guidance☆

Francisco Fabra[a], Willian Zamora[a,b,*], Joan Masanet[a], Carlos T. Calafate[a,*], Juan-Carlos Cano[a], Pietro Manzoni[a]

[a] Departament of Computer Engineering, Universitat Politècnica de València, Valencia, 46022 Spain
[b] Facultad Ciencias Informáticas, Universidad Laica Eloy Alfaro de Manabí, Manta, 130802 Ecuador

ARTICLE INFO

Article history:
Received 20 September 2018
Revised 16 January 2019
Accepted 20 January 2019
Available online 20 February 2019

Keywords:
UAV
Swarm
Multicopter
Flight coordination
ArduSim

ABSTRACT

Currently, there are some 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. In this paper we develop a 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. We validate our solution using a realistic simulation software that we developed (ArduSim), testing flights with multiple numbers of UAVs and different swarm configurations. 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.

1. Introduction

Currently, the adoption of UAVs has been steadily increasing, being these devices used for many different tasks. In fact, they are able to optimize some tasks that previously relied on other approaches (e.g. aerial recordings, where helicopters where replaced by camera-equipped UAVs), and also to perform new tasks (e.g. automated sensing). In general, they are acknowledged as being a very useful tool in a wide range of real scenarios, and their set of features keeps increasing at a steady pace, meaning that new applications can be envisioned for these devices, many more being expected in the future.

In this sense, the use of more than one air vehicle opens up new possibilities, especially when these aircrafts are used as a whole, a solution known as a "swarm" of UAVs. So, the applicability of these swarms expands their capabilities and utilities in a scalable manner, while introducing a low cost solution compared to their potential. Among the new use-cases for UAV swarms we have: applications for large-scale agriculture in search of pests or weeds [1,2], search for missing persons in wide areas and/or with poor accessibility [3,4], prevention and control of forest fires [5], border surveillance, and acting as relays for Internet distribution, among others. Additionally, using swarms allows accelerating sensing tasks within a target area by enabling each UAV to load a different sensor, while they all act on the same area simultaneously [6].

☆ This paper is for CAEE special section SI-fadn. Reviews processed and recommended for publication to the Editor-in-Chief by Guest Editor Dr. C. A. Kerrache.

* Corresponding authors.
E-mail addresses: frafabco@cam.upv.es (F. Fabra), wilzame@posgrado.upv.es (W. Zamora), joanml93@gmail.com (J. Masanet), calafate@disca.upv.es (C.T. Calafate), jucano@disca.upv.es (J.-C. Cano), pmanzoni@disca.upv.es (P. Manzoni).

https://doi.org/10.1016/j.compeleceng.2019.01.026
0045-7906/

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. In particular, this paper 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, fire tracking, 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. In addition, our focus on UAV swarms 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 paper, we propose the FollowMe protocol to define and maintain the 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. Our protocol allows three types of swarm layouts, such as configurations following a line, a matrix, or a circle around the leader. Our solution has been validated using ArduSim [6], a realistic simulation software that we developed, testing different numbers of UAVs in the swarm, different distances between UAVs, different message update intervals, and the three aforementioned swarm formations.

Experimental results under different conditions show that the proposed protocol is able to adequately maintain the swarm formation and dynamically respond to the pilot's commands, with a lag of only a few seconds in the worst case, avoiding collision between UAVs as long as some minimal distances towards neighboring UAVs are maintained.

The rest of this paper is organized as follows: Section 2 presents some related papers on this topic. In Section 3 we introduce ArduSim, the simulation tool used to implement and test the proposed protocol. Then, in Section 4, we detail the proposed FollowMe protocol. The data source and methodology used to measure the performance and precision of the FollowMe protocol are described in Section 5. A comprehensive validation of the protocol is included in Section 6. Finally, in Section 7, we present our conclusions and refer to future work.

2. Related works

Currently, the reduction of costs, along with different technological advances, have made drones accessible as a work tool in our society. In the beginning, drones had mainly military uses, but over the years their adoption has become widespread, especially among the research community. Below we describe some related works focusing on the use of drones for the creation of UAV swarms.

In [7] authors present a topology that adopts ad-Hoc networks as mechanisms that could be applied to controlling the mobility of UAV swarms. Its main features are focused on connectivity and coverage area. Later, Bekmezci et al. [8] introduced extensive studies on the use of Flying Ad-Hoc Networks (FANETs), describing the main problems of deploying ad-hoc networks based on UAVs. They describe features such as topology changes, radio propagation model, adaptability, scalability, latency, UAV platform constraints, and bandwidth.

In [9,10] the authors propose to control swarms using the Dynamic Data-Driven Application System (DDDAS) [11]. These techniques are used in [9] 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 [10], the authors use these frameworks to test different scenarios, and determine if a region can be cleaned or not using swarms.

In [12] the authors propose 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 given by a sensor network installed in a test laboratory. It also presents details about the algorithm and the implementation of the hardware used.

Palat et al. [13] 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 implemented 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 [14] authors propose using swarm clouds as multiple Ground Control Stations (GCS). 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

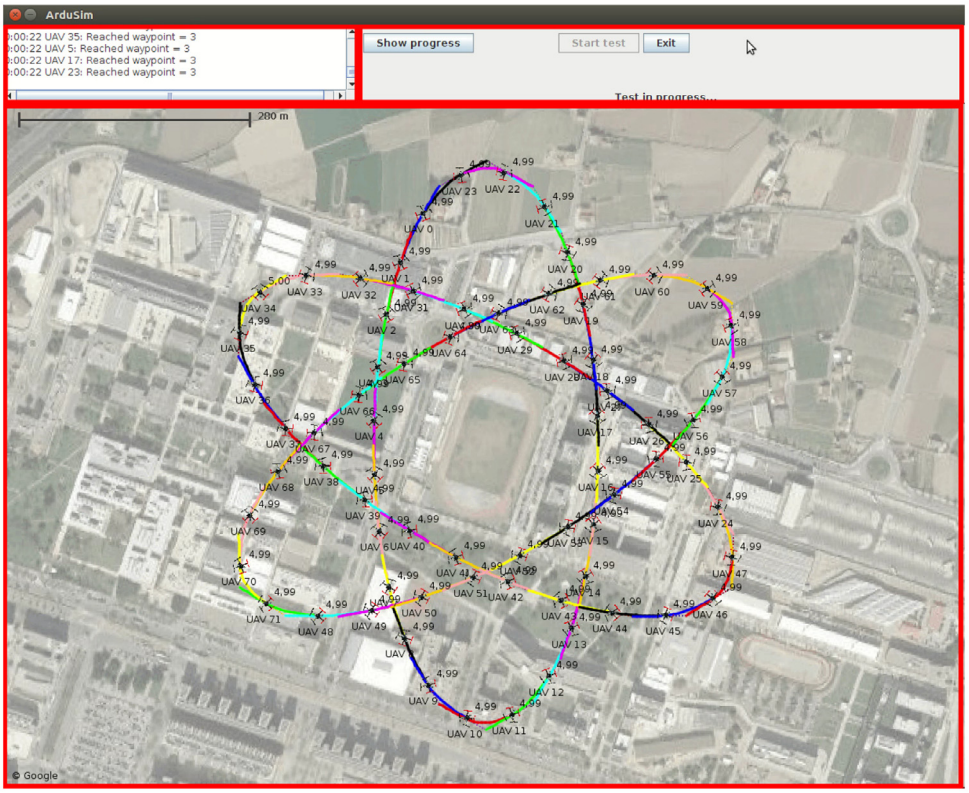


Fig. 1. ArduSim main window: 72 multicopters performing elliptical missions.

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 aircraft.

In [15] the authors propose a New Structure Particle Swarm Optimization (nPSO) algorithm that is applied successfully to achieve optimal separation between UAVs in the presence of obstacles. The algorithm used adjusts the standard PSO algorithm in such a way that the UAVs tend to converge towards the global optimum quickly. The experimental results were made by simulation using Matlab. This solution presents two problems: (i) the simulation does not use a real simulation system like our Ardusim [16], and (ii) it assumes the existence of a base station for communication where the best location for the UAV swarms is calculated.

In [17] the authors analyze the problem of area coverage by a swarm of UAVs in a military context. Authors exploit the deterministic properties of chaotic dynamics to produce mobility models that allow monitoring the behavior of the swarm from a GCS while generating UAV movements that are unpredictable for any external observer. The results obtained improve the response of the system compared to previously proposed theoretical models.

Our research work fills-in a gap by focusing on applications where drones must be manually guided, instead of following a planned mission. This is applicable to situations where the drone pilot must respond to visual stimuli in real time, and in such a way that the other drones in the swarm that follow the leading drone have to adapt to these unexpected movements dynamically. So, our proposed protocol is able to define and maintain the formation of unmanned UAVs in a swarm managed by a real pilot, who remotely controls the swarm leader in real time, while the other UAVs constantly adapt their trajectories to follow it. We use our own Ardusim [16] simulator, and validate our proposal with three types of swarm layouts - linear, matrix and circular - under different conditions.

3. ArduSim simulator: an overview

The FollowMe protocol has been developed in ArduSim, a novel multi-UAV flight simulator developed by Fabra et al. [6], and available online [16] under the Apache License 2.0. ArduSim is able to simulate the physics of up to 256 multicopters with great accuracy, and it also simulates the communication among them through virtual Wi-Fi links.

Fig. 1 shows the main window of ArduSim. On the upper left corner (1), the log of the application provides feedback about the progress of the protocol and the simulation. On the upper right corner (2), several buttons give control over the simulation. Finally, most of the window (3) is used to show the movement of the virtual multicopters. In this case, the 72

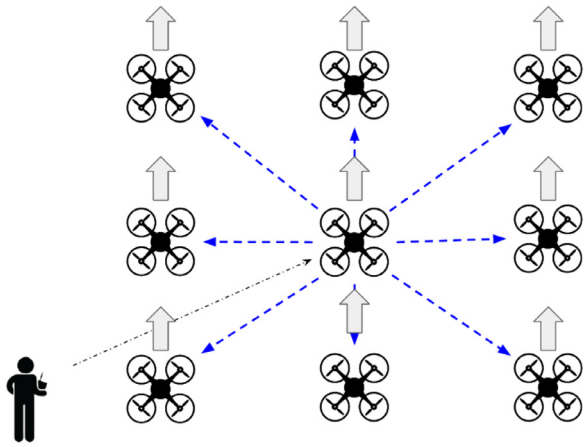


Fig. 2. FollowMe protocol operation using a matrix formation.

UAVs shown are performing an elliptical planned mission. Lines are drawn to show the path already followed by the UAVs, using different colors to make it easy to identify the route of each particular UAV.

The virtual multicopters have the firmware of a real flight controller, which enables to communicate with them using MAVLink [18], a *de facto* standard used by current open source flight controllers. This way, any protocol implemented in ArduSim can be ported to real multicopters seamlessly, if the developer follows the recommendations included with the simulator.

The most relevant characteristics of ArduSim are:

- Easy protocol deployment on real UAVs.
- Soft real-time simulation, which speeds up the protocol testing.
- High scalability and stability, enabling to simulate up to 100 virtual UAVs in near real-time, or up to 225 UAVs if hard real-time is not required.
- Realistic communication simulation, based on experiments with real multicopters, and including a model of the wireless channel occupancy.
- Easy control of multicopters with a comprehensive API.
- PC Companion. ArduSim can be executed (i) as a simulator, (ii) in a real multicopter to deploy a protocol, and also (iii) as a PC Companion in order to control the deployment of a protocol on real multicopters.
- Collision detection. ArduSim detects when two virtual UAVs collide, so that the developer can stop a test if the protocol is failing.
- Comprehensive data logging. The parameters of each experiment are stored so that it can be repeated later on if necessary. The path followed by the multicopters is also stored for later analysis, and can be exported to file formats compatible with the OMNeT++ [19] and NS2 simulators [20].

Using our proposed simulation platform, in this paper we develop a new protocol to coordinate the flight of a swarm following a multicopter manually controlled with a remote. In order to use ArduSim, we needed to extend its capabilities with a simulated remote controller.

A custom hexacopter was equipped with a Raspberry Pi 3B+, connected to the flight controller, and a program was written in Java to capture the commands received by the flight controller from the pilot's remote control. The value of the channels used for throttle, yaw, pitch, and roll were recorded to later on replicate a real trace of the path followed by the UAV. Finally, this trace was used as an input to ArduSim to control the leader multicopter in the scope of the FollowMe protocol, which will be detailed in the next section.

4. Overview of the FollowMe protocol

In this section we detail the implementation of the FollowMe protocol, which was designed to allow a swarm of multicopters to follow the leader when the latter is manually guided by a pilot using a standard remote control.

4.1. Architecture

The proposed protocol uses the master-slave model, where a single UAV is manually controlled by a pilot (master), and several UAVs follow the former automatically (slaves). Fig. 2 shows the scheme of the FollowMe protocol operation using a matrix formation.

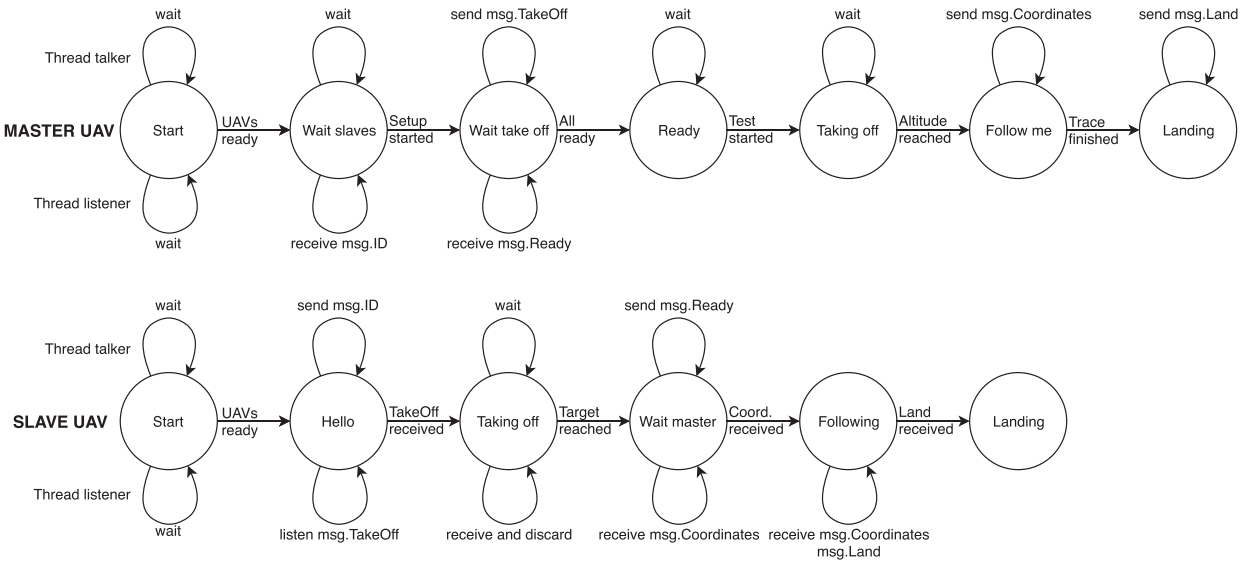


Fig. 3. FollowMe protocol finite state machine.

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 [21]. 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.

4.2. Finite state machine

Fig. 3 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*). 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 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 an 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.

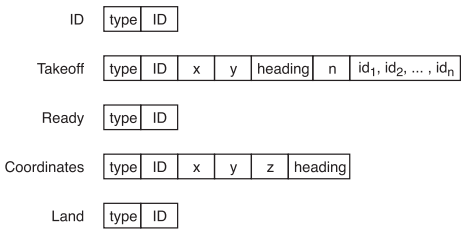


Fig. 4. Follow Me message types.

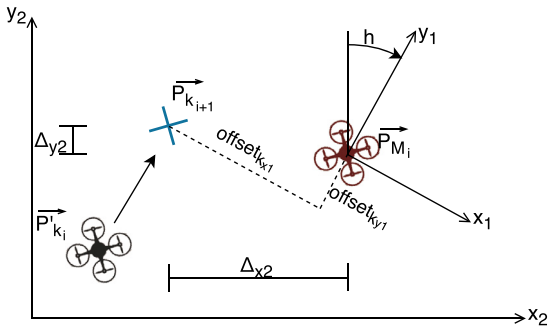


Fig. 5. FollowMe protocol: target location calculation.

4.3. Messages

Several messages are needed to coordinate the behavior of the master and the slave multicopters. Fig. 4 shows the format of these messages. Fields *type* and *ID* 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 to the protocol 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 Fig. 5) 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.

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

where

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

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

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

On each iteration i, the offset of a slave k with respect to the master UAV 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 to the master (*offset*), 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, so a slight delay between the master and its slaves is expected.

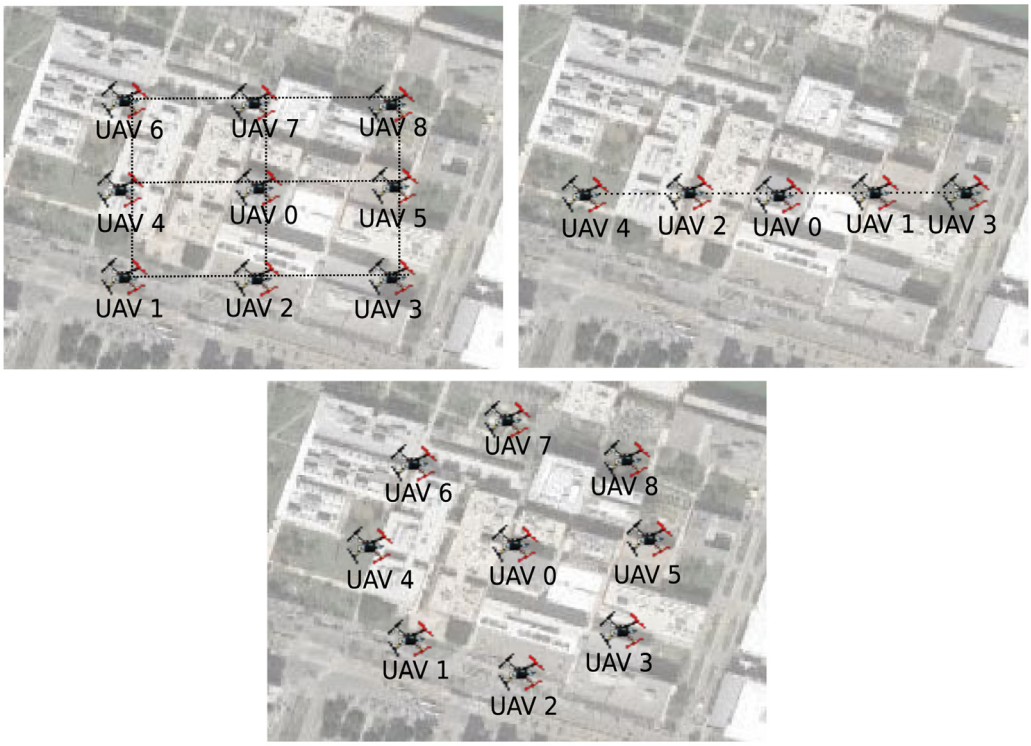


Fig. 6. Different swarm layouts tested: i) matrix with 9 UAVs, ii) linear with 5 UAVs, and iii) circular with 9 UAVs.

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.

4.4. Swarm formation

Three different swarm patterns have been implemented in the protocol for testing purposes:

- **Linear.** The UAVs are arranged by forming a straight line, perpendicular to the heading of the master, and with a fixed distance between contiguous multicopters.
- **Matrix.** The UAVs are ordered in a square matrix with the same distance between contiguous multicopters.
- **Circular.** The slaves are surrounding the master forming a circle, and with a defined minimum distance between them, and to the master.

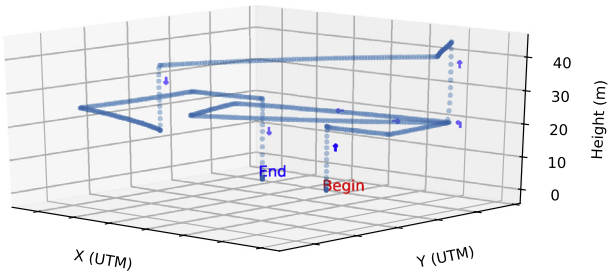
Notice that the main parameter used on each formation is the distance between multicopters, as it affects the probability of collision during flight.

Fig. 6 shows the three different formations created. In general, the master drone (UAV #0) is always located in the central position of the formation to optimize performance.

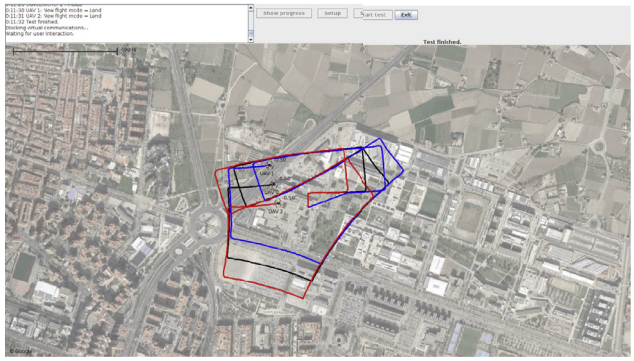
Notice that the chosen patterns are quite representative, as they provide different trade-offs. For a same number of UAVs in a swarm, and for a same minimum separation distance between UAVs, the linear scenario can be considered the worst-case situation as the average distance between the leader UAV the other UAVs in the swarm is the highest. In particular, UAVs in edge positions will be quite far away from the leader, meaning that messages coming from the leader are prone to be lost more often, making them suffer from coordination problems. On the contrary, the matrix pattern having the leader UAV located at a central position represents the opposite case, being that pattern the one that allows a more compact pattern to be generated, minimizing the distances between the leader UAV and the remaining UAVs, and thus minimizing distance-related losses. Finally, the circular pattern, having the leader UAV in the center, represents a situation in-between the previous two cases regarding distance from the leader to the rest of UAVs, with the particularity that this distance is the same for all UAVs, meaning that they are expected to experience similar message loss levels.

5. 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



(a) Real time data source 3D.



(b) Real time data source using Ardusim.

Fig. 7. Real-time data source.

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.

5.1. 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, an 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. The flight tests were made on the premises of our institution. Fig. 7(a) shows the trajectory of the data source used in the simulation, and Fig. 7(b) shows the actual flight trajectory in a real map.

As complementary information, to gain further insight into the flight pattern generated, Fig. 8 shows how different variables have evolved throughout time. It is noteworthy that, in Fig. 8 (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. Fig. 8 (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 in the simulation, and afterward we proceed to perform the interpolation for the simulated dataset.

5.2. 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). Fig. 9 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.

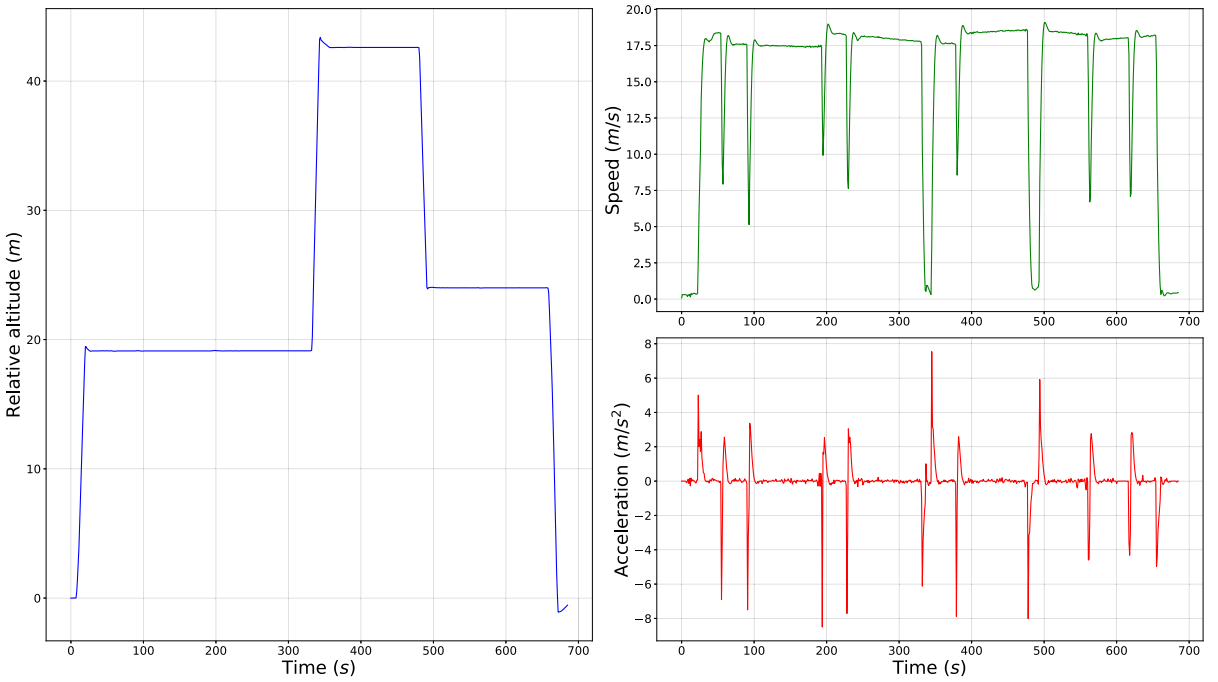


Fig. 8. Data source variables.

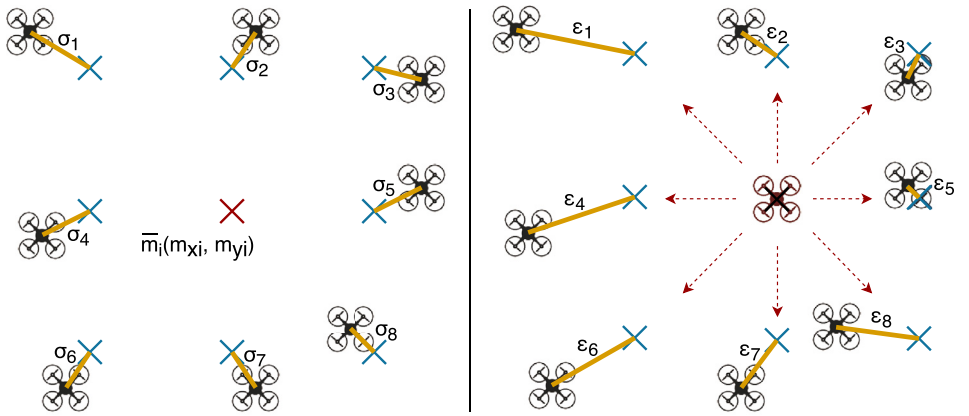


Fig. 9. FollowMe protocol: Formation stability error (left), and global error (right).

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.

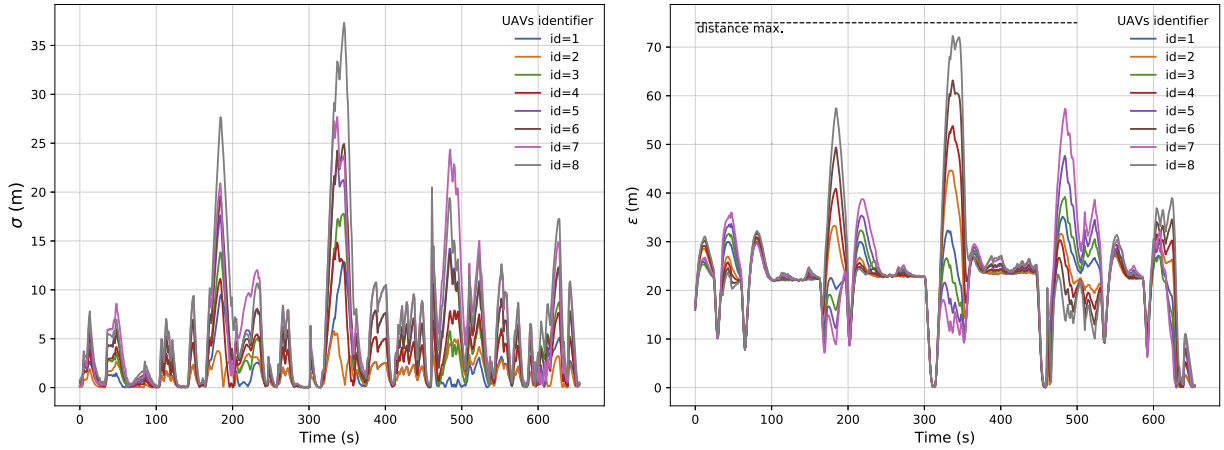
$$\overline{m_i} = (\overline{m_{x_i}}, \overline{m_{y_i}}), \text{ where} \begin{cases} \overline{m_{x_i}} = \frac{\sum_{k=1}^{n} x_k}{n} \\ \overline{m_{y_i}} = \frac{\sum_{k=1}^{n} y_k}{n} \end{cases} \tag{2}$$

In Eq. (2), i represents the current time step, k represents the different slave UAVs, and n is the total number of slave UAVs.

The theoretical position of a slave UAV $\overrightarrow{T_{k_{i+1}}}$ calculated for time instant $i+1$ is obtained at instant i, as shown in Fig. 5, using Eq. (1):

$$\overrightarrow{T_{k_{i+1}}} = (x_{k_{i+1}}, y_{k_{i+1}}), \text{ where} \begin{cases} x_{k_{i+1}} = \overline{m_{x_i}} + \Delta_{x2} \\ y_{k_{i+1}} = \overline{m_{y_i}} + \Delta_{y2} \end{cases} \tag{3}$$

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



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

Fig. 10. Error on a swarm of 9 UAVs using a linear formation, and with a separation between neighbors of 75m.

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

$$\sigma_{k_i} = \sqrt{\left(x_{k_{i+1}} - x'_{k_i}\right)^2 + \left(y_{k_{i+1}} - x'_{k_i}\right)^2} \tag{4}$$

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 $\overrightarrow{R_{M_i}}$ in Eq. (3), using the master heading to calculate the theoretical formation with Eq. (1). Finally, ε is calculated the same way as σ through Eq. (4).

6. Validation of the proposed protocol

Once the data source and the methodology used to calculate the errors were described, 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 comparing the different proposed formations. Below we provide details about the results obtained.

6.1. Swarm with linear formation

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

6.1.1. 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 m. Several simulations were made, and the average of the whole set of all the tests was taken. Fig. 10 shows both the stability error in the swarm formation (a), and the global error (b), for each of the slaves of the formation. (a) 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 m.

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 Fig. 10(a) and (b) show that the UAV IDs that are the furthest away the master UAV tend to experience higher errors. For this reason, we decided to evaluate the behavior of each of the slave drones in more detail, as shown below.

Fig. 11 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. Specifically, slave drones number 7 and 8 present a more significant error from those that are closer to the master drone. The reason why the errors experienced by UAVs increase as we move away from the master is that the slave UAVs experience a considerable delay time with respect the master, as the latter reacts immediately to the pilot's commands, while all remaining UAVs in the swarm following experience delays of different magnitude.

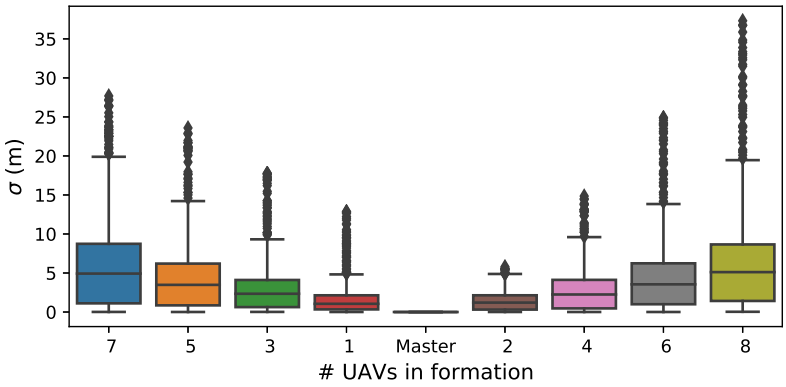
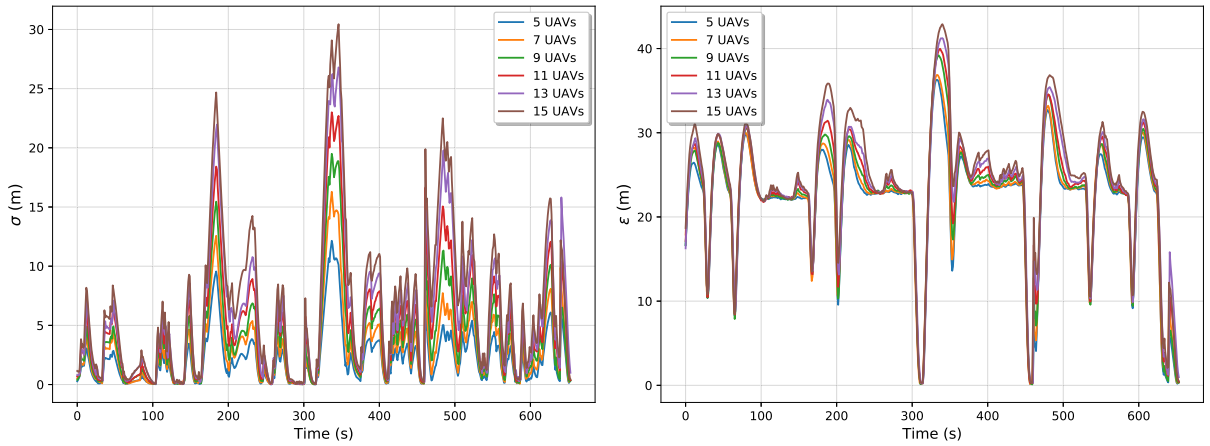


Fig. 11. Box and whisker plot of the formation error (σ) of the 9 UAVs of the swarm.



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

Fig. 12. Mean error for all the UAVs in a swarm of n drones using a linear formation, and a neighbor separation of 75 m.

6.1.2. 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. Fig. 12 shows the formation and global errors for these numbers of UAVs. Fig. 12(a) evidences that, as the number of UAVs increases, higher the formation error tends to grow. Fig. 12(b) shows that most of the route has an average error of 22.84 m, although higher values (above 40 m) can take place in simulations with 13, and 15 UAVs.

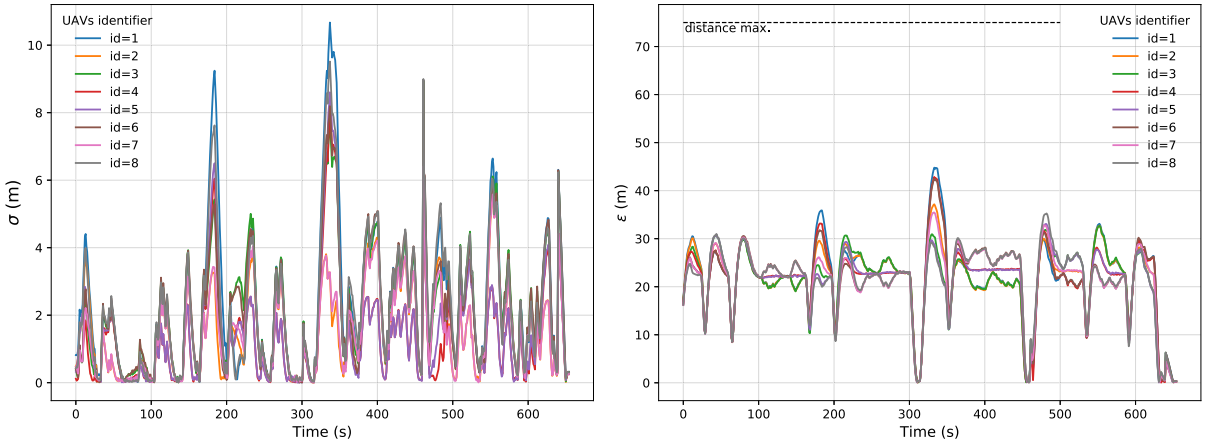
Regarding the evaluations discussed above, we find that our developed protocol is capable of achieving the desired flight pattern. In terms of swarm formation errors, they are due to the following factors: i) Slave drones calculate their target position accounting for the current location of the master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

6.2. 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.

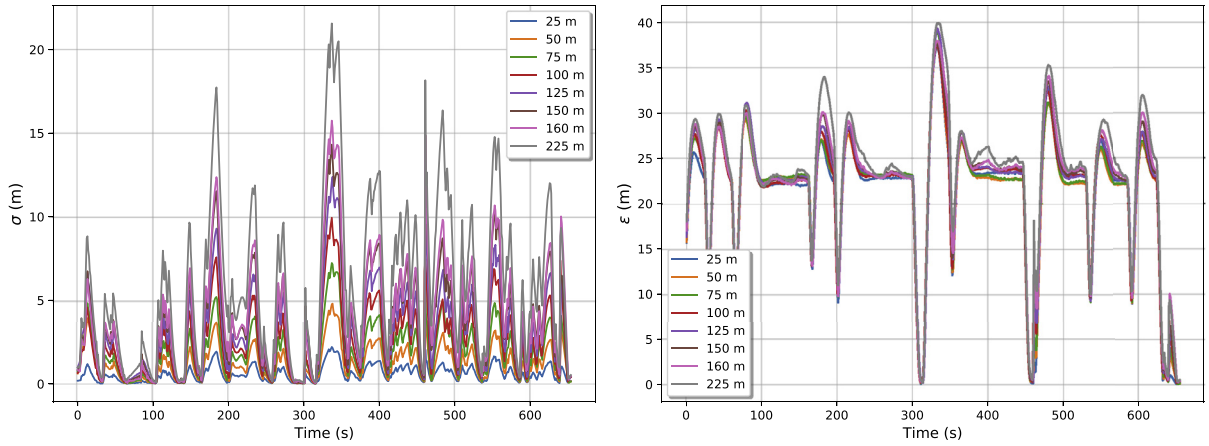
6.2.1. Formation evaluation using nine UAVs

We used the same configuration described in the linear formation, but changing the swarm layout to a matrix formation. Fig. 13 shows the results regarding 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, Fig. 13(a) shows that the



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

Fig. 13. Error on a swarm of 9 UAVs using a matrix formation, and for an inter-UAV separation of 75 m.



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

Fig. 14. Error on a swarm of 9 drones using a matrix formation, varying the distance of separation between drones.

errors are less than 10 m for most part of the route traveled in the simulation. Specifically, the average error obtained with this formation is 1.83 m. Fig. 13(b) shows that the global distance error is far from the distance between UAVs we have set, having an average of 21.67 m throughout the whole path.

6.2.2. Evaluating swarm cohesion when varying the inter-UAV distances

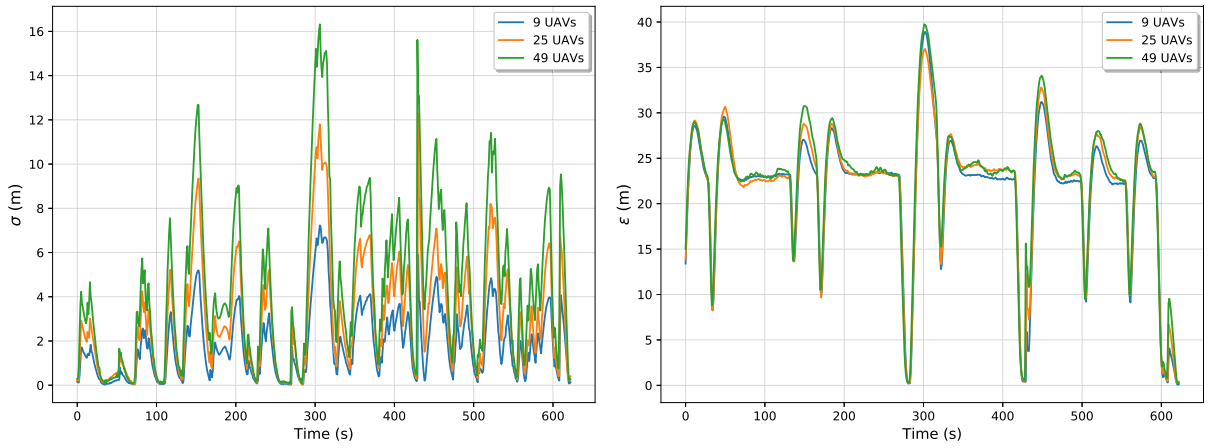
After the previous analysis, we proceeded to evaluate the error by varying the distance between UAVs, and using the same configuration of 9 drones. In this test, we used several separation distances: 25, 50, 75, 100, 125, 150, 160, and 225 m. We consider the maximum distance of the master drone equivalent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

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

6.2.3. Formation evaluation using different number of UAVs

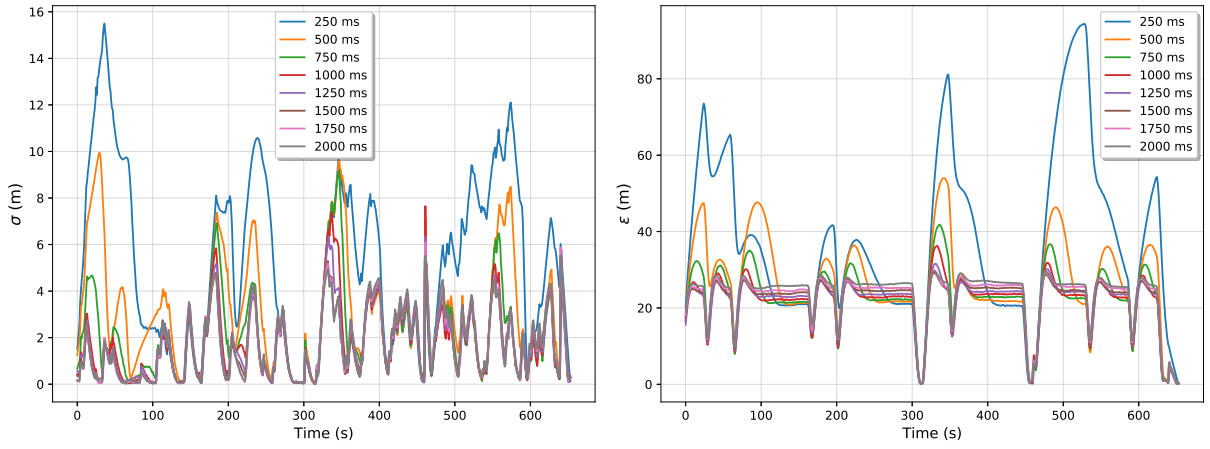
In the previous results, it has been proven that, for larger inter-UAV distances, the formation error grows, being the matrix swarm more unstable. Starting from this point, we now want to check the swarm performing for a matrix layout when having 9, 25, and 49 UAVs, while maintaining the same distance between them (75 m).

In Fig. 15(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 Fig. 15(b) the variability detected can be considered small, and the average error found is 21.58 m.



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

Fig. 15. Mean error for all the UAVs in a swarm of n drones using a matrix formation and an inter-UAV distance of 75 m.



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

Fig. 16. Mean error for all the UAVs in a swarm of 9 drones using a matrix formation, varying the network refresh period.

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.

6.3. Formation error when varying the position refresh period

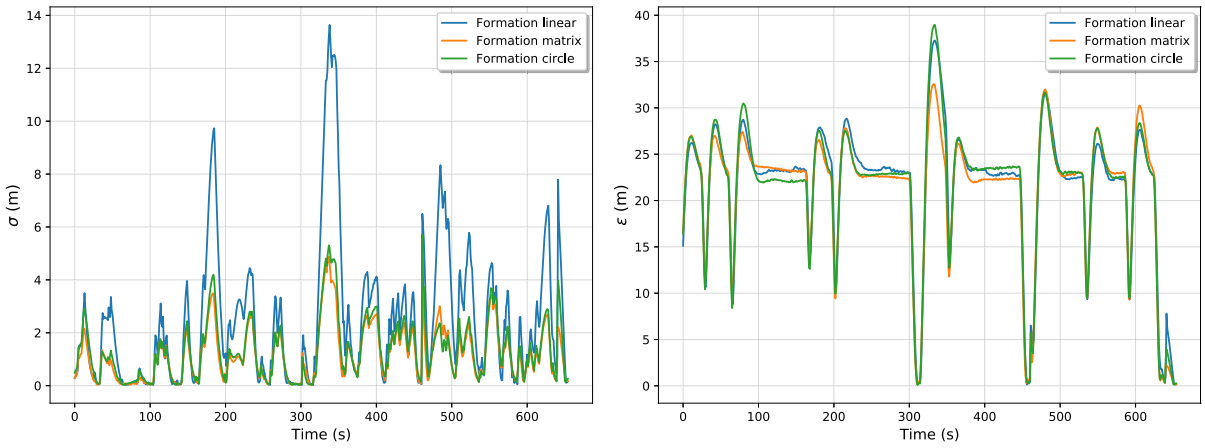
In this part, we want to evaluate how our protocol behaves if we vary the position update period in terms of communications. For this analysis we used the matrix formation. Different simulations were performed using nine UAVs for a distance between them of 75 m. The refresh time values evaluated were: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

Fig. 16 shows the results obtained. In general we observe that, the shorter the update period, the higher the evaluation error becomes. In particular, Figure 16(a) shows that, with a refresh period of 1000 ms, the performance is optimal. Figure 16(b) shows that, when using update values between 1000 ms and 1500 ms, the error tends to decrease in those peaks that correspond to situations where the UAVs slow down.

Table 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,

Table 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



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

Fig. 17. Mean error for all the UAVs in a swarm of 9 drones at a separation distance of 50 m, varying the formation type.

Table 2
Errors values using different refresh periods.

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
Circle	1.34	1.77	0.31	21.67	23.30	1.14

having a maximum general error value of 23.61, and a mean value of 21.67. Overall, we find that 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.

6.4. Formation error for the three available 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 m. Thus, the parameters established for the collision system were: the distance threshold was set to 10 m, and the altitude threshold to 20 m.

Fig. 17(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 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.

7. Conclusions and future work

In this paper 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 an 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. As future work we plan to reduce the swarm formation error by introducing predictions in the future UAV position estimations.

Acknowledgment

This work was partially supported by the “Programa Estatal de Investigación, Desarrollo e Innovación Orientada a Retos de la Sociedad, Proyecto TEC2014-52690-R”, Spain, the “Universidad Laica Eloy Alfaro de Manabí,” and the “Programa de Becas SENESCYT de la República del Ecuador.”

References

[1] FaiÃğal BS, Pessin G, Filho GPR, Carvalho ACPLF, Furquim G, Ueyama J. Fine-tuning of uav control rules for spraying pesticides on crop fields. In: 2014 IEEE 26th international conference on tools with artificial intelligence; 2014. p. 527–33. doi:10.1109/ICTAI.2014.85.
[2] Albani D, IJsselmuiden J, Haken R, Trianni V. Monitoring and mapping with robot swarms for agricultural applications. In: 2017 14th IEEE international conference on advanced video and signal based surveillance (AVSS); 2017. p. 1–6. doi:10.1109/AVSS.2017.8078478.
[3] Vincent P, Rubin I. A framework and analysis for cooperative search using uav swarms. In: Proceedings of the 2004 ACM symposium on applied computing, SAC'04. New York, NY, USA: ACM; 2004. p. 79–86. ISBN 1-58113-812-1. doi:10.1145/967900.967919.
[4] Aljehani M, Inoue M. Multi-uav tracking and scanning systems in m2m communication for disaster response. In: 2016 IEEE 5th global conference on consumer electronics; 2016. p. 1–2. doi:10.1109/GCCE.2016.7800524.
[5] Viguria A, Maza I, Ollero A. Distributed service-based cooperation in aerial/ground robot teams applied to fire detection and extinguishing missions. Adv Rob 2010;24(1–2):1–23. doi:10.1163/016918609X12585524300339.
[6] Fabra F, T Calafate C, Cano J-C, Manzoni P. Ardusim: accurate and real-time multicopter simulation. Simul Modell Pract Theory 2018;87(1):170–90. doi:10.1016/j.simpat.2018.06.009.
[7] Zhao Z, Braun T. Topology control and mobility strategy for uav ad-hoc networks: A surveyâŃĘ. In: Joint ERCIM eMobility and MobiSense workshop. Citeseer; 2012. p. 27–32.
[8] İlker Bekmezci, Sahingoz OK, Şamil Temel. Flying ad-hoc networks (fanets): a survey. Ad Hoc Netw 2013;11(3):1254–70. doi:10.1016/j.adhoc.2012.12.004.
[9] Purta R, Dobski M, Jaworski A, Madey G. A testbed for investigating the uav swarm command and control problem using dddas. Procedia Comput Sci 2013;18:2018–27 2013 international conference on computational science. doi:10.1016/j.procs.2013.05.371.
[10] McCune RR, Madey GR. Swarm control of uavs for cooperative hunting with dddas. Procedia Comput Sci 2013;18:2537–44 2013 international conference on computational science. doi:10.1016/j.procs.2013.05.436.
[11] Darema F, Rotea M. Dynamic data-driven applications systems. In: Proceedings of the 2006 ACM/IEEE conference on supercomputing, SC'06. New York, NY, USA: ACM; 2006 ISBN 0-7695-2700-0. doi:10.1145/1188455.1188458.
[12] Kushleyev A, Mellinger D, Powers C, Kumar V. Towards a swarm of agile micro quadrotors. Auton Robots 2013;35(4):287–300. doi:10.1007/s10514-013-9349-9.
[13] Palat RC, Annamalau A, Reed JR. Cooperative relaying for ad-hoc ground networks using swarm uavs. In: MILCOM 2005–2005 IEEE military communications conference; 2005. p. 1588–1594 Vol. 3. doi:10.1109/MILCOM.2005.1605902.
[14] Aljehani M, Inoue M. A swarm of computational clouds as multiple ground control stations of multi-uav. In: 2017 IEEE 6th global conference on consumer electronics (GCCE); 2017. p. 1–2. doi:10.1109/GCCE.2017.8229200.
[15] Spanogianopoulos S, Zhang Q, Spurgeon S. Fast formation of swarm of uavs in congested urban environment. IFAC-PapersOnLine 2017;50(1):8031–6 20th IFAC world congress; doi:10.1016/j.ifacol.2017.08.1228.
[16] Ardusim. Accurate and real-time multi-UAV simulation. https://bitbucket.org/frafabco/ardusim/src/master/; Accessed: 2018-09-05.
[17] Rosalie M, Danoy G, Chaumette S, Bouvry P. Chaos-enhanced mobility models for multilevel swarms of uavs. Swarm Evol Comput 2018;41:36–48. doi:10.1016/j.swevo.2018.01.002.
[18] Meier L. QGroundControl, MAVLink micro air vehicle communication protocol. http://qgroundcontrol.org/mavlink/start; Accessed: 2018/09/05.
[19] OMNeT++ discrete event simulator. https://omnetpp.org/; Accessed: 2018-09-10.
[20] NS-2 the network simulator. http://nsnam.sourceforge.net/wiki/index.php/Main_Page; Accessed: 2018-09-10.
[21] Quaternium, home of the longest flight time hybrid drone. http://www.quaternium.com/; Accessed: 2018-09-10.

Francisco Fabra is a Ph.D. student in the “Networking Research Group (GRC)” of “Universitat Politècnica de València (UPV)” in Spain. He holds a Master in Computer and Network Engineering from the UPV. His research if focused on simulation and automatic control of drone swarms.

Willian Zamora is an Associate Professor of Faculty of Computer Science, Universidad Laica Eloy Alfaro de Manabi, Ecuador. He received his Ph.D. in Informatics from Department of Computer Engineering, “Univiersitat Politècnica de València (UPV)”, Spain, in 2018. His research interest include Crowdsensing, IoT, UAVs, Data Science, and Machine Learning.

Joan Masanet Joan Masanet graduated in Informatics from the Technical University of Valencia (UPV), Spain, in 2018. His graduation project focused on solutions for the manual guidance of UAV swarms. His research interests include UAVs, embedded systems, wireless networking, and IoT applications.

Carlos T. Calafate is a full professor in the Department of Computer Engineering at the "Universitat Politècnica de València (UPV)" in Spain. He graduated with honours in the University of Oporto (Portugal) in 2001. He received his Ph.D. degree in Informatics from the UPV in 2006. His research interests include ad-hoc and vehicular networks, UAVs, Smart Cities & IoT.

Juan-Carlos Cano Juan-Carlos Cano is a full professor in the Department of Computer Engineering at the "Universitat Politècnica de València (UPV)" in Spain. He earned an MSc and a Ph.D. in Computer Science from the UPV in 1994 and 2002 respectively. His current research interests include Vehicular Networks, Mobile Ad Hoc Networks, and Pervasive Computing.

Pietro Manzoni Pietro Manzoni is a Full Professor of Computer Networks at the "Universitat Politècnica de València (UPV)", Spain. He holds a PhD from the "Politecnico di Milano", Italy. His research activity is focused on networking and mobile systems for the Internet of Things especially in the context of Community Networks and Intelligent Transport Systems. He is a senior member of the IEEE.