Received: 4 April 2022 | Revised: 14 June 2023 | Accepted: 31 August 2023

IET Intelligent Transport Systems

DOI: 10.1049/itr2.12426

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH

Assessing the limits of centralized unmanned aerial vehicle conflict management in U-Space

Pablo Boronat[1] | **Miguel Pérez-Francisco**[2] | **Jamie Wubben**[3] | **Carlos T. Calafate**[3] | **Juan Carlos Cano**[3] | **Rafael Casado**[4]

[1]Computer Languages and Systems Department, Universitat Jaume I (UJI), Castelló de la Plana, Spain

[2]Computer Science and Engineering Department, Universitat Jaume I (UJI), Castelló de la Plana, Spain

[3]Department of Computer Engineering, Universitat Politècnica de València, Valencia, Spain

[4]Computing Systems Department, Universidad de Castilla-La Mancha, Albacete, Spain

Correspondence
Miguel Pérez-Francisco, Computer Science and Engineering Department, Universitat Jaume I (UJI), Spain.
Email: mperez@uji.es

Funding information
Junta de Comunidades de Castilla-La Mancha, Grant/Award Number: SBPLY/19/180501/000159; Ministerio de Ciencia, Innovación y Universidades, Grant/Award Number: MCIN/AEI/10.13039/501100011033

Abstract

There is an important growth of unmanned aerial vehicles (UAVs) performing planned missions in urban environments, which poses significant challenges to the research community. The possibility of collisions represents a critical challenge. UAVs can suffer collisions due to different causes external or internal to their flight plans. In this context, dynamic geo-fencing is a useful approach, whereby each UAV is able to provide a prediction of its future positions within a limited time. These predictions could be used to detect conflicts, allowing to dynamically modify the flight plans so as to avoid imminent collisions. In this work, a conflict detection algorithm/method is proposed, implemented and tested on a central server performing real-time conflict analysis for a large number of UAVs flying in the aerial space of a city (U-Space). The architecture assumes that UAVs send their future locations to a traffic controller. This controller compares the predicted positions of nearby vehicles to detect possible conflicts. The results of this work demonstrate the feasibility of the proposed conflict detection algorithm and its interest to improve the security and efficiency in U-Space environments. The server is able to track thousands of UAVs in real time with a conflict anticipation around 11 s.

1 | INTRODUCTION

Unmanned aerial vehicles (UAVs), colloquially known as drones, have become a commonly used tool for different fields such as surveillance, emergency services, traffic control, assessment in topographic surveys, agricultural supervision, or rapid goods delivery, just to mention a few ([1]). The proliferation of UAVs, and the coexistence with manned aircraft, has given rise to restrictive regulations in Europe and worldwide due to obvious dangers concerning potential collisions or conflicts. In different countries or regions, these regulations are under development or have already been released; for instance, the Unmanned Aircraft System Traffic Management (UTM) in the United States ([2]), or the U-Space in the European Union ([3]).

Conflict detection in the trajectories of piloted aircraft, and their resolution, has been fundamental in traditional aviation, having as a result the present air-transportation system, which has proved to be quite robust. Yet, when addressing urban scenarios with hundreds of UAVs, new applications and problems emerge. In fact, different issues have been anticipated in the new regulations, including, for instance, the concept of Z volume in the context of U-Space, which consists in reserving, non exclusively, narrow volumes in the airspace to be used for UAVs with pre-established flying plans, possibly performing coordinated tasks, and supported by *deconfliction* services ([4]).

In this paper we propose an algorithm or method for real-time conflict detection, which is a crucial part of deconfliction services. The feasibility and effectiveness of the conflict detection algorithm is analyzed from a real implementation point of view. In particular, we provide results concerning the prediction conflict time, the amount of UAVs that can be tracked by a centralized server, the relevance of different parameters

that can be used in our detection algorithm, and the quality of the predictions regarding the number of conflicts that could be avoided.

In order to validate the proposal, two algorithms for real-time conflict detection have been developed. The first, and more obvious, is based on a single joint list that stores the updated positions of all aerial vehicles in the considered region. In this algorithm, when a new message is received, it is compared with the already stored positions in the list to detect possible conflicts. In the second algorithm, the target region is divided into cells, and each cell maintains a list of messages of flying UAVs entering the cell. For new messages, the respective cell is calculated, and the message is compared with the messages in the lists of both its cell and neighboring cells in an attempt to accelerate the process, and improve scalability. Notice that, in both algorithms, position messages are periodically sent every second, and they include, not only the current location, but also the future planned positions of the sending UAV within a short time frame. In the experiments, a look-ahead prediction time of 13 s has been used, with a granularity of 0.5 s (providing the current position plus 25 future positions).

Experimental results based on an extensive simulation study show that it is feasible to use a central server to handle U-Space traffic for a high number of UAVs flying simultaneously (more than 3,000), and without missing any collision danger. In particular, we show that our proposed spatial discretization approach is critical to achieve such high performance, even when using a standard computer as a controller server. We also show that possible critical issues are detected about 11 s before UAVs reach the actual conflict zone, offering enough margin to take preventive actions, as desired.

The rest of the paper is organized as follows: Section 2 is devoted to review and analyze other studies of interest in this field. In Section 3 we introduce the overall architecture of our proposal. The algorithms which have been implemented are presented in Section 4. Different experimental results are exposed in Section 5. Finally, in Section 6, our main conclusions and future work are presented.

2 | RELATED WORK

In this paper we focus on fleets of UAVs, each one having a coordinated flight plan to perform a mission, and probably belonging to different companies or institutions. In addition, the UAVs can share the (assigned) airspace with other manned aircraft. These scenarios are contemplated in volumes Y and Z of the U-Space ([4]).

One major issue related to such usage of the airspace is the risk of collisions. To deal with this problem, it is possible to apply *deconfliction* techniques prior to the actual flight (before the UAVs take off), as proposed in the volumes of type Y in U-Space. Another possibility is to treat the problem during flight time, as contemplated in the Z volumes defined by U-Space.

Pre-flight techniques to reduce the risk of collisions are known as *strategic*. The idea is that, before the UAVs take off, a system analyzes the flight plans of the different UAVs in the considered region or volume, and it modifies these plans to solve the detected conflicts. The problem with these approaches is that, given the uncertainty regarding drone flights in the presence of wind, GPS positioning errors, or other affecting factors, a considerable separation between UAVs must be imposed. Consequently, strategic techniques are expected to introduce a poor airspace usage as detailed in [5]. [6, 7] are examples of strategic methods for conflict detection. Thus, a better exploitation of the airspace is expected with techniques applied during flight time ([2, 3]). *In-flight* techniques can be classified as *tactical* or as *sense and avoid* techniques.

In tactical conflict resolution techniques, the philosophy is to modify the flight plan of one or several UAVs, so that the conflict disappears without violating those mentioned flight plans. These are usually centralized solutions, in which UAVs send flight data to a centralized service such as an air traffic controller. This controller analyzes the received data, looks for conflicts, and then it sends back warnings or modifications to the flight plans of the concerned vehicles. An example of a tactical method is proposed in [5] and a similar technique is proposed to assist air traffic controllers in aviation in [8].

In the case of sense and avoid solutions there is a wide range of cooperation levels than can be adopted. On the one hand, there is no cooperation and, consequently, conflict detection is done individually by using proximity sensors or artificial vision techniques ([9–11]). On the other hand, UAVs can cooperate, for instance, by exchanging position messages among them, and even by applying a coordinated avoidance protocol in case of conflict ([12–14]). Sense and avoid techniques are inevitable as a last resource to avoid collisions between vehicles or with other objects ([2]). However, they often require visual line of sight, and imply brusque evasive maneuvers which can affect the performance of the system.

Focusing specifically on centralized techniques, these are criticized for suffering larger delays and introducing a higher computation cost when compared with distributed ones. However, they have a global view of the controlled region, which enables the coordination with manned aircraft, and a better performance could be obtained if minimal modifications to the flight plans are found to solve the conflicts. In fact, Zu volumes defined in U-Space, with services still under development, will be based on such a service ([4]).

It must be noted that all mentioned techniques are far from being exclusive; instead, as many as possible should be combined to enhance safety. For instance, strategic techniques could be used to prevent conflicts among drones performing well established missions that are known in advanced, while sense and avoid techniques are necessary as a last resource to avoid collisions with other unexpected objects or manned aircraft. Any sort of cooperation will help to anticipate conflicts, and to handle them smartly so as to reduce the impact on the overall performance.

An open and general framework to manage the U-Space is presented in [15]. Modules can be added to their framework to perform and collaborate in different services. Some of these

services are expected to be progressively incorporated in the U-Space ecosystem.

Tactical techniques are composed by two related algorithms: conflict detection, and collision avoidance. From these two, conflict detection is a main issue as it must address real-time constrains. A review of collision avoidance techniques can be found in [16]. A broader review on the use of UAVs is also done in [17], where authors study the challenges and problems that need to be addressed including UAV traffic management and collision avoidance. Several studies have been done with the goal of detecting conflicts by reducing the computation time associated to this complex problem: [18–22]. However, we consider that the problem is not completely solved if thousands of UAVs have to be controlled in real-time, providing enough prediction time to apply deconfliction techniques.

Our current work has some similarity with other works such as [5] since, upon conflict detection, a central controller would send modifications to the flight plans of the involved vehicles. Yet, differently from previous works, we focus on a cooperative tactical technique for early conflict detection through an external flying controller, the Air Traffic Service (ATS). In particular, for conflict detection, we propose a novel approach where, instead of computing crossing points between trajectories, which is an inflexible and limiting approach, we check that a minimal distance in future predicted points of the vehicles is not violated, assuming that drones send periodically their current position together with a set of their future previewed positions at intervals of 0.5 s. The controller processes all these messages, tracking drone positions, and looking for future conflicts. A remarkable work based on detecting crossing points, but with a distributed method is presented in [12]. We prove through experiments, based on a real implementation, that the conflict detection system is feasible even when using a single standard server, and that it can guarantee flight safety in tactic scenarios covering a large region with thousands of UAVs.

Techniques for conflict detection close to ours are presented in [6] and [7]. In both cases, the authors propose a detection algorithm as part of a strategic deconfliction service, although they could also be used for a tactical service. These works, similarly to the work in this paper, also propose to discretize space in order to reduce the comparisons required to detect conflicts. The results in both related papers are obtained through simulations, while we have implemented the algorithm and tested it under real-time conditions, proving its applicability in tactical methods. Also, we have checked some critical parameters which can affect the performance of the implementation, such as cell size. Finally, we have measured the anticipation time for conflict prediction.

This paper addresses two aspects that, to the best of our knowledge, have not been adequately addressed in related works. First, an algorithm is proposed to track, in real time, the future trajectories of UAVs having predefined flight plans. Second, the algorithm has been implemented and tested in realistic conditions. The experimental results provide the basic hints about the feasibility and limits of a centralised deconfliction air traffic controller.

3 | SYSTEM ARCHITECTURE

Due to the ever-growing number of UAV operations in urban skies, restrictive strategic approaches based on blocking the entire flight area for the whole mission time is considered too conservative, and so more flexible techniques, like dynamic geofencing, should be enforced. Hence, we target a low-altitude U-Space environment where multiple UAVs from different authorized flight operators share a common flight area, and where every conflict (possibly leading to an actual collision) should be addressed.

Each vehicle is expected to take off vertically up to a narrow layer where they can travel horizontally to the pre-programmed points that conform their mission. This narrow horizontal space could be a Zu controlled volume of U-Space ([4]). In this Zu volume, we assume that the U-Space deconfliction service acts as a flight controller tracking all UAVs in its attributed area. The controller is in charge of detecting flight conflicts, and eventually update the flight plans of some of the UAVs involved in an imminent conflict. Such complementary topic remains outside the scope of the current work.

In this work, we analyze the capacity of a centralized ATS for conflict detection and prediction, and seek the optimal methods to maximize such capacity. If we are able to demonstrate that it is feasible to calculate all operations in real time, it means the controller would be able to apply tactical methods to solve and prevent dangerous situations with a reduced impact on the drones' pre-established flight plans.

Thus, in the depicted scenario, we assume that the UAVs communicate with the controller through a stable and reliable communications infrastructure. In the present case, communications are based on the cellular infrastructure, which is already deployed and known to provide good reliability levels compared to other wireless communications alternatives; in addition, it supports the connection of a large number of clients in a broad area thanks to the replication of base stations. It must be said that current 3G and 4G networks are optimized for terrestrial connections, meaning that antenna beaming targets ground users. Yet, optimizations in these networks and new standards will consider aerial users as well ([23]). Finally, the controller should be readily accessible via the Internet (or, alternatively, a virtualized private network). A general representation of our system architecture can be seen in Figure 1.

In the system, drones send periodically (each second) a UDP message to the server. Each of these messages contains the drone identifier, its current GPS position, and an estimation of future positions considering its flight plan, with a sampling granularity of 0.5 s. In our proposed architecture, these periodic announcements rely on UDP/IP in order to minimize delay, given that the server should receive and process in real time the messages of all tracked UAVs.

From the proposed architecture, the danger of having a single point of failure is evident in the case of a single server (i.e. the air controller). Despite this issue remains outside the scope of the paper, two solutions should be retained. First, as stated in Section 2, sense and avoid mechanisms are inevitable as a

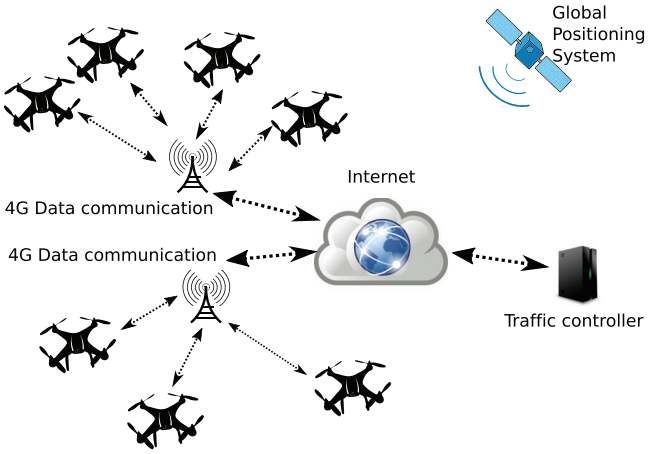

FIGURE 1 System architecture overview. UAVs are connected to the cellular infrastructure to send their path predictions to the traffic controller.

last safety resource. In this case, the whole system should continue to proceed, probably in a performance degraded mode. A second solution could consist in providing a backup server (or multiple backup servers), keeping a copy of the state of the system, and taking the role of the main server if necessary. In the case of having backup controller(s), a protocol is necessary to detect the lack of responsiveness of the main server, and for switching to the new server.

4 | DETECTION ALGORITHM

Based on the architecture and assumptions detailed previously, in this section, we proceed to detail how the proposed ATS server could handle conflicts in a centralized manner. To this end, the calculations involved in detecting conflicts between two UAVs are presented first; then, two different algorithms that focus on large-scale detection of conflicts in real time are proposed.

Concerning the conflict detection concept, our basic assumption is that, every second, the controller receives a message from each UAV containing an array with its current and future positions. The first element in these arrays is the location provided by the GPS device of the aircraft at the moment of creating the message (current position). The rest of the elements in the array are the estimations they make regarding their future positions at fixed time steps. In our tests, the time granularity adopted is half a second and it is known below as t_{step}.

The conflict detection algorithm is based on the distance between the same component of the arrays of two drones (i.e. same time instant). In our scheme, a conflict will be detected if one of the calculated distances is lower than the defined *conflict distance*. This distance usually depends on the speed of vehicles. If there is a collision danger, it must be detected at least in one of the distances calculated between the two arrays. That is, if a collision is possible between elements i and $i+1$ of both arrays, the conflict has to be detected in one (or both) of these elements (i.e. positions i, or $i+1$). The worst case happens when two UAVs are flying in opposite directions at the maximum speed. If they were able to travel the entire conflict distance during the elapsed time between consecutive components of the array (t_{step}), then there is a chance that the conflict would remain undetected, meaning that a collision could take place. This particular worst-case condition is illustrated in Figure 2.

Being V_{max} the maximum drone speed, and t_{step} the time between the consecutive elements of the position arrays, Equation (1) shows the condition for the conflict distance to guarantee conflict detection even in worst-case scenarios.

$$2 \cdot conflict\_distance > 2 \cdot V_{max} \cdot t_{step}. \quad (1)$$

However, we consider a more conservative value for the conflict distance, adding a possible GPS error (assumed to be about 5 m in [24]), which is counted twice (one per drone), and an additional margin of uncertainty, as shown in Equation (2).

$$conflict\_distance = V_{max} \cdot t_{step} + 2 \cdot GPS\_error + margin. \quad (2)$$

It must be noted that, if a conflict is detected in the i component of the arrays, it will take place in about $(i \cdot t_{step}) - t_{com} - t_{proc}$ seconds, where t_{com} is the communication time, and t_{proc} is the computation time. In the tests conducted in [25] under real conditions, the communication time through the 4G infrastructure was close to 50 ms, while the processing time was mostly load dependent, and different approaches to reduce this time are presented further in this paper. Also, the conflict distance depends on the maximum speed and on the t_{step}. If UAVs increment the maximum speed, the time between predicted positions should be reduced in order to maintain a reasonable conflict distance (i.e. to keep down the number of the potential conflicts which would not end up colliding).

In the conflict detection algorithms presented below, a *filter distance* is also used in order to look up for conflicts between vehicles which are relatively close, while avoiding conflict detection between all UAVs in the target U-Space region. Such filter distance is computed roughly, comparing the difference of each dimension (*northing* and *easting*) of the first element of the positions arrays (what should correspond to their more recent location), enabling a quick discard. Concerning the conflict distance, the actual filter distance also depends on the maximum speed of the UAVs. It is fixed to twice the distance that the drones can fly during all time steps of the positions array, given that the worst-case scenario is presented again when two UAVs are flying in opposite direction at the maximum speed.

In the experiments, time steps (t_{step}) of 0.5 s have been used, resulting in 13 s for the 27 positions stored in each array (reaching the payload capacity of a UDP datagram). This provides a filter distance of about 250 m if drones have a maximum speed of 10 m/s. That is, UAVs which are farthest than this distance can be quickly discarded as they will not be in conflict in the next 13 s. Note that position messages, as stated above, are sent each second (to match the maximum frequency of standard GPS devices). However, the prediction each UAV makes regarding its future positions (taking into account the

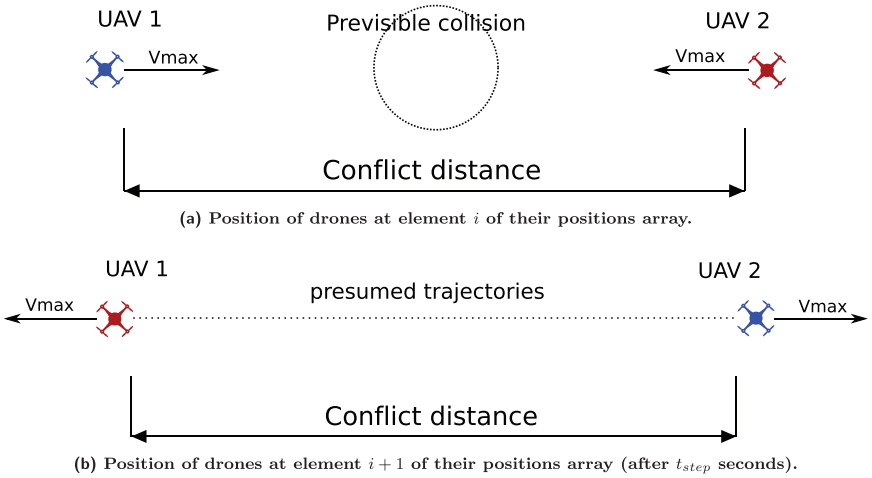

FIGURE 2 Example of the positions of two UAVs in two consecutive elements of their positions arrays (i, and $i+1$). If Equation (1) is not satisfied, a collision could remain undetected.

(a) Position of drones at element i of their positions array.

(b) Position of drones at element $i+1$ of their positions array (after t_{step} seconds).

current position, and its flight plan) takes place every t_{step} seconds.

Having now presented the different equations and distances that regulate the detection strategy, two different algorithms for the large-scale detection of conflicts on a single server are presented next.

4.1 | Single list algorithm

To start, a simple algorithm for conflict detection based on the elements that have been introduced is presented. In this algorithm, data with the position announcements from all UAVs are stored in a single list. For each new received message, the list has to be traversed looking for conflicts. This is the *List-algorithm*, and it is shown in Algorithm 1.

Algorithm 1 shows that there is a single list containing the information of all UAVs in the covered region. The elements of this list correspond to messages sent by the drones (one per UAV), creating a data structure that includes, on each entry, the UAV identifier, plus an array with the positions where the vehicle predicts it will be every t_{step} interval. The first element of the position array therefore corresponds to the last position the UAV read from its GPS interface. The UAVs will send a new message each second to update its positions in the list of the controller. UAVs send position messages each second as this is the frequency limit for GPS devices of common use. Hence, our flight controller has the real-time restriction of having to process all incoming messages from the different UAVs in less than 1 s, including the needed calculations, or otherwise messages would start to accumulate.

The condition in line 3 is true when the identity of the message is found in the list. If it is the case, the element of the list is removed, and the data of this UAV is updated when the process is finished (line 16). The loop in line 7 iterates in the positions arrays, checking the distance between each pair of positions.

In this algorithm, data for each conflict found are stored in a list (line 9). These conflicts should be further processed in a *tactical deconfliction* module. Such module will run independently, and it remains outside the scope of this work.

ALGORITHM 1 List-algorithm

```
1:  procedure ProcessMessageList(vehicleList, message) vehicleList is the list of
      all UAVs in the server region, and message is a positions message from
      a UAV.
2:    for v ∈ vehicleList do
3:      if v.id == message.id then
4:        vehicleList.remove(v)
5:      else
6:        if filterDistance(message.positions[0], v.positions[0]) then
7:          for i = 0; i < length(v.positions); i++ do
8:            if distance(message.positions[i], v.positions[i]) < conflictDistance
      then
9:              conflictList.append(message.id, v.id, timestamp(), i)
10:             break
11:           end if
12:         end for
13:       end if
14:     end if
15:   end for
16:   vehicleList.append(message)
17: end procedure
```

It must be said that the shown algorithm is a simplified version for easy understanding. Implementations of the algorithm have also to timestamp the elements in the list to detect outdated data (>5 s old), enabling us to perform some housekeeping; that is applicable to, e.g., UAVs that have already ended their mission.

The *filterDistance*() function used in line 6 of Algorithm 1 can be seen in Algorithm 2. It is used to select a nearby aircraft for which it is worth looking for potential conflicts. And finally, the *distance*() function (line 8 of Algorithm 1) calculates the distance between two points by merely applying the Pythagoras formula.

Notice how the computational cost of Algorithm 1 is quadratic with the amount of simultaneous UAVs given that the entire list has to be parsed for every single message arriving from each UAV ($\Theta(n^2)$). The most costly operations are only

ALGORITHM 2 filterDistance

```
1:  procedure filterDistance(position1, position2) Takes two positions and
    returns true if the difference of both dimensions is lower than a
    DISTANCE constant.
2:    if |position1.x − position2.x| < DISTANCE then
3:      if |position1.y − position2.y| < DISTANCE then
4:        return TRUE
5:      end if
6:    end if
7:    return FALSE
8:  end procedure
```

executed for UAVs closer than the filter distance. This consideration improves the computation time, but it does not reduce its quadratic cost.

4.2 | Space discretization (grid) algorithm

Algorithm 1 has the main drawback of having to detect potential conflicts with all UAVs in the region supervised by the traffic controller (at least the filter distance rule must be enforced). In an attempt to improve efficiency, a second algorithm has been proposed where the target region is divided in square cells forming a grid. Such *spatial discretization* is expected to significantly alleviate the air traffic controller's task. The *Grid-algorithm* is shown in Algorithm 3.

In particular, in the Grid-algorithm, each cell maintains a list of aircraft that are within its perimeter. Thus, to process a new message, the cell corresponding to the first component of its positions array (i.e. its current cell) is initially determined (line2); then, the algorithm looks for potential conflicts on that same cell, and on neighboring cells whose limits remain within the filter distance (loops in lines 3 and 4), thereby restricting safety checks (loop in line 9) to a minimum.

In Algorithm 3 the elements of the *grid* array are the cells. As previously explained, each cell contains a list with the positions' data for UAVs within its bounds. Notice that the actual code implementing Algorithm 3 has to carefully address processing in border cells. Here we have obviated such detailed checks for the sake of simplicity and readability.

In the Grid-algorithm, *RADIUS* (used in lines 3 and 4) is a constant with the range of cells considered according to the defined filter distance. As an example, Figure 3 shows the cells to be used for conflict detection (green cells), according to the distance filter defined, when the message being processed belongs to the yellow cell. In this example, the side of cells is 125 m, the distance filter is of 250 m, and *RADIUS* is equal to 2.

The *getCell*() function in line 2 of Algorithm 3 is detailed in Algorithm 4. The function uses as constants the position of the down-left corner of the target airspace region (*EAST_LOW*, *NORTH_LOW*), and the size of the cells' side (*CELL_DIM*); given an input location, it returns the identity of

ALGORITHM 3 Grid-algorithm

```
1:  procedure ProcessMessageGrid(grid, message)grid is the bidimensional
    array of cells which cover the server region, and message is a positions
    message from an UAV.
2:    (cell_x, cell_y) = getCell(message.positions[0])
3:    for x ∈ range(cell_x − RADIUS : cell_x + RADIUS) do
4:      for y ∈ range(cell_y − RADIUS : cell_y + RADIUS) do
5:        for v ∈ grid[x, y].vehicleList do
6:          if message.id == v.id then
7:            grid[x, y].vehicleList.delete(v)
8:          else
9:            for i = 0; i < length(v.positions); i++ do
10:           if distance(message.positions[i], v.positions[i]) < conflictDistance
    then
11:             conflictList.append(message.id, v.id, timestamp(), i)
12:             break
13:           end if
14:           i++
15:           end for
16:         end if
17:       end for
18:     end for
19:   end for
20:   grid[cell_x, cell_y].vehicleList.append(message)
21: end procedure
```

ALGORITHM 4 getCell

```
1:  procedure getCell(position) Given a position returns the respective cell
    in the grid.
2:    i = ⌊(position.x − EAST_LOW)/CELL_DIM⌋
3:    j = ⌊(position.y − NORTH_LOW)/CELL_DIM⌋
4:    return(i, j)
5:  end procedure
```

the respective cell in the bi-dimensional array of cells (the *grid* data structure used in the Grid-algorithm).

It is worth to point out that, similarly to the Single-list algorithm, the computational cost of the Grid-algorithm is also quadratic, but the execution logic is significantly more efficient for each message, as only messages from aircraft falling within the defined *radius* have to be processed. If we consider a uniform distribution of the UAVs in a square region, the computational complexity of the grid-algorithm would be $\Theta(nk)$, where n is the total number of vehicles, and k represents the UAVs belonging to the block of cells to be tested. That is, $k = (n/c^2) \times (2R+1)^2$, being c the number of cells in each dimension of the grid of cells, and R the value of the *RADIUS* constant. Analyzing these variables, for instance in the tests presented below, typical values can be $n = 3,000$, $c = 10,000/125$ and $R = 2$, which represents an important cost reduction when

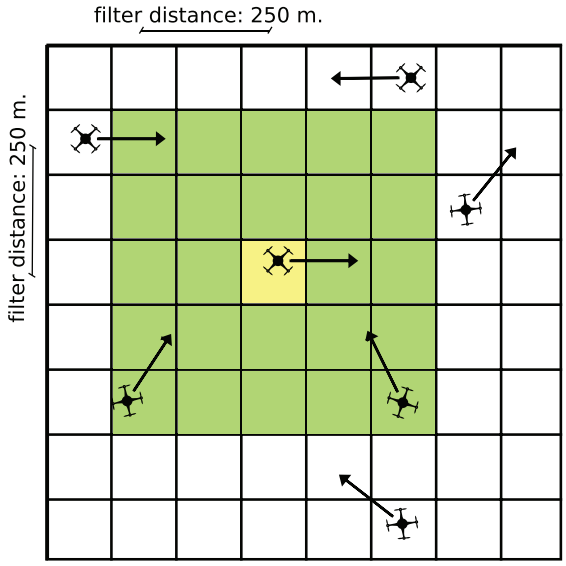

FIGURE 3 Green cells are within the *radius* of the yellow one. In this example, the cells are of 125 m of side, and the distance filter is 250 m. The *RADIUS* constant of Algorithm 3 is set to 2.

compared to the list-algorithm. We further elaborate on these improvements in Section 5.3.

As explained above, the grid-algorithm seeks to significantly lower its execution time; however, the downside of this approach is that a more complex data structure must be maintained. For instance, for cells sized 125×125 m^2 (i.e. half the filter distance), a region of 10×10 km^2 needs a two-dimensional array of 80×80 cells. So, whenever the controlled regions have a reasonable dimension, it should not represent a computational problem, as shown in Section 5. Otherwise, further space partitioning should be enforced via additional traffic management servers.

5 | EXPERIMENTAL RESULTS

In this section, we will assess the feasibility and effectiveness of our proposed air traffic management system for low-altitude UAV flights. To this end, we will first describe our test environment; then, we will validate that the system is operating correctly. In the sections that follow, we then evaluate the impact of different parameters on performance to determine the limits of our solution.

5.1 | Test environment

To check the presented conflict detection system, we have implemented both algorithms proposed, and we have conducted a wide set of tests. In the experiments, the controller part is the real software receiving messages from the Internet, and recording the detected conflicts. However, the drones themselves have been simulated with the ArduSim UAV simulator ([26]). Up to 15 computers have been used for these simulations. The server and the computers simulating the drones are in different IP networks connected to Fast Ethernet LANs (100 Mbps), to approximate to the conditions where different UAVs may be connected to the wireless infrastructure from different points, and via different providers.

It must be said that the communications part of the environment test is not fully realistic, given that packet losses due to the wireless 4G phone system to connect the UAVs are not considered (optimal conditions are assumed). Nevertheless the controller deals with this possibility in the following way: if the communications link towards a specific UAV is temporarily broken, the server estimates its current position based on the applicable future element in the positions array that was included in the last received message. It is also worth pointing out that, according to [25], tests on the 4G phone system were conducted, and the stability and reliability tests showed that such networks suit well the proposed real-time reactive system. A similar treatment is given if a UAV loses contact with the GPS system. It must be taken into account that UAV missions based on flight plans are only possible if they have an accurate geographical knowledge.

In the tests, the air traffic controller software has been executed in a server with twelve Intel Core(TM) i7-8700 CPU 3.20 GHz, 24 Gbytes of RAM, and running Debian GNU/Linux 10.10. To avoid an early overflow of the UDP buffer due to the high load expected, we have increased it from 208 Kbytes, which is the default value, to 64 MBytes. The software implementation of the traffic controller was done with Python. The Python interpreter has the well known limitation of real concurrency for the execution of threads, but, even with this limitation, the results obtained prove the feasibility of the proposal.

The simulated drones have used the following displacement pattern: for each drone, two points (departure and arrival) inside the area covered by the traffic controller are chosen within a range of 2-6 km. The drones travel from the departure point to the destination at a constant speed of 10 m/s. Figure 4 shows an example of 100 drones flying over Valencia (Spain) in a square region of 10×10 km^2. The length of the flights have been selected to ensure that the simulated drones are flying simultaneously, and thus providing enough time to observe the effect of the load on the controller. These simple flight plans allow us to overload the air traffic controller. Yet, it is worth mentioning that real-life scenarios usually involve flights with multiple waypoints, meaning that flight trajectories will not be straight all the time.

For these experiments we have used a t_{step} of 0.5 s, and the drones embed 27 positions in each message, sending them with a frequency of one message per second. Table 1 shows the default values used in the following tests if not stated otherwise.

5.2 | System validation

To validate the prediction method, we have conducted several tests comparing the collisions registered by ArduSim with the conflicts detected by our server. In these tests we have simulated just 100 UAVs, as this was our simulation limit on a single

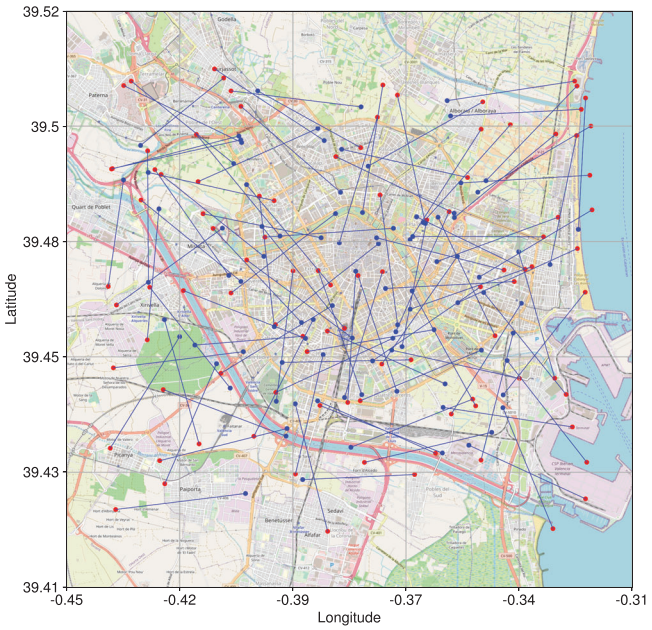

FIGURE 4 100 drones following simple missions in an area of 10×10 km^2. Departure points are marked in red, and arrival points in blue.

TABLE 1 Default values for the different parameters used in the experiments.

Parameter	Value
Side of the controlled region	10 km
Cells side	250 m
Collision distance	10 m
Conflict distance	25 m (20 + 5 m for uncertainty margin)
V_{max}	10 m/s
t_{step}	0.5 s

PC; notice that we cannot distribute ArduSim simulations over several computers as ArduSim collisions are only detectable in the domain of a single host. These tests were made in a square region sized 2×2 km^2, with cells equals to 250 m of side (i.e. a mean traffic density of 1.56 drones per cell) and a constant speed of 10 m/s.

Table 2 shows the number of collisions registered by ArduSim, and the number of potential conflicts. The data corresponds to the average of three independent tests. Potential conflicts are expected, specially when the conflict distance is greater than the collision distance. Note that non-detected collisions can take place when the limit set in Equation (2) is not satisfied, as for conflict distances below 20 m (plus 5 m of uncertainty margin to account for real-life conditions).

5.3 | Scalability analysis

The aim of this first set of tests is to detect the traffic controller saturation point. Beyond this load level, the controller is not able

TABLE 2 Validation tests in a region of 2×2 km^2 with 100 UAVs. The collision distance is 10 m and the conflict distance range between 10 and 40 m.

Conflict distance	Collisions	Potential conflicts	Undetected collisions
10	84	63	30.8
15	76.6	80.3	11
20	91.7	125.25	0
25	81	136	0
30	83.5	165.5	0
35	89.5	210	0
40	88.4	229.2	0

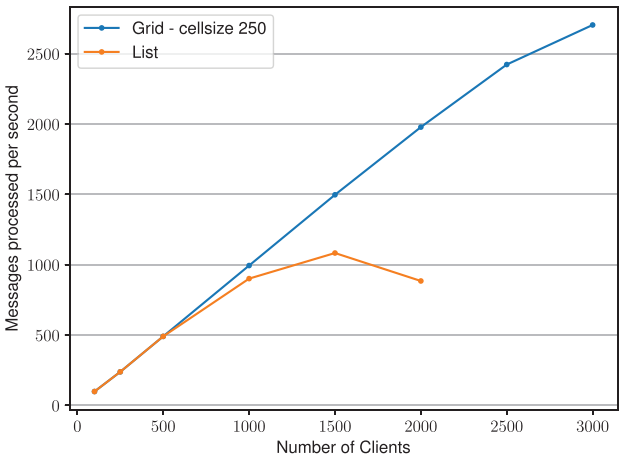

FIGURE 5 Messages processed per second in the server as function of the number of UAVs. The drones are moving in a square area of 10 km of side, the filter distance is 250 m, and the cell side used is also 250 m (grid algorithm).

to process all messages sent by the aircraft in less than 1 s, and so the input queue overflows.

To detect the saturation point, we count the number of messages that the server processes per second. When this number is smaller than the number of clients, it means that the server cannot attend all flying vehicles in real time. Figure 5 shows the saturation point for both algorithms when using a square region of 10 km of side. In these tests, the list-algorithm server begins to saturate with 1,000 clients, while the grid-algorithm is able to meet real-time responsiveness for up to 2,500 clients.

Figure 6 shows the response time to process each message. It can be seen that this time grows with the number of UAVs in the list-based algorithm, while it remains almost constant in the grid-based algorithm. This unexpected behavior of the grid-based algorithm is explained by the reduced amount of drones in each cell (for 2,500 drones, the density is just 1.56 drones per cell).

5.4 | Effect of traffic density

To provide insight on how the density of drones (i.e. the amount of drones per area unit) affects the server performance, we have

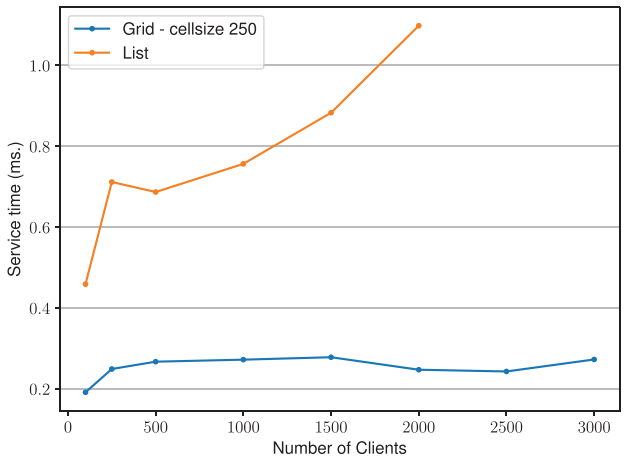

FIGURE 6 Response time per message as function of the number of UAVs. The drones are moving in a square area of 10 km of side.

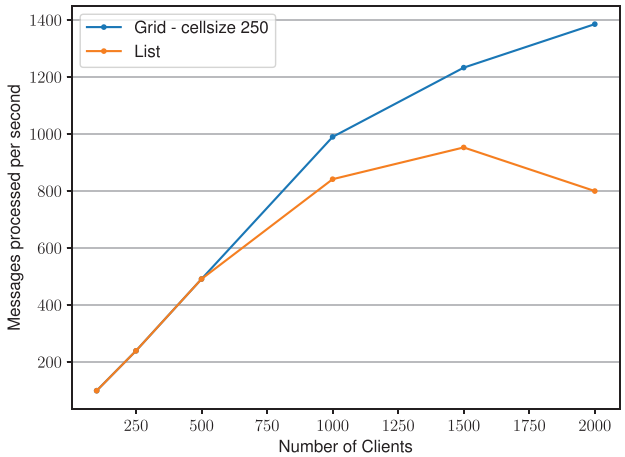

FIGURE 7 Messages processed per second in the server as function of the number of clients in a square region of 5 km of side.

made tests reducing the area covered by the traffic controller to a square of 5 × 5 km^2. In these experiments, the distance filter and the side of cells have been also fixed at 250 m. So, for a total of 1,000 drones, the density of drones increases to 2.5 drones per cell, compared to the previous experiments where the density was of 0.625 drones per cell.

It can be observed in Figure 7 that, under these higher density conditions, both versions of the server saturate with a smaller number of clients, as increasing the density of vehicles also increases the amount of vehicles within close range of each other, thereby increasing the number of comparisons to be made. The list-based algorithm fails to meet real-time requirements beyond 500 clients, while the grid-based algorithm does the same for about 1,000 clients.

Figure 8 shows the response time to process each message as a function of the number of clients. Again, the list-based algorithm is more affected by the number of clients than the grid-based algorithm.

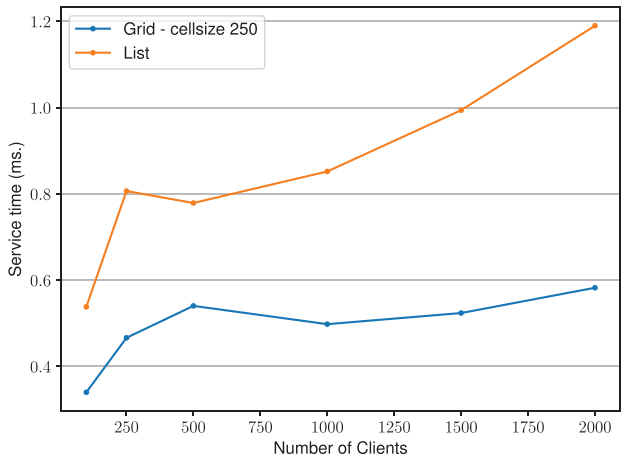

FIGURE 8 Response time per message as function of the number of UAVs in a square region of 5 km of side.

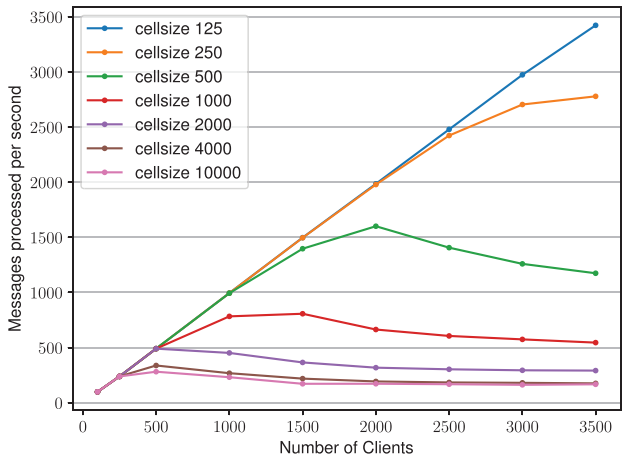

FIGURE 9 Messages processed per second as function of the number of UAVs. The drones are moving in a square region of 10 km of side. Each line is obtained with a different size of cells, ranging from 125 to 10,000 m (full map).

5.5 | Impact of the spatial granularity level

Initially, for the grid-based algorithm, we used a cell size equals to 250 m as it matches the filter distance. The idea behind this choice was that the radius constant, used in Algorithm 3, becomes 1, meaning that only neighboring cells must be tested. Yet, this is not necessarily the most efficient approach.

Figure 9 shows the saturation of the server for different sizes of cells with a filter distance of 250 m. These tests were made in a square region of 10 × 10 km^2. The respective response times per message can be seen in Figure 10.

In these tests, the best result is for cells of 125 m of side (half of the filter distance). As shown, we found that larger cell sizes have a negative impact on performance. This occurs because smaller cell sizes are able to map the filter distance more accurately, thereby reducing the number of UAVs to be compared. With cells of 125 × 125 m^2, the controller is able to handle up to 3,500 clients. Also notice that a finer granularity of the grid increases the overall number of cells, which requires a larger data

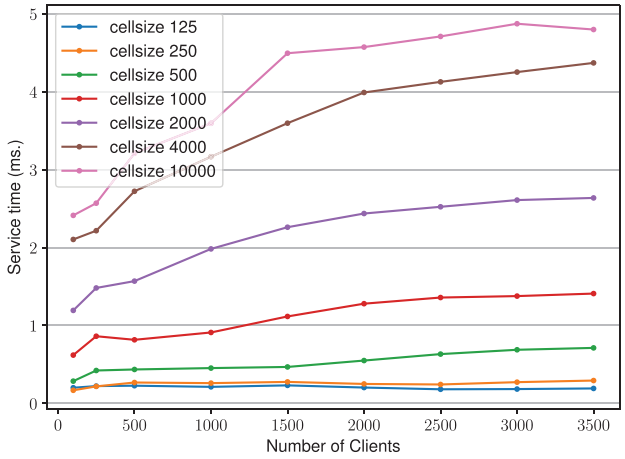

FIGURE 10 Response time per message as function of the number of UAVs. The drones are moving in a square region of 10 km of side. Each line is obtained with a different size of cells, ranging from 125 to 10,000 m (full map).

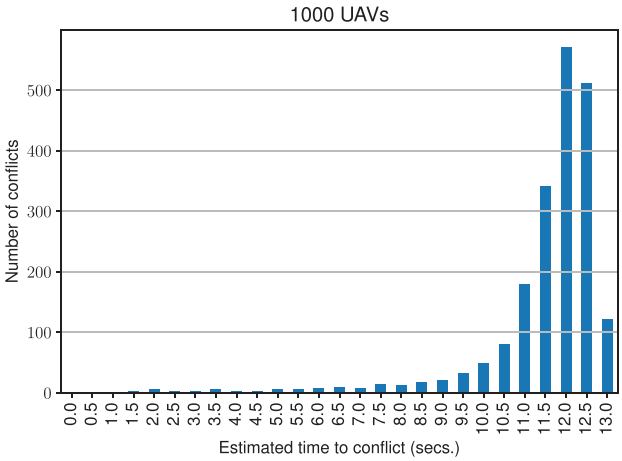

FIGURE 12 Amount of conflicts detected per element in the positions array in the grid-based algorithm for 1,000 UAVs with 250 m cells' side and a region of 10×10 km^2.

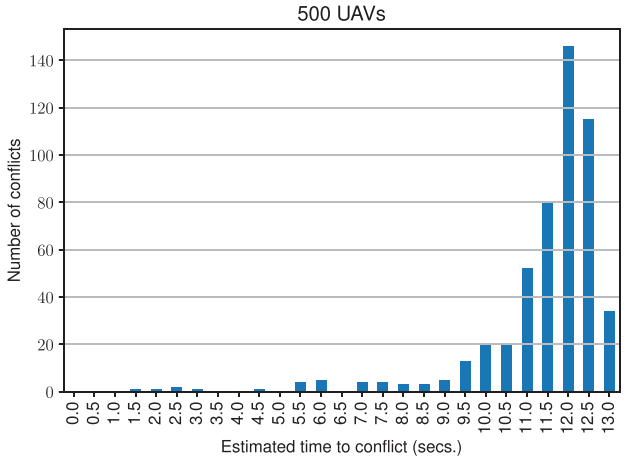

FIGURE 11 Amount of conflicts detected per element in the positions array in the grid-based algorithm for 500 UAVs with 250 m cells' side and a region of 10×10 km^2.

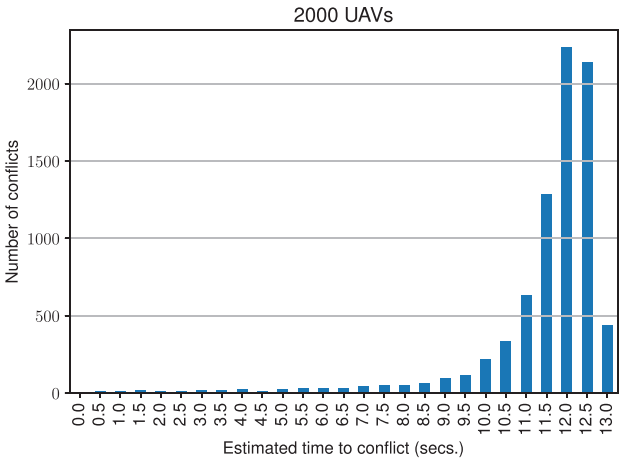

FIGURE 13 Amount of conflicts detected per element in the positions array in the grid-based algorithm for 2,000 UAVs with 250 m cells' side and a region of 10×10 km^2.

structure, but in this case we find this option to be, nevertheless, overall beneficial.

5.6 | Anticipation in the conflict prediction

To evaluate the conflict prediction capacity of the proposed system, we have checked how soon are we able to anticipate conflicts. This look ahead time margin can be deduced by the component in the positions arrays in which a conflict is detected, as the greater is the array index, the more time is available. Obtaining the distribution of the prediction time is interesting because it provides an idea about the time margin the controller will have to prioritize, and solve conflicts.

Figures 11–13 show that most of the conflicts are detected around element 25 of the position arrays, that is, between 11 and 13 s before the conflict takes place. These figures are obtained with a square region of 10×10 km^2 of side, and cells of 250 m of side. The three figures differ in the number of clients, with 500, 1,000 and 2,000 UAVs, respectively.

We consider that the minimum time margin to deal with conflicts should allow enough time to find a solution to the conflict, communicate the new flight plan to the affected UAVs, and consider a non-zero reaction time. We now proceed by discussing these times.

The process time needed to provide a solution for a predicted conflict will depend on the actual strategy enforced. In [5] a computation time per UAV involved in a conflict as a function of the number of UAVs in the system is considered. In their approach, a solution is computed in a loop, and it is modified until no further conflicts are generated. For 100 UAVs, this time is about 0.9 ms, but it grows exponentially with the total number of drones. Yet, it is reasonable to assume that it can be handled in less than 1 s.

Regarding the communications time, in [25] we measured the mean communication round trip time in real mobile scenarios,

finding results close to 0.1 s. In these tests, messages were sent by vehicles to a server on the Internet through the 4G wireless infrastructure, and the server then sends back a confirmation message (round-trip time).

The reaction time for the drones (i.e. the time to apply a modification to its flying plan) depends on their speed and on the type of maneuver. If we exclude the possibility to go backwards, the worst case is having to completely stop the drone, which we estimate in about 3.4 s for a maximum speed of 10 m/s according to [27]. Thus, the minimum prediction time margin could be approximated to 4.45 s in the worst case scenario, a value which evidences the feasibility of the proposed prediction system, which can detect conflicts with an anticipation of about 11 s in most cases.

In Figures 11–13 we can see that there are no conflicts detected in the first element of the positions array. We have forced these results by starting all UAVs at the same time and ensuring that the distance between the starting points of the drones is bigger than the conflict distance. In a real scenario, drones should start to send their position messages to the controller at least 1 s before taking off.

6 | CONCLUSIONS AND FUTURE WORK

The urban airspace is a limited and coveted resource for UAV operators in different countries. In order to optimize the use of this resource, automated solutions for the coordination of UAVs, such as air traffic control services, are crucial. The goal of these services is twofold: avoid the risk of collisions, and minimize disruptions to the original flight plans. The task of air traffic controllers is, therefore, to track the UAVs flying in a pre-established region or volume, and they should be able to detect conflicts and so correct the flight plans of the involved drones.

The main contribution of this paper is the proposal of a conflict detection technique whose basic requirements are that UAVs periodically send to the controller a prediction of their immediate-future positions. In addition, the conflict detection technique has been tested in realistic scenarios. This enables the flight controller to detect conflicts, and react in real time.

We have developed implementations of two algorithms, the List-algorithm where all UAVs in the region assigned to the controller are handled using a joint list, and the Grid-algorithm where the target region is divided into cells, and where an independent list of vehicles is maintained per cell. Exhaustive simulation experiments show that the algorithm based on space discretization (cells) largely outperforms the list-based algorithm, especially when selecting small cell sizes. Overall, we find that our conflict detection system, when running on a single, standard server, can still deal with thousands of UAVs easily, and even in high traffic density situations, providing enough time margin (around 11 s) for the application of conflict resolution techniques.

The results obtained through experimentation using a true functional server communicating with a network of simulated UAVs evidence the applicability of the work. Both in [5] and [6], which are close papers, experiments are done through simulation. In [5] the authors simulate up to 100 UAVs, but the conflict detection part is not presented separately from the whole deconflition technique. In [6] results with 500 UAVs are shown, but the conflict detection technique is presented as part of a strategic method, applied prior to the flight time with a granularity of 1 s. Similarly, in [7], there are simulations with up to 1,200 UAVs, again prior to flight time. In our experiments, we are able to track up to 3,500 UAVs under real-time restrictions, detecting all conflicts with a granularity of 0,5 s.

As future work, we plan to develop a conflict resolution technique which will be fed with the output of our conflict detection system, thereby contributing to the development of advanced air traffic control services, meeting the requirements of U-Space European regulations. Another line of work consists in developing a load balance system when an air controller starts to be saturated and, consequently, real-time restrictions are compromised.

AUTHOR CONTRIBUTIONS

Pablo Boronat: Conceptualization, formal analysis, investigation, methodology, software, validation, writing - original draft, writing - review and editing. Miguel Pérez-Francisco: Conceptualization, formal analysis, investigation, methodology, software, writing - review and editing. Jamie Wubben: Software, writing - review and editing. Carlos Tavares Calafate: Formal analysis, funding acquisition, supervision, writing - review and editing. Juan-Carlos Cano: Funding acquisition, supervision, writing - review and editing. Rafael Casado: Funding acquisition, supervision, writing - review and editing.

ACKNOWLEDGEMENTS

This work is derived from the following R&D projects: PID2021-122580NB-I00 and RTI2018-098156-B-C52, funded by MCIN/AEI/10.13039/501100011033, "ERDF A way of making Europe", and SBPLY/19/180501/000159, funded by the Junta de Comunidades de Castilla-La Mancha (JCCM) and the EU through the European Regional Development Fund (ERDF-FEDER).

CONFLICT OF INTEREST STATEMENT

The authors declare no conflicts of interest.

DATA AVAILABILITY STATEMENT

Data available on request from the authors.

ORCID

Pablo Boronat https://orcid.org/0000-0001-7850-8179
Miguel Pérez-Francisco https://orcid.org/0000-0003-3831-4075
Jamie Wubben https://orcid.org/0000-0001-8121-995X
Carlos T. Calafate https://orcid.org/0000-0001-5729-3041
Juan Carlos Cano https://orcid.org/0000-0002-0038-0539
Rafael Casado https://orcid.org/0000-0002-5170-5743

REFERENCES

1. Shakhatreh, H., Sawalmeh, A.H., Al-Fuqaha, A., Dou, Z., Almaita, E., Khalil, I., et al.: Unmanned aerial vehicles (UAVs): A survey on civil

applications and key research challenges. IEEE Access 7, 48572–48634 (2019)
2. Federal Aviation Administration OoN: UTM Concept of Operations V 2.0. 800 Independence Ave., SW. Washington, DC 20591 (2020)
3. CORUS-Project: U-Space Concept of Operations. SESAR Joint Undertaking, Av. de Cortenberh, 100, 1000 Brussels (2019)
4. Barrado, C., Boyero, M., Brucculeri, L., Ferrara, G., Hately, A., Hullah, P., et al: U-space concept of operations: A key enabler for opening airspace to emerging low-altitude operations. Aerospace 7(3), 24 (2020)
5. Jover, J., Bermúdez, A., Casado, R.: A tactical conflict resolution proposal for U-space Zu airspace volumes. Sensors 21(16), 5649 (2021)
6. Acevedo, J.J., Castaño, Á.R., Andrade-Pineda, J.L., Ollero, A.: A 4D grid based approach for efficient conflict detection in large-scale multi-UAV scenarios. In: 2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS), pp. 18–23. IEEE, Piscataway (2019)
7. Yang, J., Hu, M., Su, F.: Unmanned aerial vehicle conflict detection based on spatiotemporal data grid. J. Phys. Conf. Ser. 2246(1), 012032 (2022). Available from: https://dx.doi.org/10.1088/1742-6596/2246/1/012032
8. Wang, Z., Li, H., Wang, J., Shen, F.: Deep reinforcement learning based conflict detection and resolution in air traffic control. IET Intel. Transport Syst. 13(6), 1041–1047 (2019)
9. Park, J., Cho, N.: Collision avoidance of hexacopter UAV based on LiDAR data in dynamic environment. Remote Sens. 12(6), 975 (2020). Available from: https://www.mdpi.com/2072-4292/12/6/975
10. Zsedrovits, T., Bauer, P., Zarandy, A., Vanek, B., Bokor, J., Roska, T.: Error analysis of algorithms for camera rotation calculation in GPS/IMU/camera fusion for UAV sense and avoid systems. In: 2014 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 864–875. IEEE, Piscataway (2014)
11. Fasano, G., Accado, D., Moccia, A., Moroney, D.: Sense and avoid for unmanned aircraft systems. IEEE Aerosp. Electron. Syst. Mag. 31(11), 82–110 (2016)
12. Mahjri, I., Dhraief, A., Belghith, A., AlMogren, A.S.: SLIDE: A straight line conflict detection and alerting algorithm for multiple unmanned aerial vehicles. IEEE Trans. Mob. Comput. 17(5), 1190–1203 (2018)
13. Fabra, F., Calafate, C.T., Cano, J.C., Manzoni, P.: MBCAP: Mission based collision avoidance protocol for UAVs. In: 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA), pp. 579–586. IEEE, Piscataway (2018)
14. Sánchez, P., Casado, R., Bermúdez, A.: Real-time collision-free navigation of multiple UAVs based on bounding boxes. Electronics 9(10), 1632 (2020). Available from: https://www.mdpi.com/2079-9292/9/10/1632.
15. Capitán, C., Pérez-León, H., Capitán, J., Castaño, Á., Ollero, A.: Unmanned aerial traffic management system architecture for U-space in-flight services. Appl. Sci. 11(9), 3995 (2021)
16. Yasin, J.N., Mohamed, S.A.S., Haghbayan, M.H., Heikkonen, J., Tenhunen, H., Plosila, J.: Unmanned aerial vehicles (UAVs): Collision avoidance systems and approaches. IEEE Access 8, 105139–105155 (2020)
17. Hildmann, H., Kovacs, E.: Review: Using unmanned aerial vehicles (UAVs) as mobile sensing platforms (MSPs) for disaster response, civil security and public safety. Drones 3(3), 59 (2019). Available from: https://www.mdpi.com/2504-446X/3/3/59
18. Tang, H., Robinson, J.E., Denery, D.G.: Tactical Conflict Detection in Terminal Airspace. J. Guid. Control Dyn. 34(2), 403–413 (2011). Available from: https://doi.org/10.2514/1.51898
19. Velasco, G.A.M., Borst, C., Ellerbroek, J., Van Paassen, M., Mulder, M.: The use of intent information in conflict detection and resolution models based on dynamic velocity obstacles. IEEE Trans. Intell. Transp. Syst. 16(4),2297–2302 (2015).
20. Yang, J., Yin, D., Niu, Y., Zhu, L.: Cooperative conflict detection and resolution of civil unmanned aerial vehicles in metropolis. Adv. Mech. Eng. 8(6), 1687814016651195 (2016)
21. Ma, L., Gao, Y., Yin, T., Zhai, W.: Improved flight conflict detection algorithm based on gauss-hermite particle filter. WWuhan Univ. J. Natural Sci. 22(3), 269–276 (2017)
22. Zhang, H., Zhang, J., Zhong, G., Liu, H., Liu, W.: Multivariate combined collision detection for multi-unmanned aircraft systems. IEEE Access 10, 103827–103839 (2022)
23. Nguyen, H.C., Amorim, R., Wigard, J., KováCs, I.Z., Sørensen, T.B., Mogensen, P.E.: How to ensure reliable connectivity for aerial vehicles over cellular networks. IEEE Access 6, 12304–12317 (2018)
24. Liebner, M., Klanner, F., Stiller, C.: Active safety for vulnerable road users based on smartphone position data. In: 2013 IEEE Intelligent Vehicles Symposium (IV), pp. 256–261. IEEE, Piscataway (2013)
25. Boronat, P., Pérez-Francisco, M., Calafate, C.T., Cano, J.C.: Towards a sustainable city for cyclists: Promoting safety through a mobile sensing application. Sensors 21(6), 2116 (2021)
26. Fabra, F., Calafate, C.T., Cano, J.C., Manzoni, P.: ArduSim: Accurate and real-time multicopter simulation. Simul. Modell. Pract. Theory 87, 170–190 (2018)
27. Fabra, F., Zamora, W., Sanguesa, J., Calafate, C., Cano, J.C., Manzoni, P.: A distributed approach for collision avoidance between multirotor UAVs following planned missions. Sensors 19, 2404 (2019)

How to cite this article: Boronat, P., Pérez-Francisco, M., Wubben, J., Calafate, C.T., Cano, J.C., Casado, R.: Assessing the limits of centralized unmanned aerial vehicle conflict management in U-Space. IET Intell. Transp. Syst. 1–12 (2023).
https://doi.org/10.1049/itr2.12426