Assignment And Take-off Approaches For Large-scale Autonomous UAV Swarms

Jamie Wubben[1], Daniel Hernández[1], José M. Cecilia[1], Baldomero Imberón[2],
Carlos T. Calafate[1], Juan-Carlos Cano[1], Pietro Manzoni[1], Chai Keong Toh[2]
[1]Departament of Computer Engineering (DISCA)
Universitat Politècnica de València, Valencia, Spain
[2]Departament of Computer Science
Universidad Católica de Murcia (UCAM), Murcia, Spain
[3]GLG Group, San Francisco, USA
Email: jwubben@disca.upv.es, dhervic@doctor.upv.es, jmcecilia@disca.upv.es, bimbernon@ucam.edu,
{calafate,jucano,pmanzoni}@disca.upv.es, ck_away@hotmail.com

Abstract—In the last decade, the popularity of UAVs has increased tremendously. Nowadays, many researchers are interested in UAV swarms. Coordinating a swarm of UAVs is a complicated task and many problems should be addressed before wide-spread adoption. In this work, we focus on the take-off for large-scale UAV swarms, with an extra focus on the assignment phase. The assignment phase is the first take-off stage whereby we decide which UAV on the ground goes to which place in the air. A good assignment algorithm, is quick, and at the same time reduce the total distance travelled as much as possible. We assess the performance of three different assignment algorithms: a heuristic, the original Kuhn-Munkres algorithm (KMA), and the KMA adapted for GPU use. Each algorithm was tested while varying the number of UAVs, as well as the type of flight formation. During the experiments, we measured the calculation time, total distance travelled, and number of flight paths crossing. In terms of total distance travelled, the KMA always outperforms the heuristic. However, the KMA takes longer (orders of magnitude) to calculate the assignment. Realistically, the KMA algorithm can only be used as long as the swarm does not contain more than 500 UAVs. From that point the GPU version of the KMA is faster. We can conclude that, in most cases, it is recommendable to use the KMA for the assignment as it will reduce the distance travelled to a minimum and, consequently, also reduce the number of flight paths crossing.

Index Terms—UAV networks, Swarm takeoff, swarm formations, Kuhn-Munkres, GPU acceleration

Nomenclature

$\emptyset$	Empty set
P^i_{aerial}	Postion of UAV i in the air
P^i_{ground}	Postion of UAV i on the ground
CUDA	Compute Unified Device Architecture
GPU	Graphics processing unit
HPC	High perfromance computing
KMA	Kuhn-Munkres algorithm
n	Integer number, often the number of UAVs
O(..)	Big O notation

I. Introduction

Unmanned Aerial Vehicles (UAVs), more commonly known as drones, are now frequently used in many developed countries. Drones are used by the public for various entertainment applications, such as racing and aerial photography [1]. Also, the industry is using them, for applications such as: surveillance, thermal inspections, or in the film industry [2]. New applications in different fields arise rapidly. Especially intelligent transportation systems are benefiting tremendously from the rapid innovation in the UAV field as demonstrated in [3], [4] and [5]. Although many applications already exist, drone swarms bring new opportunities as they can collaboratively complete complex tasks with higher efficiency and lower cost, especially in harsh environments [6].

While drone swarm technology is advancing quickly, there are still many challenges remaining. Some of these challenges are of administrative nature (e.g., legislation issues), while others - communication among drones, providing resilience and safe takeoff/landing - require future research from a computational, algorithmic and communication perspective. One of the main challenges of large-scale autonomous UAV swarms is the takeoff procedure [7].

The takeoff problem can divided into two phases: (1) the assignment phase and (2) the takeoff procedure. In the assignment phase, it is decided which UAV on the ground will occupy a certain aerial position. At this first stage, three parameters should be considered: (i) the overall computation time, (ii) the total distance travelled by all UAVs, and (iii) the number of potential collisions between UAVs, i.e., flight paths that cross each other. Afterward, in the takeoff procedure phase (phase 2), the order in which the UAVs will take off is determined. One way to guarantee a collision free takeoff is to let the UAVs that have to fly the furthest start first, and proceed to take off each UAV sequentially, i.e., one-by-one. However, sequentially taking off each UAV takes a lot of time, and it grows with the number of UAVs. To reduce this ever-growing takeoff time, multiple UAVs must take off at the same moment (i.e., simultaneously). However, this can lead to collisions in many cases. Therefore, while calculating the swarm

position assignment (in phase 1), it is important that the number of potential collisions is taken into account. After all, if the number of collisions is minimized in phase 1, it will be easier to avoid all of them in phase 2. Since the number of possible collisions is strongly related to the total distance travelled (i.e. the fewer UAVs fly, the smaller the chances to collide), and since calculating all possible collisions for all possible assignments is very computationally expensive, we opted for minimizing the total distance travelled in the assignment phase.

The contributions of this paper can be summarized listed as follows:

1) We present three assignment algorithms: a heuristic, the sequential Kuhn-Munkres Algorithm (KMA), and the KMA adapted for GPUs to decrease the overall calculation time.
2) We evaluate the three assignment algorithms in terms of performance, total distance travelled by all UAVs, and the number of possible collisions between UAVs. During the evaluation, the ground and air formations are varied as well as the number of UAVs, with an emphasis on large-scale swarms, to assess the scalability of the solutions proposed.
3) Several GPU-based edge computing solutions are evaluated to figure out which platform can provide a real-time solution to the assignment problem for UAV swarms.

An important aspect of the UAV swarm takeoff process is the time needed to take off all UAVs safely. Individually, each UAV can take off rather quickly. However, when the number of UAVs grows, the time needed for all the UAVs to be deployed can become excessively long. This time is very valuable because (i) it depletes the (already limited) battery life, and (ii) during that time the UAVs are not used for their actual purpose (e.g. bridge inspections, search missions, etc.). As shown in [8], this problem is very prominent, and can impede the deployment of swarms as little as 25 UAVs. However, this problem can be avoided with intelligent takeoff procedures (phase2). This leads us to the importance of our current work; in order for those intelligent takeoff procedures to work optimally (and hence reduce the takeoff time) they need to be provided with (close to) optimal ground-to-air assignments. Hence, in this work, we evaluate different assignment algorithms while taking into consideration how these algorithms could be used in practice. In fact, some of these assignment algorithms could be easily executed on small on-board computers, whilst others would require onboard GPUs, or the use of dedicated servers.

This paper is organized as follows: in Section II we provide an overview of related works. In Section III, we explain in details the problem being addressed. In the following section (Section IV), we introduce the three assignment algorithms. Then, in Section V, we introduce our simulator called ArduSim, as well as the four GPU platforms being tested. We evaluate the three assignment algorithms in Section VI. Finally, in Section VII, we conclude our work, and raise directions for future work.

II. Related work

The amount of research focused on UAVs has increased steadily in the last decade. Nowadays, some research towards swarms of UAVs is also becoming popular. However, for practical reasons, performing real experiments with many UAVs is difficult. For that reason, many challenges related to real deployments remain untackled.

After carefully checking the related literature on the takeoff of UAV swarms, particularly of the VTOL type, we found no other related works than our own [9]. Nevertheless, we can find several works on the wider topic of UAV swarms.

The authors of [10] provided a nice example on how a swarm of UAVs can work together as a team. They were focused on a specific problem, namely path planning. Although path planning for UAVs has been extensively studied, their work was special because they restricted themselves to a low altitude urban environment with heterogeneous Global Navigation Satellite Systems (GNSS) coverage. As shown in their work, this would be impossible to achieve with a single UAV.

Mirzaeinia et al. [11] addressed the problem of the increasing number of drones in smart cities. This makes it more difficult to find the optimal station for each drone after it has completed its mission. When a drone finishes its mission, it is assigned a destination landing station to be recharged. Adverse weather situations can result in having the drone assigned to a station to take shelter. Their proposal uses the Kuhn-Munkres Algorithm (KMA) to match drone charging stations. In their study, three different scenarios were investigated. The first scenario used drones with the same energy level. The second scenario, used drones with a different energy level, and finally there is a third scenario where drones and stations had different energy levels. The results showed that a reduction in energy consumption of 30-90% can be achieved by applying this algorithm to drone station pairing, compared to an algorithm using a randomly preassigned station.

Eventually, we would prefer to execute all applications autonomously. However, we are not quite there yet, and thus in some cases it is still necessary that humans intervene. To that end the authors of [12], created a Layered Adjustable Autonomy (LAA) model for UAVs. In their solution, the agents (i.e. the drones) are given the choice to perform tasks autonomously, or with the help of humans. This adjustable level of autonomy can be updated internally by an agent, or externally by the pilot. They have tested their approach (indoors) with various tasks, and their results show that their model is able to manage human-agent interactions.

Futhermore, in our work, we also compare GPU solutions within UAVs. This is a trend that is gaining momentum, as shown in the following works. Most of the GPU-enabled UAVs use Jetson from Nvidia [13] as the main computing unit. An interesting work was provided by S. Mittal [14], where he presented a survey

of techniques for accelerating neural-network applications on the Jetson platform. These techniques can apply to other embedded systems too. The author compares them with the Jetson and other computing systems. He then proceeds by reviewing algorithm-level optimizations in the same style. Finally, there is an extensive section on various application areas such as agriculture [15], or even drone navigation [16].

Finally, the assignment problem in general, has been studied by mathematicians. They introduce the assignment problem as a combinatorial optimization problem, where there are a number of agents and a number of tasks to be performed. For each agent performing a task, there is a cost involved. The goal is to minimize the total cost. An assignment problem can be either balanced or unbalanced, depending on the number of agents and number of tasks. When the number of agents is equal to the number of tasks, we speak of a balanced assignment problem. For the specific case of UAV swarm assignment, the assignment problem will always be balanced. Currently, there are three options to solve this problem: Global methods, local methods, and heuristics. A global method promises to find the optimal solution. Currently, the Kuhn-Munkres algorithm (explained in section IV-B) is the fastest option available. In some specific cases, local methods can work better, however this is not guaranteed. Those algorithms, called auction algorithms, are used for example by Causa and Fasano [17] who worked on a strategic path planning for multi-UAVs, and by the authors of [18], who presented a dynamic task and resource assignment algorithm.

The last option is to use a metaheuristic. This will provide a suboptimal solution (in terms of reducing cost), but it is typically much faster than the previous solutions. In the literature, several metaheuristic algorithms have been applied to the assignment problem in general terms. Table I shows a comparison between those algorithms in different platorms; i.e. sequential, multicore and GPUs (we refer the reader to Kumar et al. [19] for insights). Results show the average execution time (of all their experiments) for various metaheuristics. They conclude that "for small size instances, the metaheuristic Tabu search shows the least average runtime on GPU, and Crow search algorithm has the least average runtime on CPU, while for larger size instances Crow search algorithm shows the least execution time in GPU." [19]. Although this comparison can give us a basic understanding about the execution time of various metaheuristics, they have not been used in the field of UAVs. Hence, Section ?? focuses on those algorithms that have been particularly designed for the drone swarm assignments as they provide metrics such as the number of flight paths crossing, calculation time overhead, and total distance travelled. We focus specifically on the applicability of the assignment algorithms for large-scale UAV swarms in real-world scenarios. To that end, we use a realistic multi-UAV emulator that relies on a software-in-the-loop approach, in order to really evaluate the practicality of these algorithms in the field of UAV swarms.

TABLE I
Comparison between various metaheuristics in terms of CPU and GPU time.

Name	Execution time CPU [s]	Execution time GPU [s]
Iterated Local Search	9.515	0.333
Simulated annealing	9.953	1.128
Genetic algorithm	5.102	0.669
Particle swarm optimization	17.526	0.841
Crow search algorithm	1.570	0.285
Tabu search	2.334	0.152

III. Problem statement

The final goal that we want to achieve is taking off a swarm quickly while ensuring that none of the UAVs collide during the process. As stated earlier, the entire takeoff process can be divided into two phases. In the first phase (the assignment), we must decide which UAV on the ground goes to which place in the air. In the second phase (the takeoff procedure), we can manipulate the order of the UAVs taking off; changing the order can speed up the entire takeoff process, but it must be ensured that the UAVs do not collide. The goal of this work is to find an assignment algorithm that calculates an assignment which minimizes the total distance travelled. Notice that this, in turn, will reduce the number of possible collisions, which will make phase II easier, and thus the overall takeoff time is reduced while avoiding all collisions.

A. The assignment problem (Phase I)

In order to take off all UAVs, we must first pre-assign each UAV on the ground a position in the air. Typically, when the UAVs are on the ground, they are placed at a random position, and usually close to each other. When the UAVs take off, they fly to a certain position in the air, which depends on the desired formation (e.g., linear, circle or matrix). Since it is usually not relevant which UAV occupies which position in the formation (homogeneous swarms), it is in our best interest to assign the positions in a manner such that the overall travelled distance by the UAVs is minimum. A graphical representation is given in Figure 1.

One solution might be to analyze all possible solutions (brute force), and then select the one where the distance is minimal. However, as shown in Figure 2, we can have many different solutions for a swarm of just four UAVs. As shown in our previous work [9], this brute force algorithm has a time complexity of $O(n! * n^2)$, and thus practical restrictions impede its use on swarms with more than 15 UAVs.

Hence, our next logical option is to reach a compromise, as we no longer search for the optimal solution, but one that comes close to it. Our heuristic, which is fully explained in Section IV-A, is able to provide a solution within $O(n^2)$. However, this compromise might affect the performance (i.e., total distance travelled and number of possible collisions) of the takeoff sequence. Fortunately,

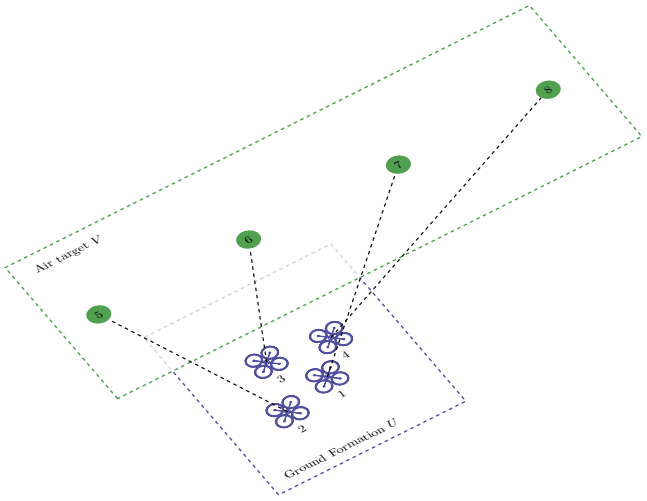


Fig. 1. Graphic presentation of a possible UAV swarm flight assignment.

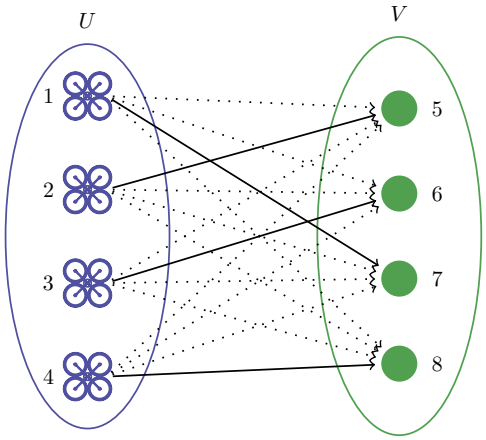


Fig. 2. All possible solutions that exist with only four UAVs.

the assignment problem was already studied in the mathematics field a long time ago in graph theory. In fact, the Kuhn-Munkres Algorithm (KMA) [20] (also referred to as the Hungarian algorithm) is able to reach the optimal solution within $O(n^3)$. In Section IV-B, we will explain how the algorithm works, and how we have implemented it in our ArduSim simulator. Later, in Section IV-C, we also describe how this algorithm can be adapted for GPU use.

IV. Assignment algorithms

A. Heuristic algorithm

The heuristic that we present here, offers a trade-off between calculation time and accuracy (i.e., total distance travelled). We first presented this heuristic in [9]. This algorithm was created to cope with the very slow calculation time of the brute-force algorithm. It can quickly generate an assignment that, for a low number of UAVs, slightly deviates from the optimal solution. The heuristic consists of determining a location on the ground, which is central to the UAVs deployed. Then, this central position is used to compute the distance towards all positions in the desired flight formation, which are then sorted in descending order. Using this sorted list, the UAV closer to each of these positions is then assigned to it. All details are explained in algorithm 1. This algorithm is able to calculate an assignment very fast ($O(n^2)$) because it reduces all the ground locations to a single point. However, since in general formations are symmetrical, the total distance travelled (see Equation 1) will only be slightly higher, compared to the theoretically optimal/brute-force solution. For further details, please refer to our previous work [9].

Algorithm 1 : Heuristic(numUAVs, groundLocations, flightFormation)

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

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

$$Total\ distance = \sum_{i=1}^{N} dist(P^i_{ground}, P^i_{aerial}) \qquad (1)$$
$$N \rightarrow total\ No.\ of\ UAVs.$$
$$dist \rightarrow euclidean\ distance$$

B. The Kuhn-Munkres algorithm (KMA)

The Kuhn-Munkres Algorithm (KMA) was developed by James Munkres in 1957. He based his work on an algorithm first developed by H. W. Kuhn [21], who in turn was inspired by two Hungarian mathematicians. Therefore, the algorithm is also known as the Hungarian or the Munkres algorithm. In the original problem, the authors were trying to match a set of n persons to a set of n jobs in the most cost-efficient way. Each person had a specific cost for a job; after some matrix operations on those costs (detailed in Algorithm 2) an optimal solution was guaranteed within $O(n^3)$. However, as one might imagine, the KMA can be used for many applications as long as a cost matrix is provided. In our work, the

cost matrix is an $n \times n$ matrix where n is equal to the number of UAVs, and the elements are calculated by the euclidean distance between a ground and an air location (see Algorithm 2).

Towards this end, the first step to be taken is to set a slack matrix of size numUAVs × numUAVs, and fill it out using function $distanceMatrixCalc()$. The distance matrix is calculated by using Algorithm 3, where function d returns the Euclidean distance. Then, the Subtract_row_minima() function is executed; it subtracts the smallest entry in each row from each entry in that row in the distance matrix; it is followed by Substract_column_minima(), which subtracts the smallest entry in each column from each entry in that column in the distance matrix. After minimum row and column values are subtracted, we draw lines (l) over rows and columns in order to cover all zeros in the matrix using the smallest number of lines possible (Cover_all_zeros() function). If the minimum number of covering lines is equal to $numUAVs$, then an optimal assignment of $zeros$ is possible, and the process is finished. If the minimum number of covering lines is less than $numUAVs$, an optimal assignment of $zeros$ is not yet possible. In that case, we need to determine the smallest entry not covered by any line. Afterward, we subtract this entry from each uncovered row, and then add it to each covered column, and finally run the Cover_all_zeros() function again.

Algorithm 2 Kuhn-Munkres CPU(numUAVs, groundLocations, flightFormation)

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

```
DistanceMatrix = distanceMatrixCalc(numUAVs, ∧
                 groundLocations, flightFormation, errorMatrix)
SLACK = distanceMatrixCalc()
Substract_row_minima(SLACK, ∧
                 DistanceMatrix)
Substract_column_minima(SLACK, ∧
                 DistanceMatrix)
while true do
    Cover_all_zeros(l,DistanceMatrix)
    if All_Columns_Covered then
        return                                  ▷ optimal found
    else
        Get_smallest_line(DistanceMatrix)
    end if
end while
```

Algorithm 3 distanceMatrixCalc(numUAVs, groundLocations, flightFormation, errorMatrix)

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

```
for i ∈ {0, ..., numUAVs} do
    for j ∈ {0, ..., numUAVs} do
        errorMatrix[i][j] ← d(groundLocations[i],
                                flightFormation[j])^2
    end for
end for
```

C. The Kuhn-Munkres adapted for GPUs

Several GPU implementations have been proposed in the literature for the KMA [22], [23], [24]. Moreover, they have been applied to different problems such as real-time image systems for multiple bee activity monitoring [25], or the subcarrier assignment problem [26]. To the best of our knowledge, our work introduces for the first time the application of KMA to drone location assignments at swarm takeoff. Particularly, our starting point is the NVIDIA CUDA [27] based proposal by Lopes et al. [28]. This solution is a general GPU adaptation to the linear assignment problem through the KMA, focusing on steps 2 to 6 (see Algorithm 4). Some modifications of this code have been made to adapt the GPU algorithm to the drone swarm problem. Firstly, the size of the distance matrix n is rounded up to the next power of 2; this allows the GPU to work more efficiently. Later, upon assigning the solution, we only take into account the results of the first k positions that match with the number of UAVs. Algorithm 4 summarizes the entire procedure, and we refer the reader to [24] for insights on the CUDA implementation.

Algorithm 4 : Kuhn-Munkres GPU(numUAVs, groundLocations, flightFormation)

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

Step 1 → Look for the maximum number of zeros in the cost matrix. CUDA Parallel reduction is applied at this step [29].

Step 2 → A initial assignment is made, identifying each zero as a possible assignment. In this case, the vector zeros and the cost matrix are used for this assignment.
Step 3 → The search structures are initialized to find alternative routes. When all columns are covered the algorithm ends.
Step 4 → Alternative routes are searched. Now, an optimization of the assignment is performed.
Step 5 → New alternative routes are proposed and the vectors are updated for a new evaluation in Step 3.
Step 6 → A CUDA reduction is applied to obtain the minimum value of the cost matrix. This minimum value is subtracted if the row and column are uncovered or added otherwise. Finally, it identifies and saves the elements of the slack matrix marked with 0. Then, return to Step 4.

The implementation is composed of several CUDA kernels. The first kernel performs an initial matching where each GPU thread will look for a zero of the slack matrix. Whenever a thread finds a zero, it fills this position out using $distanceMatrixCalc()$. This is performed atomically in order to avoid race conditions between threads (we refer the reader to [24] for insights). Moreover, two CUDA kernels are needed to perform the graph search initialization and termination testing due to global synchronization requirements. The former is for the initialization, and the later to perform the step itself. Once the matrix is initialized, the second kernel sets up a thread per item in the matrix which covers the column associated with it. Then, if there is a starred zero in that column, it counts the overall number of starred zeros using an atomic-add. Step 4 is implemented using a kernel with the

graph split in blocks processed by GPU thread-blocks (see [24]). The graph is split by distributing the zeros across the GPU blocks. It consists of repeatedly processing each zero as described by the KMA in parallel until all zeros are covered, or an alternative path is found. Step 5 basically applies the alternating paths, and it is carried out using two CUDA Kernels. The former performs a forward pass that builds the green vector, and the latter performs the reverse pass that applies the alternating path. Multiple paths may be visited at the same time, since the former kernel has a thread for each row, and the latter kernel has a thread for each column. Step 6 consists of two kernels as well: the former kernel for the add–subtract operation, and a final kernel to do the final reduction. After the reduction kernel, the minimum found is added to the elements that are covered by the rows and by the columns, and it is subtracted from the elements that are uncovered.

V. Hardware and software environment

In this section, we first detail the ArduSim simulator, which is used for evaluating the quality of the three assignment algorithms. Secondly, we shortly explain the various ground and air formations used during the experiments. Lastly, the details of the different GPU platforms, where the performance evaluation is carried out, are given.

A. ArduSim

The multi-UAV simulator/emulator ArduSim is available under the Apache License 2.0. This tool has many useful features, and it was formally presented in [30], and kept up to date in our GitHub project [31]. The most relevant characteristics of ArduSim are detailed below.

- Protocol deployment: ArduSim is designed to test protocols in a simulator, and then deploy the protocols quickly and reliably on real UAVs. To accomplish this, ArduSim (as a simulator) uses the same protocols and standards as real UAVs would use. In order to make the deployment straightforward, all this is abstracted inside the core of ArduSim. Deployment on real UAVs is done by adding a Raspberry Pi (a single board computer) to the UAV, and connecting it via a serial connection to the flight controller (in our case the open-source flight controller Pixhawk). When deploying the UAVs in a real experiment, the Raspberry Pi will use that serial connection to send and receive messages to/from the flight controller. Since nearly all UAV open-source controllers on the market use the MAVLink communications protocol [32], ArduSim-based developments can be deployed on flight controllers from various manufacturers.
- Scalability: ArduSim is designed to be a multi-UAV flight simulator. Therefore, it can scale to a large number of UAVs. The only limitation is the hardware and protocol used in the simulator.
- UAV-to-UAV communication: ArduSim uses the wireless standard 802.11a, in ad-hoc mode, to communicate, both between UAVs themselves, and between UAVs and the ground station. When ArduSim is used as a simulator, communication is accomplished with virtual links. Whenever protocols are thoroughly tested, they can be deployed on real UAVs. In this case ArduSim will send the messages via User Datagram Protocol (UDP) broadcasts.
- API: Many different protocols need similar functions: taking off, landing, communicating between UAVs, etc. Ardusim gives access to these through an Application Programming Interface (API) in order to achieve faster protocol development.
- Data logging: ArduSim extensively logs data in various formats after a UAV flight, making development and debugging user friendly.

B. UAV formations used

In our study, we noted that swarm formations can have an influence on the performance of the assignment algorithms. Hence, the three most common UAV formations are evaluated; i.e., Circular, Linear, and Matrix formations. At the start of an experiment, the UAVs are placed on the ground in a random manner. We placed them in a square with sides of 400 meters, and with a minimum distance of 7.5 meters between all the UAVs. The UAVs then have to go towards their locations in the flight formation (Circular, Linear, or Matrix) at an altitude of 50 meters. Figure 3 shows the three swarm formations used in our experiments. Their characteristics are described below.

- Circular formation: Given the number of UAVs, a circumference is drawn with the smallest possible radius so that each drone is placed along the periphery, and a minimum euclidean distance of 50 meters is established between two neighboring UAVs.
- Linear formation: Starting from the center, new UAVs are placed along the line every 50 meters, and on both directions, to keep the formation symmetrical.
- Matrix formation: To obtain the matrix shape independently of the number of drones in the formation, each drone is added alternately between the quadrants of two perpendicular axes and, when the square is completed, each axis is extended by 50 meters to create a larger square; the process continues until all the UAVs are assigned a coordinate.

C. GPU platform

We focused on four different architectures to assess the performance of the proposed solutions (see Table II): a High Performance Computing (HPC) node called Pedra, and three low-power edge computing devices from the NVIDIA Jetson family (i.e., Jetson nano, Jetson TX2, and Jetson AGX Xavier). Although Pedra cannot be mounted directly on UAVs (due to its weight, size, and energy consumption), it could be used via a cloud solution when the mobile internet speed and coverage are sufficient. The main purpose of this comparison is to determine whether the Kuhn-Munkres Algorithm (KMA) adapted

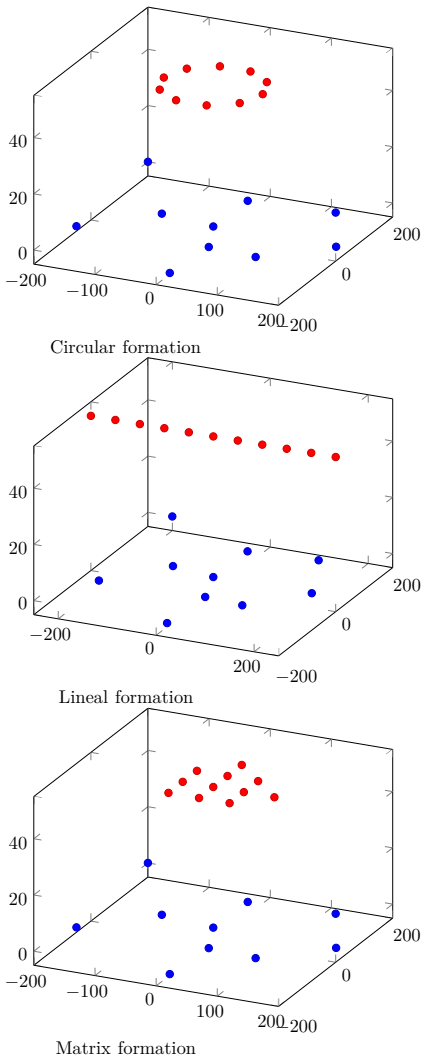


Fig. 3. UAVs formations for the experiment.

for GPUs reduces the calculation time, and whether it can be performed on the edge in a reasonable time frame and, if so, which platform is the most suitable.

VI. Evaluation

This section presents a detailed evaluation of the assignment algorithms on the different targeted architectures. We evaluated the assignment algorithms under three different criteria: (i) the total distance travelled, (ii) the number of possible collisions, and (iii) the calculation time. Since the obtained assignment is the same in both the original Kuhn-Munkres Algorithm (KMA) and the GPU-adapted KMA, the results for the first two criteria (i.e. total distance and number of possible collisions) will be the same. Therefore, in the first two experiments, just the original KMA will be mentioned. In the last experiment, however, we do distinguish between the two KMA versions.

A. Total distance travelled

In our first experiment, we evaluate the difference in the total distance travelled by all UAVs. The original work of the KMA [20] contains the proof that the KMA will always return the minimal cost (in our case, the minimal total distance travelled). This means that our heuristic will always have a higher total distance travelled (in rare cases it might be equal). Nevertheless, to evaluate the effectiveness of our heuristic, it is interesting to compare both distances. Therefore, we conducted the following experiment: we placed a number of UAVs on the ground in a random way, and calculated the assignment using the heuristic as well as the KMA. We experimented with three different air formations (Circle, Linear, Matrix), and also varied the number of UAVs; the results are shown in Figure 4. As we can see in the figures, and as expected, the total distance travelled is somewhat higher when using our heuristic. As shown on the zoomed-in view, the difference between the two algorithms remains relatively constant. On average, this extra distance is of 2879 m, 2378 m, 2457 m for the circle, linear, and matrix formation, respectively. Although this might seem a lot, one must remember that this is the total distance for all UAVs (on average 1000 UAVs). Nevertheless, we can see that there is a substantial difference between using the KMA and our heuristic, and w.r.t the total distance travelled, the KMA clearly outperforms the heuristic. Furthermore, we can also observe that the total distance travelled is different for each formation. This is because some formations, such as the matrix formation, are a lot more compact (i.e. more UAVs fit in the same area) than others (linear).

B. Number of possible collisions

For our second experiment, we measured the number of possible collisions. During the flight, we measured the distances between all the UAVs constantly; whenever two UAVs are closer than 5 meters (typical GPS error) a collision counter is increased. Both UAVs are also allowed to continue their flight, and might virtually collide with other UAVs at another time instance. Again, we experimented with three formations and a variable number of UAVs. The results are shown in Figures 5 and 6. As shown, the KMA performs better for all formations. In some cases, such as the circle and the matrix formation, it nearly avoids all collisions. In the linear formation it is still performing better than the heuristic; however, the number of collisions are still quite high. Due to its one dimensional character, there are more possible collisions for the linear formation.

C. Computation time

Up to now, the KMA has shown to clearly perform better than our heuristic. However, as expected from the pseudocode given before (see Section IV), the computation time for the KMA will be higher. In order to know how large the difference actually is, we performed various

TABLE II
Specification of the various GPU platforms used in our experiments.

Platform	Pedra	Jetson AGX Xavier	Jetson TX2	Jetson Nano
CPU	Intel Silver 4216	NVIDIA Carmel ARM v8.2	ARMv8	ARM Cortex-A57 MPcore
GPU (NVIDIA)	GeForce RTX 2080 Ti	Volta	Pascal	Maxwell
Memory [Gb]	376 DDR4	32 LPDDR4x	8 LPDDR4	4 LPDDR4
Size [mm]	N/A	105 x 105	50 x 87	70 x 45
Weight [g]	N/A	280	85	61
Energy consumption [W]	N/A	10-30	7.5	3-5

experiments (again with three formations, and varying the number of UAVs).

As shown in Figure 7, our heuristic is able to generate a drone assignment much faster than the KMA. This figure shows that our heuristic can always be used withing a reasonable time. Nevertheless, the same cannot be said for the KMA. Here we see that, for 750 UAVs, the calculation time is close to 16 minutes, which is definitely too much for any practical application. In this figure we also included the brute force method as a reference. Of course, compared to the brute force approach, the KMA (which returns the same answer) is a lot faster.

Similarly to previous experiments, we find that the actual formation adopted has a significant influence. As shown in Figures 8 and 9, when the heuristic approach is used, the calculation time is independent of the formation itself. This cannot be said when using the KMA. The KMA performs notably worse when the UAVs are spread out over one dimension (linear formation). Yet, whenever the formation is more uniform, over the two dimensions, the performance is improved.

As shown in the previous experiments, the KMA performs better in terms of total distance travelled, and number of possible collisions. However, we can only use it up to 500 UAVs due to time constrains. Fortunately, the KMA is parallelizable to some extent, and thus we can decrease the calculation time by using GPUs. We compared the CPU version (the one which we used in all previous experiments) with a GPU version for four different platforms.

All platforms targeted (see Section V-C) have a CPU and GPU on board. The KMA performance on each platform depends on two main factors: (a) the drone swarm formation used, and (b) the number of drones in the swarm. We can observe (from Figures: 10, 11, and 12) that, for all formations, and for all platforms, the calculation time for CPUs grows quickly with the number of UAVs. In most cases, the GPU platform outperforms the CPU (see Table III). However, the GPU needs a large computational and parallel workload (i.e. many drones in the swarm) to outperform its CPU counterpart, and, therefore, for swarms with less than 500 UAVs, the CPU often performs better.

TABLE III
Peak performance difference of CPU vs GPU.

	Pedra	AGX Xavier	Jetson TX2	Jetson Nano
Speed-up	40x	62x	62x	45x

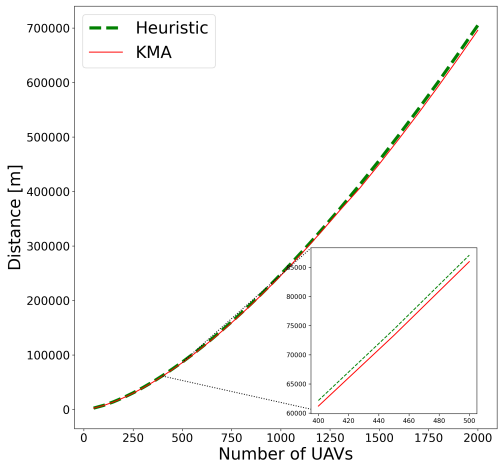
Furthermore, we can observe a great difference between the various formations, especially in the case of the linear formation, where calculations take 10 times longer than for the circular formation. In the KMA, steps 1, 2, and 3 are fully parallel, whereas steps 4, 5, and 6 require a CUDA reduction, which is less appropriate for a GPU. Therefore, the execution time will depend highly on the number of times each step is executed. From Table IV, this percentage is not equal for each formation. The more time is spent in steps 1, 2, and 3, the higher the speed-up. Since the Matrix formation is the only formation where more than half of the time is spent in the parallel part, it benefits from a high speed-up.

TABLE IV
KMA profiling on GPU. Percentage of each algorithm step performed on the overall execution time for each formation targeted.

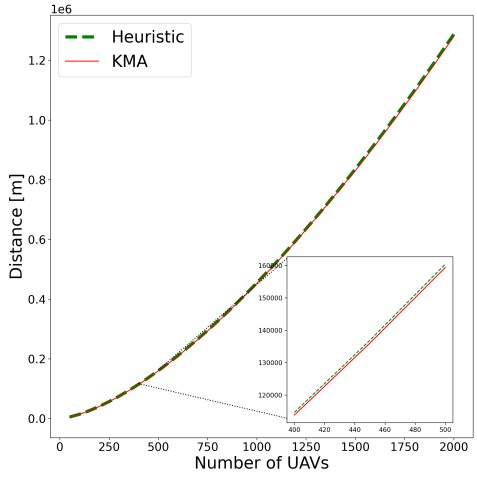
Steps performed(%)	Circle	Linear	Matrix
Steps 1, 2, and 3	42 %	31 %	59 %
Steps 4, 5, and 6	58 %	69 %	41 %

Concerning the performance differences between different GPUs, the HPC node Pedra offers the highest performance. However, communication with any external infrastructure (e.g. GPU-based cloud) is not always possible, and thus we investigated different edge computing platforms (AGX Xavier, Jetson TX2, and Jetson Nano). In order to make a comparison between them, we take the HPC platform Pedra as a baseline. Results (shown in Figures 10, 11, and 12) indicate that the AGX Xavier is up to 1.6× slower; the Jetson TX2 is up to 4.6× slower; and the Jetson Nano is up to 5.6× slower. These results are reported for the worst-case scenario, i.e., for a swarm of 2000 UAVs.

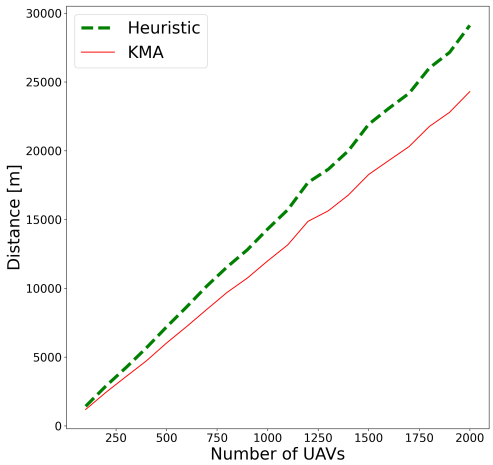
Figure 13 shows the box-and-whisker plot of the speed-up factor obtained by the GPU included in the HPC-node, and for the low-power GPUs included in edge computing devices. The HPC-node outperforms the low-power GPUs in general terms; yet, there are some cases where low-power GPUs outperform the HPC node. This is particularly true in the AGX Xavier, but it also happens in Jetson TX2 and Nano. When there are fewer drones



(a)



(b)



(c)

Fig. 4. The total distance travelled by all UAVs using the heuristic and KMA for various formations: (a) Circle formation; (b) Linear formation, and (c) Matrix formation.

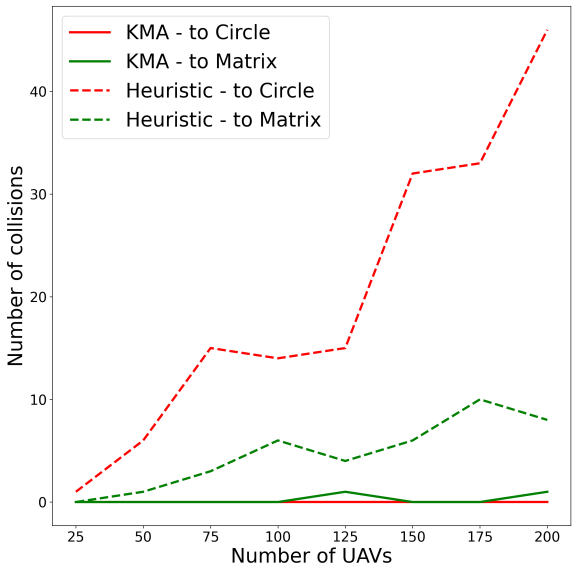


Fig. 5. A comparison of the heuristic vs. the KMA algorithm in terms of potential collisions when varying number of UAVs (Circle and Matrix formations).

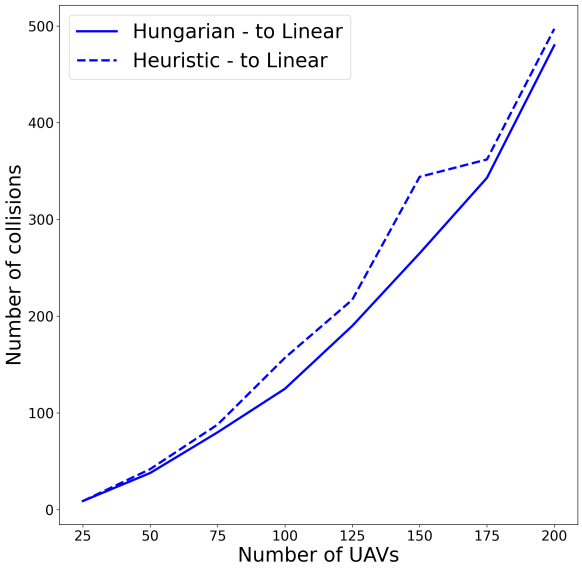


Fig. 6. A comparison of the heuristic vs. the KMA algorithm in terms of potential collisions when varying number of UAVs (Linear formation).

in the swarm, the computational resources of the HPC-GPU are not sufficiently occupied. In fact, there are scenarios (e.g. those with very few drones) where only 1% of the GPU resources are utilized. Low-power GPUs take more advantage of these resources in configurations with very few drones, being even more efficient than high-performance GPUs. However, as the computational workload increases, the computational differences become more pronounced for HPC environments.

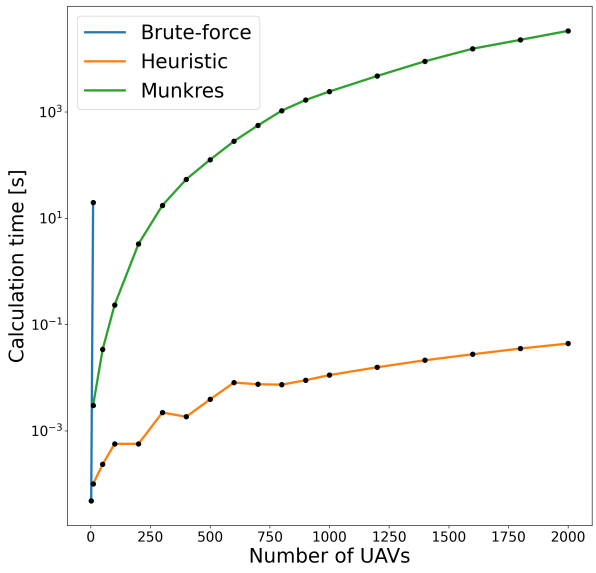


Fig. 7. Average calculation time for all UAV formations and assignment algorithms.

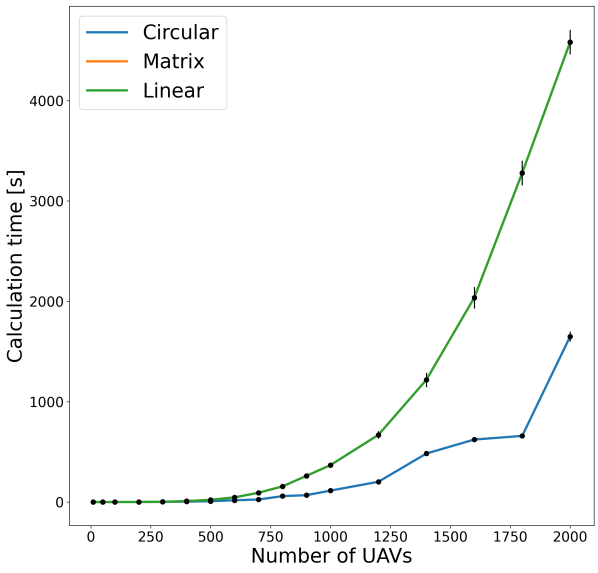


Fig. 9. Calculation time using the Kuhn-Munkres algorithm for various UAV formations.

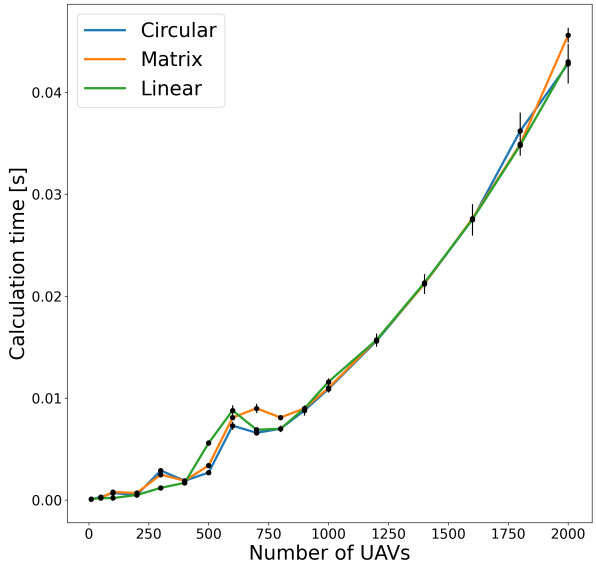


Fig. 8. Calculation time using the heuristic algorithm for various UAV formations.

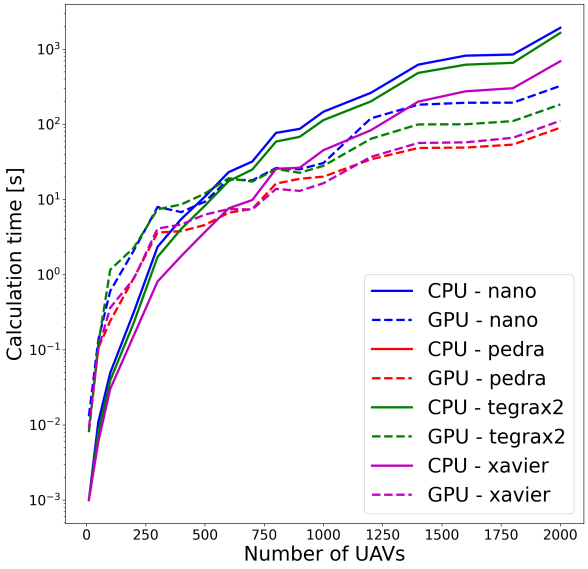


Fig. 10. CPU vs GPU: calculation time for the circular UAV formation.

VII. Conclusion and future work

An important, but not yet widely addressed problem, is the UAV swarm takeoff problem. For each UAV on the ground, a position in the air has to be assigned. The assignment should keep both the overall distance travelled by the UAVs, and the number of possible collisions (i.e. flight paths crossing), to a minimum.

The main contributions of this work are: (i) we presented three assignment algorithms, (ii) we tested these algorithms considering many parameters and different metrics; and (iii) we tested the different algorithms on several GPU-based edge computing solutions.

From our experiments, we conclude the following: first, we have seen that the total distance travelled by all UAVs is lower when the Kuhn-Munkres Algorithm (KMA) is used. The actual differences depend on the specific formation, but only to some extent. Secondly, we could also observe that the number of possible collisions is reduced when the KMA is used. This effect was most prominent in the matrix and circle formation, and less so in the linear formation. Thirdly, the calculation time of the KMA is a lot longer, and the normal CPU version can only be used up to (at most) 500 UAVs for any practical use case. Fourthly, the GPU version allows us to speed

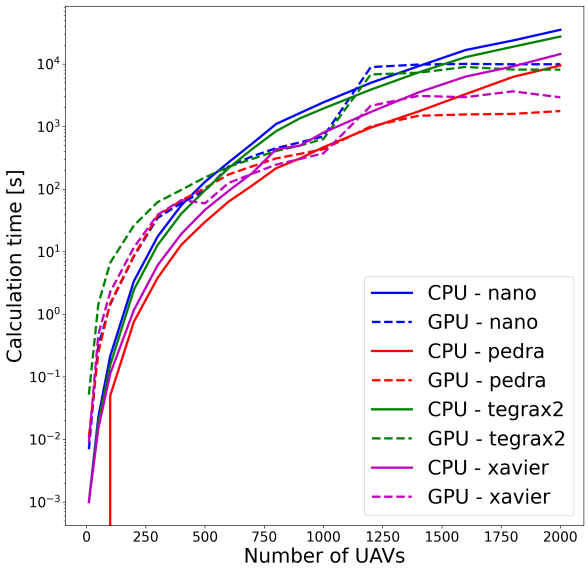


Fig. 11. CPU vs GPU: calculation time for the linear UAV formation.

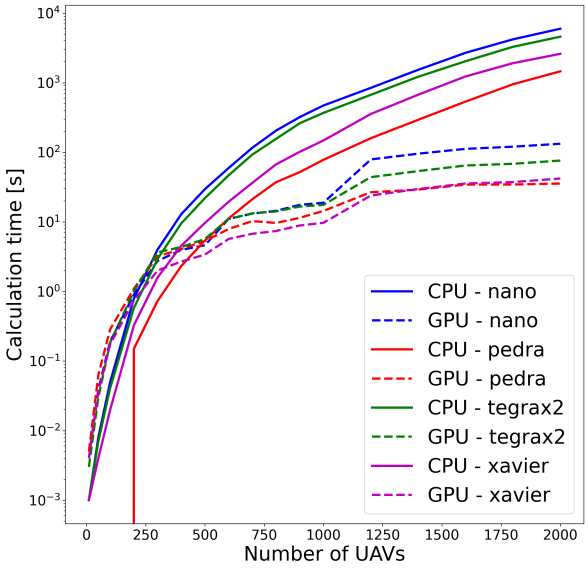


Fig. 12. CPU vs GPU: calculation time for the Matrix UAV formation.

up the calculation time. A speed-up is only realized if the number of UAVs is sufficiently high; in most cases this was of about 500 UAVs. The speed-up provided by the GPUs ranges between 5× and 62×, depending on: the number of UAVs, the flight formation adopted (Linear, Circle, Matrix), and the GPU-architecture. Nevertheless, even when using GPUs, our heuristic will still calculate the assignment faster.

With this work, we have demonstrated that the KMA is the most suitable algorithm to calculate the assignment. If, however, the application requires a high number of UAVs in a swarm, the GPU version of the KMA can decrease the calculation time. If, for some specific reason, the calculation time still exceeds the limits of an application, then our heuristic exists. However, when using the heuristic, a higher price in terms of total distance travelled and number of possible collisions needs to be paid.

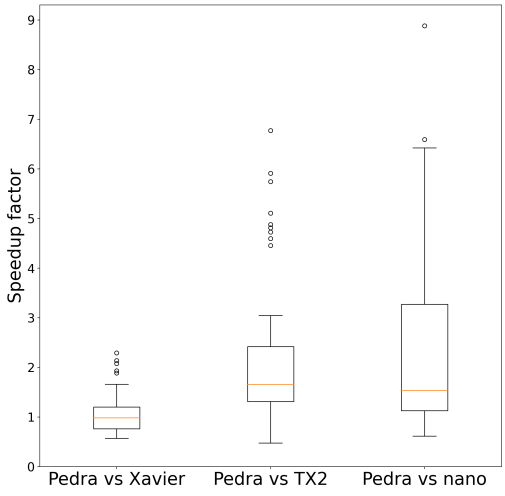


Fig. 13. GPU platforms comparison: speed-up factors for the low-power GPUs compared to the GPU included on HPC node PEDRA.

The takeoff problem can be divided into two phases: (1) the assignment phase, and (2) the takeoff procedure. In this work, we focused on the first phase while in previous works such as [33], [34], we have shown that we can achieve a collision-free semi-simultaneous takeoff procedure using the assignment as detailed in this work. Hence, our future work is twofold: first, we will explore other algorithms such as metaheurstics that have shown an increasing interest in the last decade for the graph assignment problem; second, we plan to optimize our collision-free semi-simultaneous takeoff procedure to further reduce the takeoff time.

Acknowledgments

This work is derived from R&D projects PID2021-122580NB-I00 and RTC2019-007159-5, as well as the Ramon y Cajal Grant RYC2018-025580-I, funded by MCIN/AEI/10.13039/501100011033 and “ERDF A way of making Europe”.

References

[1] S. J. Kim, Y. Jeong, S. Park, K. Ryu, and G. Oh, “A survey of drone use for entertainment and avr (augmented and virtual reality),” in Augmented Reality and Virtual Reality, pp. 339–352, Springer, 2018.

[2] F. Giones and A. Brem, “From toys to tools: The co-evolution of technological and entrepreneurial developments in the drone industry,” Business Horizons, vol. 60, no. 6, pp. 875–884, 2017.

[3] X. Li, J. Tan, A. Liu, P. Vijayakumar, N. Kumar, and M. Alazab, “A novel uav-enabled data collection scheme for intelligent transportation system through uav speed control,” IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 4, pp. 2100–2110, 2021.

[4] Y. A. Nijsure, G. Kaddoum, N. Khaddaj Mallat, G. Gagnon, and F. Gagnon, “Cognitive chaotic uwb-mimo detect-avoid radar for autonomous uav navigation,” IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 11, pp. 3121–3131, 2016.

[5] Y. Lin and S. Saripalli, "Sampling-based path planning for uav collision avoidance," IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 11, pp. 3179–3192, 2017.

[6] W. Chen, J. Liu, H. Guo, and N. Kato, "Toward robust and intelligent drone swarm: Challenges and future directions," IEEE Network, vol. 34, no. 4, pp. 278–283, 2020.

[7] A. Tahir, J. Böling, M.-H. Haghbayan, H. T. Toivonen, and J. Plosila, "Swarms of unmanned aerial vehicles — a survey," Journal of Industrial Information Integration, vol. 16, p. 100106, 2019.

[8] C. Sastre, J. Wubben, C. T. Calafate, J.-C. Cano, and P. Manzoni, "Safe and efficient take-off of vtol uav swarms," Electronics, vol. 11, no. 7, 2022.

[9] F. Fabra, J. Wubben, C. T. Calafate, J. C. Cano, and P. Manzoni, "Efficient and coordinated vertical takeoff of uav swarms," in 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), pp. 1–5, 2020.

[10] F. Causa, G. Fasano, and M. Grassi, "Multi-uav path planning for autonomous missions in mixed gnss coverage scenarios," Sensors, vol. 18, p. 4188, 11 2018.

[11] A. Mirzaeinia, S. Bradley, and M. Hassanalian, "Drone-station matching in smart cities through hungarian algorithm: power minimization and management," in AIAA Propulsion and Energy 2019 Forum, p. 4151, 2019.

[12] S. Mostafa, M. Ahmad, A. Mustapha, and M. Mohammed, "Formulating layered adjustable autonomy for unmanned aerial vehicles," International Journal of Intelligent Computing and Cybernetics, vol. 10, pp. 00–00, 10 2017.

[13] Nvidia, "Nvidia jetson solutions for drones and UAVs ." https://www.nvidia.com/pt-br/autonomous-machines/uavs-drones-technology/, 2021. Accessed: 2021-03-03.

[14] S. Mittal, "A survey on optimized implementation of deep learning models on the nvidia jetson platform," Journal of Systems Architecture, vol. 97, pp. 428–442, 2019.

[15] M. A. Guillén, A. Llanes, B. Imbernón, R. Martínez-España, A. Bueno-Crespo, J.-C. Cano, and J. M. Cecilia, "Performance evaluation of edge-computing platforms for the prediction of low temperatures in agriculture using deep learning," The Journal of Supercomputing, vol. 77, no. 1, pp. 818–840, 2021.

[16] D. Hernández, J.-C. Cano, F. Silla, C. T. Calafate, and J. M. Cecilia, "Ai-enabled autonomous drones for fast climate change crisis assessment," IEEE Internet of Things Journal, 2021.

[17] F. Causa and G. Fasano, "Multiple uavs trajectory generation and waypoint assignment in urban environment based on dop maps," Aerospace Science and Technology, vol. 110, p. 106507, 2021.

[18] X. Fu, P. Feng, and X. Gao, "Swarm uavs task and resource dynamic assignment algorithm based on task sequence mechanism," IEEE Access, vol. 7, pp. 41090–41100, 2019.

[19] M. Kumar, A. Sahu, and P. Mitra, "A comparison of different metaheuristics for the quadratic assignment problem in accelerated systems," Applied Soft Computing, vol. 100, p. 106927, 2021.

[20] J. Munkres, "Algorithms for the assignment and transportation problems," Journal of the society for industrial and applied mathematics, vol. 5, no. 1, pp. 32–38, 1957.

[21] H. W. Kuhn, "The hungarian method for the assignment problem," Naval research logistics quarterly, vol. 2, no. 1-2, pp. 83–97, 1955.

[22] C. N. Vasconcelos and B. Rosenhahn, "Bipartite graph matching computation on gpu," in International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition, pp. 42–55, Springer, 2009.

[23] K. Date and R. Nagi, "Gpu-accelerated hungarian algorithms for the linear assignment problem," Parallel Computing, vol. 57, pp. 52–72, 2016.

[24] P. A. Lopes, S. S. Yadav, A. Ilic, and S. K. Patra, "Fast block distributed cuda implementation of the hungarian algorithm," Journal of Parallel and Distributed Computing, vol. 130, pp. 50–62, 2019.

[25] T. N. Ngo, K.-C. Wu, E.-C. Yang, and T.-T. Lin, "A real-time imaging system for multiple honey bee tracking and activity monitoring," Computers and Electronics in Agriculture, vol. 163, p. 104841, 2019.

[26] S. S. Yadav, P. A. C. Lopes, A. Ilic, and S. K. Patra, "Hungarian algorithm for subcarrier assignment problem using gpu and cuda," International Journal of Communication Systems, vol. 32, no. 4, p. e3884, 2019.

[27] NVIDIA, P. Vingelmann, and F. H. Fitzek, "Cuda, release: 10.2.89," 2020.

[28] P. Lopes, S. Yadav, A. Ilic, and S. Patra, "HungarianGPU on GitHub." https://github.com/paclopes/HungarianGPU, 04 2019. Accessed: 2021-03-03.

[29] M. Harris et al., "Optimizing parallel reduction in cuda," Nvidia developer technology, vol. 2, no. 4, pp. 1–39, 2007.

[30] F. Fabra, C. T. Calafate, J.-C. Cano, and P. Manzoni, "ArduSim: Accurate and real-time multicopter simulation," Simulation Modelling Practice and Theory, vol. 87, pp. 170–190, sep 2018.

[31] GRC, "ArduSim: a novel real-time flight simulator." https://github.com/GRCDEV/ArduSim, 2021. Accessed: 2021-03-23.

[32] L. Meier and QGroundControl, "MAVLink Micro Air Vehicle Communication Protocol." https://mavlink.io/en/, 2007. Accessed: 2019-05-11.

[33] C. Sastre, J. Wubben, C. T. Calafate, J.-C. Cano, and P. Manzoni, "Safe and efficient take-off of vtol uav swarms," Electronics, vol. 11, no. 7, 2022.

[34] J. Wubben, C. Sastre, C. T. Calafate, J.-C. Cano, and P. Manzoni, "Collision-free swarm take-off based on trajectory analysis and uav groupings," in 23rd IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2022), 06 2022.