MGRAPH: A Multi-Graph Homography Method to Generate Incremental Mosaics in Real-Time from UAV Swarms

J. J. Ruiz[1], F. Caballero[1] and L. Merino[1]

***Abstract*—During the last years, UAVs have proved to be an essential tool in the mapping industry. Furthermore, the next generation of UAVs is envisioned to work cooperatively, following the swarming/teaming concept. This article presents MGRAPH, a novel approach to generate an incremental mosaic in real-time from an UAV swarm. The algorithm is based on the generation and fusion of multiples sub-mosaics represented by homography graphs. Implementation takes advantage of parallel processing and UAV metadata to generate a georeferenced mosaic over a Geographic Information System (GIS). A comparison with other mosaicking pipelines on three datasets demonstrates that MGRAPH could be an alternative to address the generation of mosaics from a swarm of UAVs.**

***Index Terms*—Aerial Systems: Perception and Autonomy, Mapping, Multi-Robot Systems.**

I. Introduction

In the following years, UAVs are foreseen to operate in dynamic, non-hostile environments where initial conditions will change rapidly, forcing the system to react in consequence. The swarming paradigm implies a higher level of autonomy, so human intervention is supposed to decrease. Nonetheless, the added value of this topology consists in the possibility of obtaining huge amounts of information from sensors onboard in less time. Processing and fusing this information as soon as possible may lead to a vital advantage in future environments [1].

Particularly, cameras on-board the UAVs offer a limited Field-of-View (FoV) of the scene. Image mosaics have been widely addressed during the last decade to enhance this limited FoV, and offer a solid alternative to traditional mapping in domains such as agriculture, mining or construction [2], [3]. Furthermore, recent research on mosaicking has made great strides in improving the performance of its algorithms [4], [5], [6], [7], [8], [9]. For example, in the research carried out by [4] and [5], the presented mosaicking algorithms were evaluated in large scale environments. However, according to the authors, working with these scenarios requires to downscale the original datasets due to memory limitations. Further research on real-time mosaicking is presented in [6], [7]. In this case, authors work with highly overlapped datasets to create the final mosaic. Nonetheless, they do not generate georreferenced mosaics as an output. Recent research closer to the MGRAPH scope is analyzed separately. In the work by [8] (BIMOS), seamless mosaics are generated adopting a multi-threading strategy, minimizing the time spent in each process. Although BIMOS can generate incremental results, it does not use external metadata so the algorithm is not suitable for images from a multi-vehicle system. Additionally, [9] presents a Structure from Motion (SfM) algorithm that works with large datasets in a reasonable amount of time. The main drawbacks of this approach are that it requires a SLAM system and it is not possible to work with unconnected mosaics.

Reaching real-time performance requires the parallelization of parts of the algorithm [10]. As a result, different companies such as Pix4D[1] or PrecissionHawk[2] started to offer mapping services based on a cloud computing architecture. However, externalizing the mosaicking process to the cloud requires Internet access.

Cooperative mapping using an UAV swarm could be considered novel research and imposes different constraints. Firstly, the mosaicking algorithm should work with images coming from different heterogeneous sources. Secondly, the system should be robust enough to work with unconnected mosaics and provide an incremental output. Finally, the system should generate mosaics in real-time.

One of the main advantages of using an UAV swarm is the possibility to cover an area faster than the single UAV approach. Generating a real-time view of an area covered by an UAV swarm have potential applications in time-constrained fsituations such as natural disasters or military applications. Nonetheless, little attention has been paid to this scenario in current research.

In this paper, a novel mosaicking pipeline called MGRAPH (**M**ulti-**GRAP**h **H**omography-based method) is presented. The aim of this algorithm is to generate incremental mosaics in real-time from a swarm of UAVs. To this end, the mosaicking pipeline has been split into different sub-processes, most of them running on the GPU side. An important contribution of this research is the generation and fusion of different sub-mosaics, represented in the system as graphs. Due to this

Manuscript received: February, 24, 2018; Revised April, 2, 2018; Accepted May, 19, 2018.

This paper was recommended for publication by Editor Jonathan Roberts upon evaluation of the Associate Editor and Reviewers' comments. *The work of F.C. and L.M. was partially supported by MINECO (Spain) grant OCELLIMAV (TEC-61708-EXP).

[1]Juan Jesus Ruiz, Fernando Caballero and Luis Merino are with School of Engineering, Universidad Pablo de Olavide, Crta. Utrera km 1, Seville, Spain jjruipav@alu.upo.es, fcaballero@upo.es, lmercab@upo.es

Digital Object Identifier (DOI): see top of this page.

[1]https://cloud.pix4d.com/login

[2]http://www.precisionhawk.com/es/precisionmapper

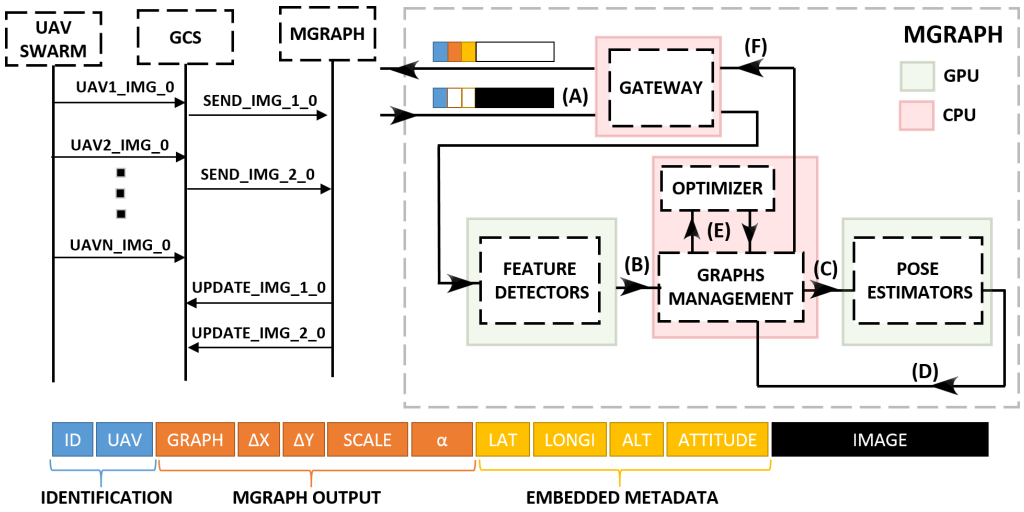

Fig. 1. (Top-Left) Information exchange between the UAV swarm, the GCS and MGRAPH. (Top-Right) General algorithm work-flow: (A) Metadata extraction. (B) Feature detection. (C-D) Graph registration. (E) Non-linear optimization. (F) Mosaicking update. (Bottom) Resulting datagram from MGRAPH: image identification (blue), absolute translation in pixels (tx,ty), scale factor and relative angle in degrees (α) (orange), geographic information (yellow) and raw image (black).

characteristic, MGRAPH can work with datasets independently of the order the images are injected into the system, making this algorithm suitable to work with an UAV swarm. Finally, MGRAPH does not generate mosaics as a single file but represents the final result as an overlay over a GIS. This approach allows to minimize the processing time and georeference different mosaics in a global context.

The remainder of this article is organized as follows: Section II describes briefly the MGRAPH workflow. In Section III, a more in-depth explanation about the algorithm is made. Section IV details the mosaic generation. Section V shows the experimental validation of the proposed method while Section VI presents comparative results with respect to other methods in the state of the art. Finally, Section VII summarizes the result of this research and draws conclusions.

II. System Overview

MGRAPH is composed by two separate processes: an external visualizer usually integrated in the Ground Control Station (GCS) and the mosaicking algorithm. As shown in Fig. 1, a datagram containing identification parameters and the image data is sent to MGRAPH. In this process, MGRAPH extracts the embedded metadata containing information from the UAV and computes the graph where the image has been inserted and its position within the graph. This information is used by the GCS in order to update the position of the image in the mosaic.

The current implementation of MGRAPH takes advantage of the separation in both CPU and GPU sub-processes so they can be hosted in different machines. This separation allows to split the computation not only into processes but also in hardware, offering a scalable solution if different UAV swarm sizes are considered. For instance, several instances of feature detectors and pose estimators can be executed in parallel in order to balance the algorithm workload.

Although a more in-depth explanation is made in following sections, a brief description for each phase in MGRAPH is introduced according to Fig. 1. First (A), information from the GCS is extracted in order to initialize the arriving picture with the embedded geographic information. Then, the gateway sends the image to an idle feature detector that extracts the main image features using a binary descriptor (B). After that, the graphs management process determines if the new image matches an existing graph or not. To this end, different pose estimator processes are used in parallel to estimate feature matches between images (C) and to compute an initial homography estimation (D). With this information, the current list of graphs is updated and a non-linear optimization is performed (E). Finally, MGRAPH returns to the GCS information regarding the mosaic output.

III. The MGRAPH Algorithm

A. Graphs and motion model

In the context of this research, each mosaic is represented by a graph, denoted as $g = (V, E)$, where V is a set of vertices and E is a set of edges, formed by pairs of vertices. Vertices on graphs represent images in the mosaic. An edge $e = \{i, j\}$ has as endpoints vertices i and j, and is created if there exists a relative homography transformation $\mathbf{H}^{i,j}$ between the two images.

We will consider that $\mathbf{H}^{i,j}$ represents a relative homography from vertex j to vertex i. In the case of the absolute homography of the same vertex i, the second parameter will refer to the *reference vertex* as $\mathbf{H}^{i,R}$. Furthermore, we restrict these homographies to similarity transformations, defined as follows:

$$\mathbf{H}^{i,j} = \begin{pmatrix} scos\alpha & -ssin\alpha & \Delta x \\ ssin\alpha & scos\alpha & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

where (Δx,Δy) represents a relative translation in pixels, α the relative rotation in radians and s the scale factor in both axis.

The optimal movement of an ortho-mapping camera on-board an UAV can be decomposed in a two axis translation and one axis rotation. Therefore, the motion model $\mathbf{H}^{i,j}$ is a 4-DoF transformation, as commonly used in previous research [8].

The validity of this approach considers a planar geometry, where the altitude of the UAV is large when compared with the variations in height of the map. Additionally, the camera is assumed to be gyrostablized in UAV ortho-mapping approaches, so that roll and pitch angle can be considered negligible, reinforcing our choice of a 4-DoF model for $\mathbf{H}^{i,j}$.

Each graph has a *reference vertex* (R) associated to an identity transformation ($\mathbf{H}^R = I$). Any other vertex in a graph can be related to its reference vertex by means of an absolute homography, that can be obtained by composition:

$$\mathbf{H}^{i,R} = \mathbf{H}^{i,j} \cdot \mathbf{H}^{j,R} \tag{2}$$

Notice that a given vertex might be connected to several vertices through different edges. As it will be explained in following sections, the *strong edge* is used to compute the absolute homography, while the rest of edges (defined as *weak edges*) are only taken into account in the optimization problem. Therefore, the resulting oriented graph contains at least one edge indicating the path to the *reference vertex* and different *weak edges* showing neighborhood relationship. The orientation of the path represents the unique way to calculate the absolute homography.

B. Image reception and feature extraction

Initially, MGRAPH receives an image structure, including an ID, an UAV identifier and some metadata containing information from the sensors on-board the UAV (GPS and attitude data), see Fig. 1. After that, the content of the image itself is sent to the first idle feature detector process. In this step, binary keypoints and descriptors are computed using the GPU version of the ORB algorithm [11]. If the number of retrieved keypoints is less than a threshold, the image is dropped and not considered in this iteration. This might happen in presence of blurred images or flying over untextured areas such as seas, desserts or large countryside terrains. Since time per iteration is a critical measure to reach real-time processing, the ORB algorithm was selected over other alternatives such as SURF [12].

C. Neighborhood selection

Pairwise selection is the first step before adding a new image as a vertex in a graph. To this end, a zero-overlapping distance is calculated by using information from the metadata and the camera calibration. Assuming a completely orthogonal camera to the terrain (see Fig. 2), the size of the footprint in the horizontal axes can be calculated as $F_w = 2 \cdot h \cdot tan(FoV/2)$ where h is the flight height and FoV the camera's horizontal field of view. Taking into account the aspect ratio of the image

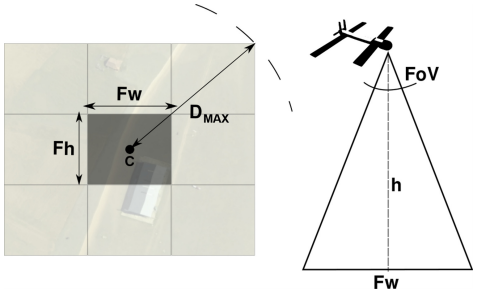

Fig. 2. Definition of the maximum distance (D_{max}) used by MGRAPH for the neighborhood selection process. The footprint of the image (F_w and F_h) can be calculated as a function of the Field-of-View of the camera and the flight height.

(AR), the footprint size in the vertical axes can be calculated as $F_h = F_w/AR$. Finally, the maximum distance D_{max} is:

$$D_{max} = 2 \cdot \sqrt{F_w^2 + F_h^2} \tag{3}$$

This distance corresponds to a zero percentage overlap between spatially neighbor images in both axis as shown in Fig 2. Hence, for each graph it is possible to calculate a candidate list whose elements are images located at a distance to the current image below D_{max}. By using this criteria, a full comparison is avoided in each iteration and only a local neighborhood is used.

D. Relative homography estimation

After creating the sorted candidate lists, each pairing set is sent to a different process in order to compute a relative homography matrix. To this end, descriptors are matched by a BruteForce algorithm (GPU version) using the Hamming distance [13]. A first set of matches passes through a ratio-distance filter in order to reject some outliers before applying the Random Sampling and Consensus (RANSAC) method. The homography estimation algorithm is based on the 2D affine estimator from [8]. Information about good matches and the initial homography estimation are sent back to the main process.

E. Graphs management

The next step involves the addition of the current image to the graph system. Let $G = \{g_1, g_2, g_3, \ldots, g_n\}$ be the current set of graphs in the system, and *img* the current image. The algorithm used to initialize the *img* into the graph system is shown in Algorithm 1.

For each graph, a candidate list is created based on the neighborhood selection criteria and using the metadata from the image (line 4). Then, features from each candidate are matched against the current image. The candidate is added to a special vector sorted by the number of matches (lines 5-8) if this number is greater than a fixed threshold. For every filtered candidate, a relative homography is computed (*relHomo* in line 10). The first candidate in the sorted list is used to insert the image in the specific graph, adding a *strong edge* and computing the absolute homography for the new vertex (lines

Algorithm 1: Graphs insertion algorithm

```
Input: Image img to be processed and graphs graphs
Output: Matched graphs
1  function processImage
2      matchedGraphs ← null
3      foreach g ∈ G do
4          candidates ←
             g.getNeighborhood(img.position)
5          foreach c ∈ candidates do
6              matches ← g.computeMatches(c, img)
7              if matches.size > thresh then
8                  sortCandidates.addSorted(c, matches)
9          foreach sc ∈ sortCandidates do
10             relHomo ← computeHomography(sc, img)
11             if !g.containsVertex(img) then
12                 g.addVertex(img)
13                 matchedGraphs.addSorted(g)
14             if sc.index == 0 then
15                 g.addStrongEdge(img, sc)
16                 g.computeAbsHomo(img, sc, relHomo)
17             else
18                 g.addWeakEdge(img, sc)
19             g.addConstraints(img, sc)
20     return matchedGraphs
```

11-16). If the candidate does not represent a strong edge, a *weak edge* is added to the vertex (lines 17-18). In any case, the constraints for the optimization problem are added (line 19). Finally, the total number of matched graphs is returned (line 20).

As a consequence of the previous algorithm, three possible cases are considered. First, it is possible that the current image does not match any of the existing graphs. In this case, a new graph with the current image as the reference vertex is created. Second, the image only matches one graph. In this case there is nothing left to do. Third, it is possible that the current image matches more than one graph. In this case, the current vertex becomes a *trigger vertex* T and starts a special process called Graph Fusion.

This fusion is defined as the process of concatenating two graphs (denoted as primary and secondary) by means of a common vertex in both structures. As a result, the secondary graph is reallocated in the reference system of the primary graph, deleting this graph after completion. The selection of the secondary graph is based on the number of vertices. However, if both graphs have the same size, the graph with more inliers with the trigger vertex will be selected as primary.

After defining the graph roles, an adjacency list from T to the rest of vertices in the secondary graph is created. An adjacency list is the representation of every edge in the graph as a list [14]. In the context of this research, every entry in the list is a tuple of two vertices and an attribute indicating if the edge is strong or weak. The objective of this list is to define the

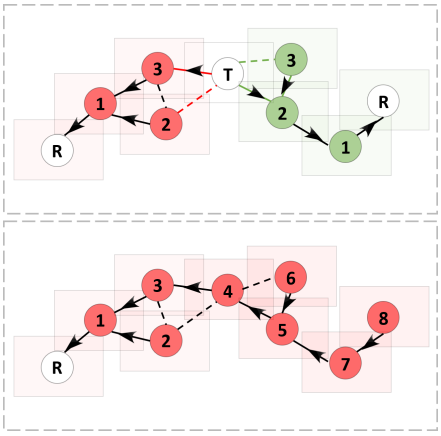

Fig. 3. (Top) A new image (called *trigger vertex*, T) matches two graphs simultaneously. (Bottom) As a consequence, the secondary graph (top image, in green) is referenced to the primary graph (in red). After fusion, the secondary graph is deleted. Dashed lines represents *weak edges* meanwhile stroke lines represent *strong edges*.

entrance order of vertices coming from the secondary graph into the primary graph. Assuming an adjacency list composed by n tuples $(v_0, v_1, type)$, representing the vertices IDs and the edge type, the following algorithm is executed for each element in the list:

1) If the tuple *type* is a 0 (*weak edge*), get the corresponding v_0 and v_1 from the primary graph and add the residuals to the optimization problem. If v_0 or v_1 are not present, push this tuple at the end of the list and continue with the next iteration.
2) If the tuple *type* is a 1 (*strong edge*), execute:
   a) Get the first vertex in the tuple v_0 from the primary graph.
   b) Retrieve the relative homography between vertices from the secondary graph $\mathbf{H}^{v_1,v_0}$. This homography was already computed in the past, when the original edge was created.
   c) Add a new node v_2 in the primary graph with absolute homography $\mathbf{H}^{v_2,R} = \mathbf{H}^{v_1,v_0} \cdot \mathbf{H}^{v_0,R}$.
   d) Add the corresponding edge and the residual to the optimization problem of the primary graph.

An example of a graph fusion is shown in Fig. 3. The red graph is selected as primary and the green graph as secondary. The following adjacency list can be computed from green graph:

$$X = [(T,2,1),(T,3,0),(2,3,1),(2,1,1),(1,R,1)] \quad (4)$$

The first tuple (T,2,1) creates the node 5 in the primary graph (number 2 in the secondary graph) by using a *strong edge* previously calculated. The second tuple (T,3,0) indicates a *weak edge* for vertices T and 3. Since vertex 3 has not been added yet, this value is pushed at the end of the list. In the following iterations, the tuples (2,3,1), (2,1,1) and (1,R,1) are added as vertices 6, 7 and 8 in the primary graph. Finally, the tuple (T,3,0) is processed again and a *weak edge* is added for vertices 4 and 6 in the primary graph.

If the current image matches more than two graphs, the same algorithm is executed between graph pairs until only one graph is left.

The main advantage of this approach is that we can work with datasets independently of the order the images are injected to the system. Different graphs that initially could be considered isolated can be connected to another graph through the graph fusion mechanism if the trigger image is introduced.

F. Non-Linear Optimization

Each node i belonging to a certain graph holds an absolute homography matrix $\mathbf{H}^{i,R}$. This matrix is initialized as the concatenation of the different relative transformations of *strong edges* from i to the reference frame R. Nonetheless, each relative homography has its own drift, so cascading them into an absolute homography can lead to cumulative errors. In order to minimize this effect, a global optimization is performed. According to the assumption of planar geometry in Section III-A, each pair of matched points (p_i, p_j) in two images i and j are related by a homography $\mathbf{H}^{i,j}$ satisfying $p_i = \mathbf{H}^{i,j} \cdot p_j$. Since the objective is to optimize the absolute homography of each image inside a graph, it is possible to express the previous relation as $p_i = \mathbf{H}^{i,R}(\mathbf{H}^{j,R})^{-1} \cdot p_j$. Therefore, the function to minimize is defined as:

$$f = \sum_i \sum_j \sum_p d(p_i, \mathbf{H}^{i,R}(\mathbf{H}^{j,R})^{-1} \cdot p_j) \tag{5}$$

where i and j are images with P correspondences, (p_i, p_j) are the matched points and finally $d(p_i, \mathbf{H}^{i,R}(\mathbf{H}^{j,R})^{-1} \cdot p_j)$ represents the geometric distance resulting from the reprojection. Due to the non-linear nature of the problem, the optimal parameters are determined using a non-linear optimization algorithm implemented in the C++ Ceres library [15].

In MGRAPH, since time constraints are critical and the convergence time of the optimization problem increases with the number of residuals, some strategies have been adopted. For instance, every graph has its own optimization problem and it is only executed every ten new images for two seconds maximum. Although this restriction does not assure convergence, an intermediate solution is given until the next optimization. Additionally, only the best 20 matches are used to generate the initial homography estimation so the number of residuals is more controlled.

IV. Mosaicking Output

In common mosaicking pipelines, the blending step occurs at the end of the process. This step implies allocating enough memory to create a seamless mosaic using the optimized homographies previously calculated. Despite the efforts for using CPU parallelization in previous research [8], the time spent in generating the seamless mosaic increases notably with the resolution and the number of images selected.

In this context, MGRAPH skips the traditional blending process. As previously shown in Fig. 1, MGRAPH returns a datagram to an external GCS containing information about the absolute homography of an *image i* in the *graph g*. In the GCS side, every image is already loaded in memory.

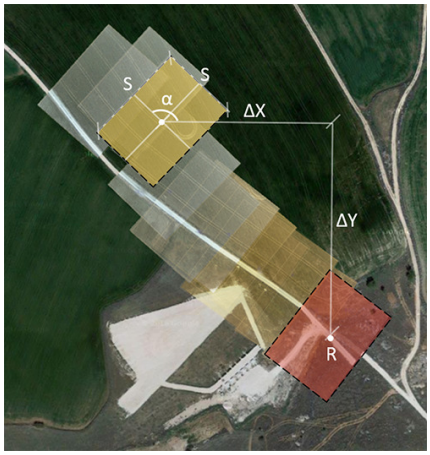

Fig. 4. Generation of the mosaic as an overlay in the GIS. A image i in a graph g is updated with the output generated by MGRAPH. A similarity transformation applies a translation $(\Delta x, \Delta y)$, a rotation (α) and a scale factor (S) to the image i in relation to its reference frame (R) in the graph g.

When it receives the result from MGRAPH, the GCS updates the absolute position, rotation and scale factor of the object corresponding to *Image i* with relation to the reference object of graph g.

In the example shown in Fig.4, the generation of a single-graph mosaic in the GCS is shown. The *reference frame* (R) is located using the metadata from sensors on-board the UAV (latitude, longitude and heading). Additionally, the corresponding zoom level is calculated taking into account this metadata and the camera model. Every time an update from MGRAPH is received, the corresponding image recalculates its absolute position and orientation ($\Delta x, \Delta y$, s and α) regarding the *reference frame* in its graph.

As a result, this approach is suitable to create incremental mosaics. Unfortunately, it introduces some drawbacks. Depending on light conditions, flight height and image overlap, skipping the blending part might cause visual artifacts. However, these drawbacks are considered acceptable when a real-time estimation is required. The creation of small seamless mosaics is considered as further work to complement this real-time view of the mosaic.

V. Experimentation

In order to evaluate the performance and accuracy of the proposed algorithm, two datasets have been selected: a publicly available dataset (Golf course[3]) and Drone airfield. The Drone airfield has been used with MGRAPH in order to check the performance and quality with large datasets. The Golf course dataset has been used to simulate a swarm dataset in order to address the on-line estimation with real data. The selected hardware for the experimental setup was an Intel Core i7 (4th Gen) 4800MQ with 32GB of RAM memory and a NVIDIA GPU GTX870m.

A. Golf course dataset

Having multiple UAVs flying at medium altitude in a large enough area is a risky and costly task, and we did not have

[3]https://dronemapper.com/sample_data

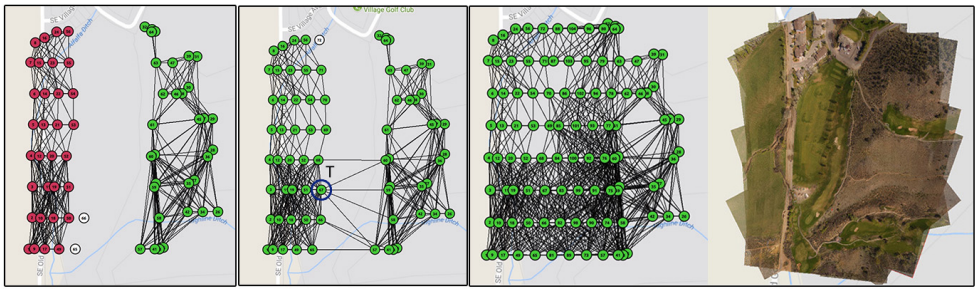

Fig. 5. (Left) Sequential MGRAPH execution of the Golf course dataset, reordered to simulate a swarm of 16 UAVs. At the begginning, two different graphs (red and green) are generated separately. (Center) Then, a trigger vertex (circled in blue) activates the graph fusion procedure. (Right) Final graph topology and result.

the resources to perform such experiment at this moment. Furthermore, it is difficult to find a public mapping dataset taken by more than one UAV. For this reason, this single-UAV dataset was specially selected due to its flight pattern. It was reordered to generate 16 rows, 8 from East to West and 8 from West to East, simulating a swarm of 16 UAVs. A set of 104 1000x750-pixel images was used. Additionally, the injection of images into the system was simulated by alternating one image per UAV in each iteration.

Figure 5 shows the initial state of the mosaic where two different graphs are generated (left). At some point, an arriving picture matches both graphs and a graph fusion process is started (center). After that, the graph topology and the final result are shown (right).

The time spent in the whole process was 32 seconds. This time includes graphs generation, graph fusion, non-linear optimization and periodically output generation to the GCS. The main goal of this dataset was to demonstrate the possibility of working with unconnected high-resolution datasets, creating separate graphs and generating an usable output in a reasonable amount of time.

B. Drone airfield dataset

The Drone airfield dataset was taken by a single Skywalker X5 Flying wing[4] customized with a Raspberry camera as the image sensor. The objective of this dataset was to test MGRAPH in a real environment with a large dataset, composed of 1436 pictures (1296x972 pixels) that covered an area of 2 by 1.5 km. In this case, camera calibration parameters were used in order to undistort images before processing. Simulating the injection rate of one picture per second, MGRAPH generated an output before the next image arrived. Every ten images global optimization was performed for two seconds (maximum), generating small delays.

During the processing of this dataset, small isolated graphs were created due to texture of the terrain. However, along the next tracks of the UAV, the small graphs were fused to the main graph due to the side overlap of the new images. In terms of performance considering an offline analysis, MGRAPH took 308 seconds to generate the final result that is illustrated in Fig. 6.

[4]https://hobbyking.com/es_es/skywalker-x-5-fpv-uav-flying-wing-1180mm.html

Fig. 6. Drone Airfield dataset composed by 1436 pictures.

VI. Comparative Results

This section evaluates the benefits and drawbacks of MGRAPH with respect to two different methods: a cloud-based mapping solution and the public state-of-art algorithm BIMOS [8]. Three datasets have been selected for the benchmark: Aukerman[5], Golf course[6] and Quarry[7], which are publicly available for download and incorporate geolocation metadata through embedded EXIF geotags. Before processing the images, it is assumed that they are undistorted. Finally, the hardware used is the same as in Section V.

A. Cloud-based mapping comparative

As a result of the development of both telecommunications and UAVs, different companies in the mapping industry started to offer their solutions as cloud-based services. For instance, Precision Hawk recently unified their products and released Precision Mapper, a cloud-based photogrammetry software for off-line aerial imagery processing[8]. Figure 7 illustrates the resulting mosaic from both the cloud-based service (left) and MGRAPH (center and right). Particularly, the Aukerman

[5]https://github.com/OpenDroneMap/odm_data_aukerman
[6]https://dronemapper.com/sample_data
[7]https://www.sensefly.com/drones/example-datasets.html
[8]http://www.precisionhawk.com/es/precisionmapper

Fig. 7. (Left) Cloud-based output of the Auckerman dataset compared with the output generated by MGRAPH (center) and the graphs generated (right).

Fig. 8. (Left) BIMOS output of the Quarry dataset compared with the output generated by MGRAPH (center) and the graphs generated (right).

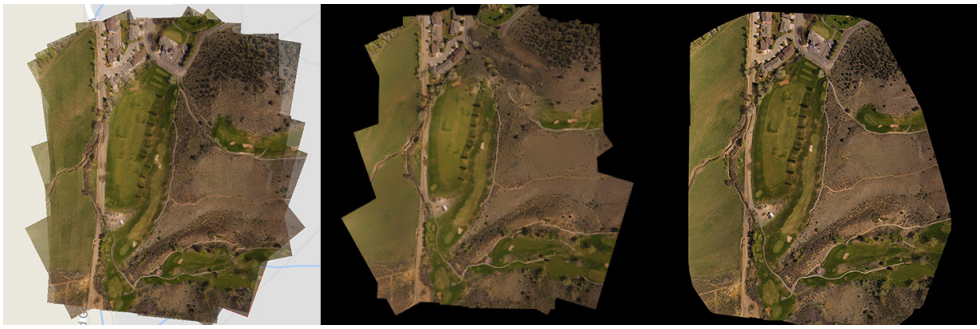

Fig. 9. (Left) MGRAPH output of Golf course dataset compared with the output generated by BIMOS (center) and Precision Mapper (right).

TABLE I
EXPERIMENTAL RESULTS FROM THE THREE DATASETS COMPARATIVE. MINIMUM VALUES ARE REPRESENTED IN BOLD.

			MGRAPH				BIMOS				PRECISION MAPPER	
			Times (s)		Rep. error (px)		Times (s)		Rep. error (px)		Times (s)	
Dataset	#Imgs	Sizes	t_{PROC}	t_{BLEN}	Avg	Std	t_{PROC}	t_{BLEN}	Avg	Std	Upload	Processing
Quarry	210	1152x864	94.59	–	11.15	17.73	**88.48**	1805.14	**10.21**	17.67	500	1200
Golf	104	1000x750	**32.23**	–	7.23	14.61	94.46	502.94	**6.54**	50.86	200	202
Aukerman	77	1224x918	**19.64**	–	9.53	16.56	47.66	498.58	**8.40**	20.65	150	553

dataset was selected for this comparison and the results are shown in Table I. Since the final user is isolated from the mosaicking pipeline, the selected metric is based on the total waiting time, split into upload and processing time.

Taking into account the results from Table I, MGRAPH is much faster than the results obtained by Precision Mapper and, more importantly, it offers incremental solutions. On the other hand, the total time spent by the Precision Mapper approach highly depends on the network status. Finally, since isolated graphs from MGRAPH are not shown in the final result, Precision Mapper obtains more accurate results in areas where MGRAPH presents some artifacts.

B. BIMOS comparative

The second pipeline to be compared with is called BIMOS [8]. This publicly available stack introduces a generic image mosaicking algorithm that produces seamless composites, using a multi-threading approach in a ROS-based environment. While there are different assumptions behind the proposed approach and BIMOS, we consider BIMOS a solid reference to evaluate the re-projection errors, the mosaic and the processing times obtained by MGRAPH. Two are the main differences

of MGRAPH with respect to BIMOS: i) the integration of multiple disconnected graphs during the mosaicking process and ii) considering the metadata associated to the images to build georreferenced mosaics and to provide candidate matching pairs.

Figure 8 shows the difference between both algorithms for the Quarry dataset. In this case, the output quality is similar, presenting the same artifacts in the bottom part of the mosaic. Table I compares different quality metrics for the selected datasets. Processing time is significantly lower in the MGRAPH case due to its mosaic generation approach. In the case of BIMOS, the generation of a seamless mosaic delays considerably the performance of the algorithm. Nonetheless, the time spent before blending is quite similar. Although BIMOS does not use GPS for registration, MGRAPH includes the graph generation and graph fusion procedures in t_{PROC}. Additionally, BIMOS uses a keyframe selector that decides if the image is part of the final result. These reasons could explain the difference in time for the Quarry Dataset. In terms of reprojection error, there is no significant difference between measures (about 1 pixel), although BIMOS obtains a slightly better result. This might be caused by the shorter time dedicated by MGRAPH for graph optimization (up to two seconds every 10 images), while BIMOS was configured with for up to 30 seconds every 30 images plus up to 600 seconds before blending. Better re-projection errors can be obtained if larger times are allowed in the optimizer, it is a trade-off between quality and processing time.

C. Triple comparative

The Golf Course dataset was tested in both alternatives, as shown in Fig. 9. In this case the final mosaic looks similar with all methods. Also, the time spent by MGRAPH was clearly shorter than Precision Mapper and BIMOS. Finally, the Drone airfield dataset was also benchmarked against BIMOS and Precision Mapper. In the case of BIMOS, due to the known limitations of the blending submodule in the OpenCV stack, a memory allocating error occurred. On the other hand, with Precision Mapper it was impossible to upload the dataset because the camera model was not in their database.

VII. Conclusions and Future Work

The main purpose of the research presented in this paper has been to introduce MGRAPH, a novel mosaicking pipeline specifically designed to work with images from an UAV swarm. From the research that has been carried out, it is possible to conclude that MGRAPH can generate an output faster than other solutions, thanks to its incremental output generation approach. Additionally, the proposed graph-based approach allows to transparently integrate numerous image streams from different UAVs, independently of the order injection. MGRAPH has been tested using publicly available datasets and successfully benchmarked against a web-based mosaicking system and a state-of-art method obtaining better results in terms of performance. Moreover, it was demonstrated that MGRAPH can work with large datasets in a reasonable amount of time.

As future work, it is intended to test MGRAPH in a real UAV swarm scenario to check that there is not any performance degradation and the generation and fusion of graphs is performed properly. Furthermore, avoiding the blending part can lead to artifacts in the mosaic. A partial seamless mosaic is proposed to be generated on demand to palliate the artifacts. Moreover, temporal constraints needs to be verified in order to maintain the real-time execution. The optimization problem has been identified as the main temporal bottleneck in the system. Further research is needed to minimize this effect. Finally, considering GPS data into the optimization problem can lead to better alignments [16] and thus is also proposed as future work.

References

[1] Kolling, A., Walker, P., Chakraborty, N., Sycara, K., and Lewis, M., "Human Interaction with Robot Swarms: A Survey," *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 1, pp. 9–26, 2016.

[2] Elibol, A., Gracias, N., Garcia, R., Gleason, A., and Gintert, B., "Efficient autonomous image mosaicing with applications to coral reef monitoring," 2011.

[3] Navia, J., Mondragon, I., Patino, D., and Colorado, J., "Multispectral mapping in agriculture: Terrain mosaic using an autonomous quadcopter UAV," in *2016 International Conference on Unmanned Aircraft Systems, ICUAS 2016*, pp. 1351–1358, 2016.

[4] Pinto, A. M., Pinto, H., and Matos, A. C., "A Mosaicking Approach for Visual Mapping of Large-Scale Environments," *Proceedings - 2016 International Conference on Autonomous Robot Systems and Competitions, ICARSC 2016*, pp. 87–93, 2016.

[5] Moussa, A. and El-Sheimy, N., "A fast approach for stitching of aerial images," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, vol. 41, pp. 769–774, 2016.

[6] Nam, D. and Aouf, N., "Automated Mosaicing for Improving Vehicle Situational Awareness in Real Time," *IEEE Intelligent Vehicles Symposium, Proceedings*, no. Iv, 2017.

[7] Richmond, K. and Rock, S. M., "An operational real-time large-scale visual mosaicking and navigation system," in *OCEANS 2006*, 2006.

[8] Garcia-Fidalgo, E., Ortiz, A., Bonnin-Pascual, F., and Company, J. P., "Fast image mosaicing using incremental bags of binary words," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1174–1180, 2016.

[9] Bu, S., Zhao, Y., Wan, G., Li, K., Liu, Z., and Han, J., "Map2DFusion : Real-time Incremental Aerial Images Mosaic Based on Monocular SLAM," pp. 4564–4571, 2016.

[10] Alamareen, A., Al-Jarrah, O., and Aljarrah, I. A., "Image mosaicing using binary edge detection algorithm in a cloud-computing environment," *International Journal of Information Technology and Web Engineering*, vol. 11, no. 3, pp. 1–14, 2016.

[11] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G., "ORB: An efficient alternative to SIFT or SURF," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2564–2571, 2011.

[12] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L., "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[13] Garcia, V., Debreuve, E., Nielsen, F., and Barlaud, M., "K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching," in *Proceedings - International Conference on Image Processing, ICIP*, pp. 3757–3760, 2010.

[14] Singh, H. and Sharma, R., "Role of Adjacency Matrix & Adjacency List in Graph Theory," *International Journal of Computers & Technology*, vol. 3, pp. 179–183, 2012.

[15] Sweeney, C., Sattler, T., Hollerer, T., Turk, M., and Pollefeys, M., "Optimizing the viewing graph for structure-from-motion," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 801–809, 2015.

[16] Ferrer, J., Elibol, A., Delaunoy, O., Gracias, N., and Garcia, R., "Large-area photo-mosaics using global alignment and navigation data," in *Oceans Conference Record (IEEE)*, 2007.