CRANFIELD UNIVERSITY

ITXASO DÍAZ PÉREZ

SWARMING DRONE SENSORS

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Autonomous Vehicle Dynamics and Control

MSc
Academic Year: 2017–2018

Supervisor: Dr Hyo-Sang Shin and Prof. Antonios Tsourdos
August 2018

CRANFIELD UNIVERSITY

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Autonomous Vehicle Dynamics and Control

MSc

Academic Year: 2017–2018

ITXASO DÍAZ PÉREZ

Swarming drone sensors

Supervisor: Dr Hyo-Sang Shin and Prof. Antonios Tsourdos
August 2018

This thesis is submitted in partial fulfilment of the requirements for the degree of MSc.

Abstract

In the last decades, the presence of UAVs has increased widely in the military world, as they are able of monitoring conflict areas without endangering human lives. Many of these UAVs have the disadvantage of being quite big and expensive; therefore, the trend now is to use lots of smaller and cheaper drones which make it possible for the system to continue working even if a couple of drones get lost or are unable to contribute.

In this project it has been designed a robust system that using a fixed number of drones with single cameras on them delivers a good resolution picture comparing with the ones that are obtained from expensive systems. Given a certain mission area, first a task allocation algorithm assigns some tasks or positions in the mission area to each UAV, in a way that the information collected by the images is maximized. After that, an image mosaicing algorithm will process those images in order to return the final mosaic.

The whole thesis has been developed in a simulation environment in Matlab. The results show that the proposed algorithms guarantee that the complete mission area will be covered by the UAVs in the shortest possible time. In addition, the obtained final mosaic represents perfectly the considered mission area when an adequate overlapping area is considered. Finally, the system has been proven to be resilient if a UAV is unable to contribute for some reason.

Keywords

UAV; Image mosaicing; Task allocation; Greedy algorithm; Clustering; Travelling Salesman Problem.

Contents

List of Figures

List of Tables

List of Abbreviations

ANMS	Adaptive Non-Maximal Suppression
DP	Dynamic Programming
FOV	Field Of View
MI	Mutual Information
NCC	Normalized Cross Correlation
RANSAC	Random Sample Consensus
SATM	School of Aerospace, Technology and Manufacturing
SEEA	School of Energy, Environment and Agrifoods
SSD	Sum of Square Differences
TSP	Travelling Salesman Problem
UAV	Unmanned Aerial Vehicle

Acknowledgements

I would like to thank my supervisors Dr Hyo-Sang Shin and Professor Antonios Tsourdos for their support during the thesis, giving me suggestions and helping me with any issue I have had along the project.

Above all, I would like to thank my parents for their constant support since the beginning of my studies and in my personal life, making me feel that I can overcome any challenge that I propose. My twin sister has also played an important role in my life, as I have been lucky to grow with her both personally and academically, and we have shared as well this year's experience.

Finally, I want to thank all my new friends I have met in Cranfield this year. Because of them this year has been probably the most special one of my life, and I am sure I will keep most of these friendships with me all my life.

Chapter 1

Introduction

In the last decades, the presence of Unmanned Aerial Vehicles (UAVs) has increased widely in the military word. The reason behind this increase is that they are able of monitoring or attacking an area of conflict without endangering human lives. Therefore, numerous researches have been performed in the last decade, which have led to a huge prolgress in this field. Currently they can be found a variety of sizes of these vehicles, which range from several meters to few centimeters. The hundreds of applications related to the UAVs such as monitoring, mapping, rescue missions in disaster areas, aerial imaging,... and others that are emerging, make it clear that it is a field that will experiment a huge growth in the near future.

1.1 Project Motivation

The idea behind this project was based in the UK Watchkeeper Unmanned vehicle that Thales UK has supplied to the UK army. The Watchkeeper X is an unmanned aircraft system designed for a wide range of military and homeland security mission requirements, as it can operate in extreme and challenging environments to collect and distribute crucial information. One of its applications is that it can transmit high quality images and video securely and reliably to different locations, providing commanders information that highly

improves situational awareness.

Figure 1.1: Thales Watchkeeper Unmanned Aircraft System [1]

However, the main disadvantage of this system is that it is quite big and expensive. So, losing this vehicle or having any damage in any part of it while performing the mission would have a big impact, as it would not be able to complete its purpose. That is why, the trend now is to use multiple UAVs with single cameras incorporated on them working together as a swarm.

This alternative is supported by several advantages. On the one hand, nowadays there is a huge variety of UAVs, of different sizes and prizes. Due to this cost reduction, it is possible to use smaller and cheaper drones that work together to perform the same tasks that were previously performed by expensive systems. In addition, a system which is formed by multiple UAVs makes it possible to reduce the time required to fulfill the mission, as the time needed to solve all the tasks is reduced. On the other hand, interconnection of multiple UAVs provides also a more flexible solution, as the system can continue working even if a couple of drones get lost or are unable to contribute for any reason.

[1] Image obtained from https://www.thalesgroup.com/en/global/activities/defence/unmanned-aerial-vehicles-systems/tactical-uav

Therefore, in order to obtain high quality images of a certain area by using multiple agents working together as a swarm, it will be necessary to coordinate their positions in an optimal way, with the objective of achieving a wider field of view of the area of interest. However, it should be taken into account that the performance of this system will always be limited by the resolution of the cameras they use. So, this proposal is not about achieving the same results that are obtained from expensive systems, but about providing a more affordable alternative.

1.2 Aims and Objectives

The aim of this individual research project could be defined as: "*To design a robust system that using a fixed number of drones with single cameras on them delivers a good resolution picture comparing with the ones that are obtained from expensive systems.*"

Therefore, the mission scenario considered in this individual research project is the case in which a certain area of high interest requires persistent surveillance and monitoring. It will be assumed to be a 2D area from which a complete image wants to be obtained. Therefore, the agents should be allocated to certain positions in the space in an optimal way.

After defining the aim of this project they will be listed below the set of objectives that must be fulfilled to achieve it:

1. *"To develop an image mosaicing algorithm."* The input data of this algorithm will be multiple overlapping images of a scene. Therefore, the code must be able to process these photographs and establish a geometric correspondence between them. Once this is achieved, a final mosaic should be formed, which is the output data, by stitching these images among them correctly.

2. *"To develop a mobile tasking algorithm."* The efficient cooperation of a group of UAVs is a vital part for the mission success. This task allocation code should

allocate the mobile sensing platforms to some positions in the mission scenario where the drones should go and take a photograph. In addition, once the most suitable tasks are selected, it should be decided which UAV goes to each task, that is, each selected task should be assigned to a certain vehicle, trying to minimize the travelled distance by each drone. Finally, it will be required to find the sequence of waypoints for each UAV.

3. *"Integration between the image mosaicing and the mobile tasking algorithms."* The main point in this step will be to develop a mobile tasking algorithm which can reflect the performance of mosaicing.

4. *"Validation of the proposed solution."* First, the different algorithms of the proposed solution will be validated. After that, the integration between the image mosaicing and the mobile tasking algorithms will be validated by means of a simulation in Matlab.

1.3 Project Definition

In order to present the ideas in a clear way this project has been organized as follows:

1. **Literature Review (Chapter 2):** In this chapter it will be performed a search and evaluation of the available literature (papers, books...) about the different subjects related to this project. This is the most important step before starting the project, as it is essential in order to choose the most suitable techniques in each developed algorithms.

2. **Developed image mosaicing and task allocation algorithms (Chapter 3):** After choosing the most appropriate techniques for the image mosaicing and the mobile tasking algorithms, the methodology followed in the developed codes will be explained.

3. **Analysis and Results (Chapter 4):** In this chapter the performance of the developed algorithms will be analyzed in order to validate them.

4. **Conclusions (Chapter 5):** After analyzing the obtained results, some conclusions will be presented along with the line in which future students should continue investigating.

Chapter 2

Literature Review

2.1 Introduction to image mosaicing

Image mosaicing is a process that consists on stitching multiple overlapping images of a scene with the objective of achieving a wider field of view. Therefore, the idea behind this technique is to increase the field of view of a camera without decreasing the resolution of the image and without having to introduce an undesirable deformation in the lens of the camera. Nowadays, as there are a wide range of applications where it can be applied, such as motion detection, resolution improvement, medical images..., it has been the subject of numerous studies in the last two decades and different algorithms have been proposed.

Image mosaicing algorithms can be classified into various groups according to the technique they use. However, even if there are different techniques mosaicing always involves the same steps of image processing:

- Image acquisition
- Registration
- Reprojection
- Stitching

- Blending

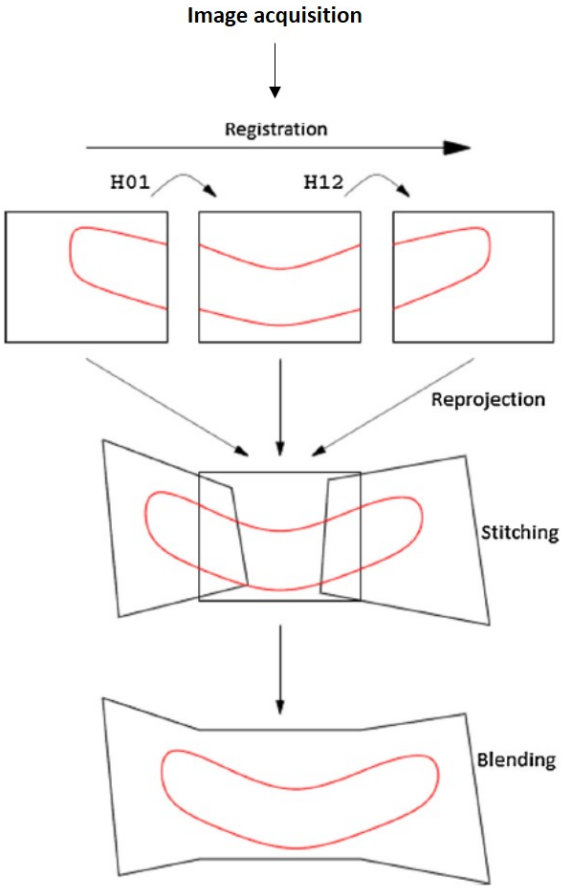


Figure 2.1: Steps of image mosaicing algorithms [1]

1. **Image acquisition:** In this step input images are obtained from some source, which normally is a hardware-based one. This input data consists of two or more images taken from the same scene but at different times, from different points of view or by different sensors.

2. **Registration:** A geometric correspondence is established between the different images that have been obtained in the previous step. Registration requires to calculate the geometric transformations that are required to align the images with respect to a reference. This step includes calculating the homography matrices (3×3) between

[1] Image obtained from (Ghosh and Kaabouch, 2016*a*)

source images which was algebraically demonstrated in (Hartley and Zisserman, n.d.) that a mapping on the projective plane P^2, that is, from $P^2 \rightarrow P^2$ is a projectivity if and only if there exists a non-singular 3×3 matrix H which for any point in P^2 is represented by a vector x, it is verified that its mapped point equals the vector given by the multiplication Hx. Therefore, calculating H in the mosaicing algorithm will be enough to calculate the homography that maps each x_i point from an image to its corresponding $x_i^{'}$ in another image.

3. **Reprojection:** It is the action of aligning the images into a common coordinate system applying the geometric transformations that have been calculated previously.

4. **Stitching:** The objective of this step is to overlay the images that have been aligned in the reprojection step on a bigger canvas. This is done by combining the pixel values of the overlapping zones of the images and maintaining the same values where no overlap happens.

5. **Blending:** As a consequence of geometric and photometric misalignment errors can appear in the boundary between the different images. Some of these errors might be object discontinuities and seams visibility. Therefore, a blending algorithm becomes necessary during or after the stitching step to avoid big discontinuities in the final mosaic.

2.2 Classification of image mosaicing algorithms

The performance of the image mosaicing algorithms is mostly influenced by the registration and blending steps. This makes sense because they correspond with the first and last steps of image mosaicing; so, their proper operation becomes essential in order to build successful algorithms. Therefore, below image mosaicing algorithms will be classified as in (Ghosh and Kaabouch, 2016*a*) based on registration or based on blending.

1. Classification of image mosaicing based on registration

Registration takes images from different sources that correspond to the same target but that come from different sensors or from different perspectives and calculates the optimal geometric transformation by comparing both images and obtaining the correspondences between them. These correspondences can be obtained by different ways. These can be divided in two main groups as indicated in Figure 2.2. On the one hand, algorithms that apply spatial domain-based mosaicing algorithms that can be divided at the same time into area based or feature based, and on the other hand, by using the frequency domain-based phase correlation property.

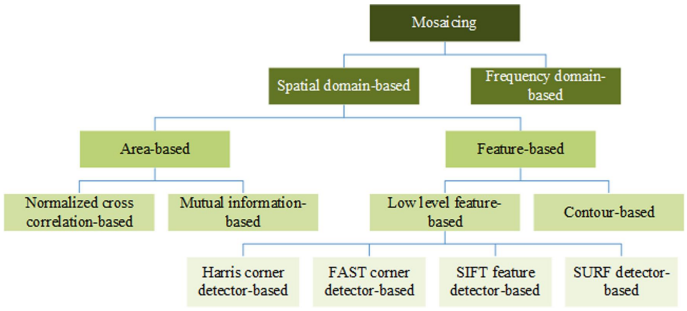


Figure 2.2: Classification of image mosaicing based on registration

1.1 Spatial domain image mosaicing algorithms

Registration is performed considering the properties of pixels of the images. Most image mosaicing algorithms correspond to this category. As it has been mentioned before, these can be divided as well into two other groups: area-based or feature-based.

Area-based algorithms

Area-based codes depend on computations between ”windows” of pixel values between the images that want to be stitched. In (Ghannam and Lynn, 2013) a comparison is performed between two of the most important methods in this category, which are based on Normalized Cross Correlation (NCC) and on Mutual Information (MI) respectively. The results demonstrate that both methods have similar performance in a lot of situations,

[1]Image obtained from (Ghosh and Kaabouch, 2016*a*)

however, there are some differences between them. On the one hand, when there are large rotation angles NCC-based ones perform better. However, the choice of a distinctive template becomes crucial in order to obtain good results. On the other hand, when using MI an acceptable performance was obtained even if no distinctive templates were used. Anyway, both techniques have the disadvantage of being computationally slow and also of requiring significant overlappings between the input images.

Feature-based

These mosaicing methods rely on feature-to-feature matching between a pair of input images. Therefore, they depend mainly on feature extraction algorithms, which are used to detect salient features from the input images such as points, edges, corners, colours, textures... The resulting features will be subsets of the image domain, which usually have the form of isolated points, continuous curves or connected regions (Islam and Kabir, n.d.).

Most popular algorithms inside this category are the low level feature-based ones, that is, Harris, FAST, SIFT and SURF, as the computation of the countour-based method is too expensive because of the use of high-level features. Each of these four methods has its own advantages and disadvantages, so, it is fundamental to analyze these in order to choose the most appropriate one for each application.

Harris corner detector: Feature points are detected following the algorithm explained in (Ghosh and Kaabouch, 2016*b*). This algorithm provides a simple and accurate computation. In addition, even if it usually detects closely crowded features this can be solved by restricting the maximum number of features in the neighborhood, that is, some of these points are excluded. This has been performed in (Okumura et al., 2013) or (Brown et al., 2005). The main disadvantage of this algorithm is that it is only good for moderate changes in scale an rotation.

FAST corner detector: It has an efficient and fast computation, so they are good for real-time image processing applications. One of its disadvantages is that it requires prior

knowledge about the optimal threshold required in order to detect the corners, which becomes usually a challenge. That is why, it can be added a threshold selecting algorithm as in (Jichao Jiao et al., 2011).

SIFT feature detector: It is efficient with high resolution images. In addition, it is not influenced by translations, rotations and scaling transformations in the image domain and robust to small perspective transformations and illumination variations. However, it is computationally expensive.

SURF: Because of its fast computation it is convenient for real-time applications. Nevertheless it has a poor performance under some transformations such as colour, illumination...

1.2 Frequency domain image mosaicing algorithms

These methods require computation in the frequency domain in order to find the best transformation parameters between the input images. These algorithms apply the property of phase correlation for registration. After calculating the cross-power spectrum of the two images, the shift theorem ensures that the phase of the cross-power spectrum is equivalent to the phase difference between the images.

2. Classification of image mosaicing based on blending

Blending in used to avoid errors in the boundary between images. These errors can have different sources such as difference in camera exposure, variation in the illumination of the environment, appearance of moving objects between frames, geometric misalignments... which can decrease the consistency of the final mosaic.

Blending algorithms can be classified into 2 groups as it can be seen in Figure 2.3: Transition smoothening-based and optimal seam-based. In the methods of the first group the information of the overlapping region between two images is combined so that the boundaries of the images of the mosaic become undetectable. On the other hand, optimal seam-based methods try to minimize the seams in the boundary between images by trying

to allocate the seam line in the optimal location. This is performed by analyzing the information content in the overlapping regions of the images.

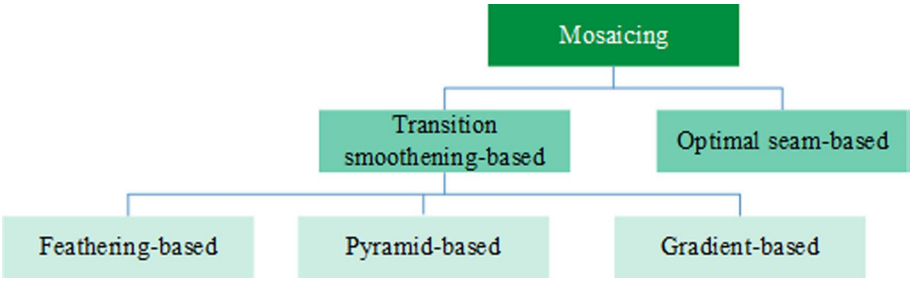


Figure 2.3: Classification of image mosaicing based on blending [2]

It can be seen in Table 2.1 a review on the main advantages and disadvantages of the different registration algorithms.

Method	Advantages	Disadvantages
Feathering-based	Easy to implement, fast and good performance under exposure differences	The final mosaic frequently experiences blur and ghosting effect
Pyramid-based	Appropriate to prevent blur and duplication of edges	If the error from the registration step is big, output will suffer from double contouring and ghosting effect
Gradient-based	The final mosaic is usually more attractive than with other methods	High computational requirements and the error from the registration step must be small in order to obtain a good performance
Optimal seam-based	Good performance when there are moving objects and parallax effects	Transition between images becomes visible when there are exposure differences

Table 2.1: Comparison among the different blending mosaicing methods

[2] Image obtained from (Ghosh and Kaabouch, 2016*a*)

2.3 Task allocation

The efficient cooperation of a group of UAVs, which is called, task allocation is fundamental in order to successfully fulfill the mission. In the sensor placement problems there are M candidate positions where the sensor can be placed, and N sensors, being N quite smaller than M in general. The main problem with the task allocation matter is that it has been verified to be NP-hard most of the times (Shin and Segui-Gasco, 2014), which means that the problem requires exponential time to be solved optimally.

In order to deal with this NP-hardness concern multiple studies have been developed and different kind of algorithms have been proposed. These algorithms can be classified in three different groups:

- Exact approach algorithms
- Heuristic approach algorithms
- Approximation approach algorithms

Exact approach algorithms obtain optimal solutions, but usually they do not guarantee polynomial time complexity. When the size of the problem is small, these algorithms can find the optimal solution in limited time. However, if the size of the problem increases or there is a limitation in the available time to find the solution, these kind of algorithms are not useful.

Heuristic approach algorithms find viable solutions with certain converge speed. However, the quality and the optimality of the result are not guaranteed.

Finally, approximation approach algorithms are able to find a solution that compensates the optimality and the computational complexity. That is why numerous approximation algorithms have been proposed. In addition, if the problem satisfies the submodularity condition its level of optimality and computational complexity can be mathematically proven, that is the reason why many maximization problems use submodular

functions. In (Krause and Golovin, 2013) the concept of submodularity is introduced and several example functions are mentioned. Algorithms that make use of these properties are called submodular maximization algorithms.

Submodularity has been already implemented in many path planning problems, in different areas such as artificial intelligence (Singh et al., n.d.) or robotics (Heng et al., 2015) .

The most popular submodular optimization algorithm is the greedy algorithm. Its main characteristics are that it is easy to implement and that it can be mathematically verified its level of optimality and computational complexity. However, even if the polynomial time complexity in the task allocation algorithm that uses the greedy technique is guaranteed, it should be taken into account that if the size of the problem gets bigger the computational load increases excessively. In (Nemhauser et al., 1978) it is proven that the result obtained by applying the greedy algorithm is a good approximation to the optimal solution of the NP-hard maximization problem.

2.4 Travelling Salesman Problem

Once the task allocation problem is performed each UAV will have a list of tasks to perform, that is, a list of positions to visit. As the endurance of the UAVs is limited, it will be necessary to travel to all these positions trying to minimize the total distance travelled by each drone. This issue can be faced as a case of the Travelling Salesman Problem.

The Travelling Salesman Problem (TSP) (Goyal, n.d.) is one of the most famous NP-hard problems, as it can be applicable in a wide range of fields. The problem is formulated in the following way:

"*Given a list of cities and their pair wise distances, the task is to find a shortest possible tour that visits each city exactly once.*"

The TSP can is divided usually into two categories: the Symmetric TSP where the distance between each pair of two cities is the same in both directions, and the Asymmetric TSP where it might only exist a path in one of the directions or the distance between each pair of two cities is not the same in both directions.

In order to solve this problem there are several ways. Solving the TSP by means of a naive approach provides an exact algorithm, but is usually a highly inefficient option. One example of applying a naive approach consists on trying to find the lightest hamiltonian circuit in the graph. So, for these case it will be necessary to find all possible Hamiltonian circuits in the graph and selecting the one with the shortest length. However, these is a NP complete problem, as for N nodes in a graph and having one starting and ending nodes, there are $(n-1)!/2$ maximum possible Hamiltonian cycles in a symmetric TSP. Therefore, this leads to a $O(n!/2)$ runtime if these circuits are compared between them. In order to reduce this runtime, they have been suggested numerous deterministic and non-deterministic algorithms through the years.

Between the deterministic approaches it should be mentioned the dynamic programming formulation that Held and Karp presented in (Held and Karp, 1961), and which had a time complexity of $O(2^n n^2)$, but which had a limitation in the N number. The branch and bound technique based algorithm published in (Little et al., 1963) made it possible to increase the size of the problem up to 40 cities with appreciable average runtime.

Non-deterministic solutions to TSP are helpful when the running time of the algorithm is more important than the accuracy of the result. In (Hahsler and Hornik, 2007) they are described some of the implemented approximate algorithms such as: Nearest Neighbour Algorithm, Insertion Algorithms and K-Opt Heuristics.

In (Goyal, n.d.) a greedy non-deterministic approach to solve the TSP in polynomial time has been proposed. Even if as in other common greedy approaches this algorithm does not work perfectly for some cases of the problem, it halts in polynomial time for every case and it provides an exact solution to the problem instances it works for.

Chapter 3

Developed image mosaicing and mobile tasking algorithms

3.1 Image mosaicing algorithm

As it has been explained in Chapter 2, image mosaicing is a process that consists on stitching multiple overlapping images of a scene with the objective of achieving a wider field of view (FOV). In this project an image mosaicing algorithm has been developed in Matlab. The diagram in Figure 3.1 shows the principal steps of this algorithm, which are going to be explained below.

3.1.1 Harris Corner Detector Algorithm

As it has been explained in the literature review, there are different feature extraction techniques in order to perform the registration technique. Among all these techniques it has been selected the Harris corner detector based algorithm because of its simple and accurate computation. This algorithm was developed by Chris Harris and Mike Stephens in 1988 (Harris and Stephens, 1988).

The procedure followed by this algorithm was explained clearly in (Vaghela and Naina, n.d.).

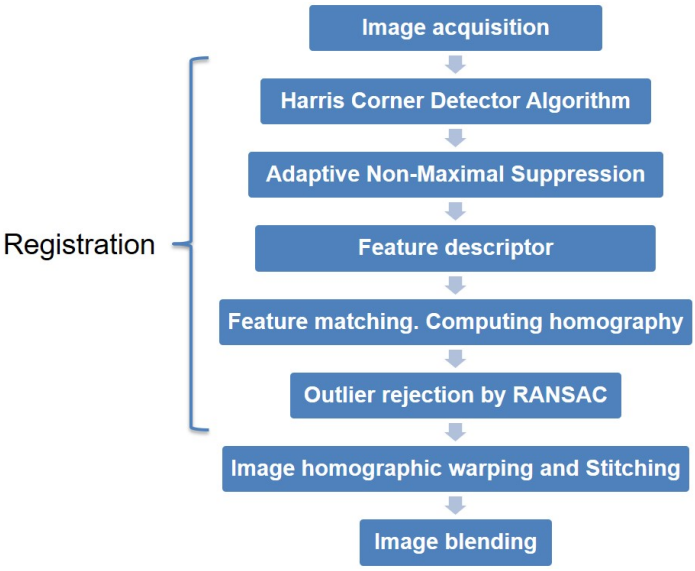


Figure 3.1: Main steps of the image mosaicing algorithm developed in Matlab

First, a small local detecting window is depicted in the photograph, and the average variation in intensity obtained by shifting this window a little bit in different directions is calculated. When the window is shifted there are three possible alternatives that might happen.

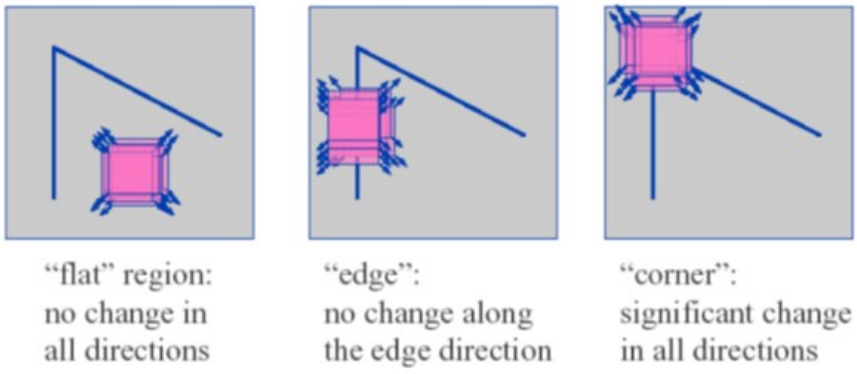


Figure 3.2: Possible alternatives when shifting the local window in Harris Corner Detector Algorithm [1]

- Flat region: When the window is shifted there will not be any change in intensity in all direction.

[1] Image obtained from (*The Effect of Using Inter-Frame Coding with Jpeg to Improve the Compression of Satellite Images*, 2017)

- Edge: When the window is shifted there will not be any change in intensity in the direction of the edge.

- Corner: When the window is shifted there will be an important change in intensity in all the directions.

So, Harris corner detector provides a mathematical approach in order to determine if the region that it is been analyzed is flat, or if there is an edge or corner. This algorithm tends to find closely crowded features, but this will be overcome by applying a novel Adaptive Non-Maximal Suppression algorithm later. However, it should be taken into account it is not a suitable technique for images with large changes in scale and rotation.

This is the mathematical procedure followed by Harris corner detector algorithm:

The change of intensity produced by a shift [u,v] is calculated as:

$$E_{x,y} = \sum_{u,v} w_{u,v}[I_{x+u,y+v} - I_{u,v}]^2 = \sum_{u,v} w_{u,v}[xX + yY + O(x^2, y^2)]^2 \tag{3.1}$$

Where:

- I(x,y): Intensity of the individual pixel

- I(x+u,y+v): Shifted intensity

- w: Specifies the image window. It is recommended to use a smooth circular window, for example a Gaussian:

$$w_{u,v} = exp[-(u^2 + v^2)/2\sigma^2] \tag{3.2}$$

The first gradients are approximated by:

$$X = I \otimes (-1, 0, 1) \approx \frac{\partial I}{\partial x} \tag{3.3}$$

$$Y = I \otimes (-1,0,1)^T \approx \frac{\partial I}{\partial y} \tag{3.4}$$

So, for small shifts the value of the change E produced by a shift can be calculated as:

$$E_{x,y} = Ax^2 + 2Cxy + By^2 \tag{3.5}$$

Where:

$$A = X^2 \otimes w \tag{3.6}$$

$$B = Y^2 \otimes w \tag{3.7}$$

$$C = (XY) \otimes w \tag{3.8}$$

As the operator answers too early to edges because only the minimum of E is taken into account, the expression of the change E will be rewritten as:

$$E_{x,y} = (x,y)M(x,y)^T \tag{3.9}$$

Being M a 2x2 symmetric matrix defined as:

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \tag{3.10}$$

Finally, not only do we need corner and edge classification regions, but also a measure of corner and edge quality of response. Defining the values of Tr(M) and Det(M) as:

$$Tr(M) = \alpha + \beta = A + B \tag{3.11}$$

$$Det(M) = \alpha\beta = AB - C^2 \tag{3.12}$$

And calculating the corners measure for each image pixel (x,y), it is obtained:

$$R = Det - kTr^2 \tag{3.13}$$

Being R positive in the corner region, negative in the edge region, and small in the flat region.

Harris method considers that the feature points are the pixel values that correspond with the local maximum interest point. So, a corner region pixel will be selected as a nominated corner pixel if its value is an 8-way local maximum.

Finally, a threshold T is set and corner points are detected.

In the image mosaicing algorithm that has been developed in Matlab, Harris corner detector algorithm is applied to the input images by means of the Matlab function named as: *corner_detector.m*. The input and output arguments of this function are:

Input arguments:

- image_b: an HxW matrix representing the gray scale image whose feature points want to be extracted.

Output arguments:

- corner_img: an HxW matrix representing the gray corner metric matrix of the input image.

Being H and W respectively the height and width of the image.

3.1.2 Adaptive Non-Maximal Suppression algorithm

As it has been mentioned before, Harris corner detector tends to find closely crowded features. In addition, as the computational cost of matching increases as the number of

interest points increases, it will be beneficial to limit the maximum number of feature points selected from each image. At the same time, it is important for the feature points to be spatially well distributed over the image, since for image mosaicing applications between various images the overlapping area between them might be small.

In order to satisfy the above mentioned requirements, it has been applied a novel adaptive non-maximal suppression (ANMS) (Brown et al., 2005) strategy which selects a fixed number of interest points from each image.

In this proposed algorithm feature points are suppressed based on the corner strength f_{HM}, and just the ones that are a maximum in a neighbourhood of radius r pixels are kept.

Theoretically, the proposed algorithm is initialized with the suppression radius r = 0, and then, this value is increased until the wished number of feature points n_{ip} is obtained.

This step can be performed without search as the set of feature points that are generated by this method conform an ordered list. The first entry in this list will be the global maximum, that is, the one that is not suppressed at any radius. The following interest points will be introduced to the list as the suppression radius is reduced from infinity to lower values. Anyway, taking into account that if a feature point is maximum in radius r, then, it will also be a maximum in any $r_j < r$, it can be affirmed that once an interest point appears, it will always stay in the list.

In the implementation, this algorithm is robustified demanding that a neighbour must have certain sufficient larger strength.

So, the minimum suppression radius r_i is defined as:

$$r_i = \min_j |x_i - x_j|, \quad s.t. f(x_i) < c_{robust} f(x_j), \quad x_j \in \Gamma \tag{3.14}$$

Where:

- x_i: The vector representing the position of the feature point in the image

- Γ: represents the set of all feature point positions

It is defined $c_{robust} = 0.9$, in order to guarantee that a neighbour has significantly higher strength for suppression to happen.

In the developed image mosaicing algorithm in Matlab, the adaptive non-maximal suppression technique has been applied by means of the Matlab function named as: *anms.m*. This function has the following input and output arguments:

Input arguments:

- corner_img: an HxW matrix representing the gray corner metric matrix of the input image. It is the output argument of the *corner_detector.m* function.

- max_pts: number of corner points desired from the image. It corresponds with the n_{ip} value defined previously in the explanation of the ANMS technique.

Output arguments:

- x: Column matrix with the x coordinates of the n_{ip} corner points.

- y: Column matrix with the y coordinates of the n_{ip} corner points.

- r_max: value of the suppression radius used to obtain n_{ip} corner points.

The n_{ip} value can be changed depending on the image that it is being analyzed. In this project its value has been defined as n_{ip} = 300. Therefore, from the adaptive non-maximal suppression algorithm they are selected the 300 feature points with the largest values of r_i.

3.1.3 Feature descriptor

40x40 patch descriptor:

Once it has been determined where to place the interest points, a description of the local image structure that will support reliable and efficient matching of features across images should be extracted. First, a 40x40 patch descriptor will be chosen around each corner point.

Geometric blur technique:

Secondly, in this code it will be applied the geometric blur technique explained in (Berg and Malik, 2001) to each patch in order to obtain a $64 \times n$ matrix with the column i being the 64 dimensional descriptor computed at location (x_i,y_i) in the corresponding image.

This technique is based on trying to find point similarities between two images by applying "geometric blur" to both of them.

Habitual methods to solve the correspondence problem consist on some kind of template matching, comparing image windows (patches) that are centered at the two possibly corresponding points.

The essential part of this method is that before performing the comparison images are decomposed into channels of feature responses. This is useful as it permits to separate the uncertainty about the location of the feature points from the uncertainty about their aspect.

In order to perform the matching between two hypothetical corresponding points, it is necessary to consider image windows around these points and then calculate some measure of similarity between the obtained windows. The difficult part here is to be able to obtain "discriminative" windows and a "robust" matching at the same time. If the windows have a small size they are not discriminative, because they do not consider enough context. However, it they are too big, the total change in the windows will be big from different camera views.

The usual method in computer vision to perform the matching between the above mentioned windows is to use a pyramid-like coarse-to-fine approach. This consists on having at a coarse scale of the pyramid, a blurred version of the image at a fine scale. Using this technique, all the pixels from a certain window around a feature point have obtained the same uncertainty about the location by an amount related to the Gaussian standard deviation (σ).

Supposing that the theoretically corresponding points are in the center of the windows, it should be no positional uncertainty at the central pixel of the patch, and increasing uncertainty level related to more external features. However, the standard methodology introduces uniform positional blur in the window. Therefore, is not the most appropriate one as it is been introduced more positional uncertainty than the one required in the central area of the window and maybe less positional uncertainty than the one required for the external area of the window.

The advantage of using the geometric blur technique is that the applied blur to the pixels of the image is proportional to the distance of each pixel to the origin, considering the origin the central pixel of the patch.

In the developed image mosaicing algorithm in Matlab, this step will be performed by means of the Matlab function named as: *geo_blur.m*. This function has the following input and output arguments:

Input arguments:

- image_i: d×d matrix representing the 40x40 gray squared feature patch

Output arguments:

- blurred: d×d matrix representing the 40x40 geometric blurred feature patch

In this function the blur is generated by means of the *B=imgaussfilt(A,sigma)* function that is already defined in Matlab. This function filters image A with a 2 dimensional

Gaussian smoothing kernel with a standard deviation defined by the value of sigma. The bigger the sigma value, the blurrier the image.

Downsample to 8x8:

Once the 40×40 geometric blurred feature patch descriptor has been obtained it will be downsampled to 8×8.

Normalization:

Finally, the feature patch will be normalized in order to have a mean of 0 and a standard deviation of 1.

In the developed image mosaicing algorithm in Matlab, the whole feature descriptor step will be performed by means of the Matlab function named as: $feat\_desc\_geoblur.m$. This function has the following input and output arguments:

Input arguments:

- image_i: H×W matrix representing the gray scale matrix that it is being analyzed.
- x_corners_i: n×1 column matrix with the x coordinates of the corner points. It corresponds with the *x* value that was the output from the *anms.m* function defined previously.
- y_corners_i: n×1 column matrix with the y coordinates of the corner points. It corresponds with the *y* value that was the output from the *anms.m* function defined previously.

Output arguments:

- descriptor: a 64×n matrix of double values with column *i* being the 64 dimensional descriptor computed at location (x_i,y_i) in image_i.

The *feat_desc_geoblur.m* function can be explained by means of the following pseudo-code:

Pseudocode of the "feat_desc_geoblur.m" function

N: number of corner points

for i=1 to N **do**

1. A 40×40 feature patch it is obtained around each corner point.
2. A 40×40 geometric blurred feature patch is obtained around each corner point using the *geo_blur.m* function.
3. The 40×40 patch descriptor is downsampled to 8x8
4. The 8×8 patch descriptor is normalized in order to have a mean of 0 and a standard deviation of 1.
5. The obtained 64 dimensional descriptor around the *i* corner point will be saved in the *i* column of the output matrix named *descriptor*.

end

3.1.4 Feature matching. Computing homography

Given multi-scale oriented patches extracted from all n images. The goal of the matching step is to find geometrically consistent feature matches.

In the developed image mosaicing algorithm in Matlab, this step will be performed by means of the feature_match.m function. This function has the following input and output arguments:

Input arguments:

- descriptor1: a $64 \times n_1$ matrix of double values with column *i* being the 64 dimensional descriptor computed at location (x_i,y_i) in image_1. It corresponds with the

output descriptor value obtained from applying the feat_*desc_geoblur.m* function to image_1.

- descriptor2: a $64 \times n_2$ matrix of double values with column i being the 64 dimensional descriptor computed at location (x_i,y_i) in image_2. It corresponds with the output descriptor value obtained from applying the $feat\_desc\_geoblur.m$ function to image_2.

Output arguments:

- match is $n_1 \times 1$ vector of integers where match(i) points to the index of the descriptor in "descriptor2" that matches with the descriptor "descriptor1(:,i)". If no match is found, match(i) = -1.

The objective will be to find a set of candidate feature matches using an approximate nearest neighbour algorithm. In this case it will be used the "KDTreeSearcher" function already defined in Matlab.

"KDTreeSearcher" is used to store results of a nearest neighbor search using the Kd-tree algorithm. This algorithm separates an $n \times K$ data set by recursively dividing n points which are in K-dimensional space into a binary tree. This is done by executing the following expression in Matlab:

$$kdtree = KDTreeSearcher((descriptor2)')$$

where (descriptor2)' is the transpose of "descriptor2".

Once the KDTreeSearcher model object is created, the stored tree is examined in order to find all neighboring points to the query data by performing a nearest neighbor search using "knnsearch".

"[INDEX,D] = knnsearch(X,Y,K)" finds the nearest neighbor in X for each point in Y, being X an $M_X \times N$ matrix and Y an $M_Y \times N$ matrix. Rows of X and Y correspond

to observations (in this case they will be the 64 dimensional descriptors of each feature point) and columns correspond to variables (which will correspond with different feature points in this algorithm). INDEX is a column vector with M_Y rows, and each of these contains the index of the nearest neighbor in X for the corresponding row in Y.

Here are some remarks about the "knnsearch" function:

- 'K': It is a positive integer that specifies the number of nearest neighbors in X to find for each point in Y. If no value is inserted, 1 is chosen by default.
- INDEX and D are $M_Y \times K$ matrices.
- D sorts the distances in each row in ascending order.
- Each row in INDEX contains the indices of K closest neighbors in X corresponding to the K smallest distances in D.

In the programmed code, it has been defined K=2, therefore, they will be found 2 nearest neighbors in X for each point in Y.

$$[idx \;\; D] = knnsearch(kdtree,\; desc',\; 'K',\; 2)$$

Where "desc" takes in each iteration a 64 dimensional descriptors of a feature point from the variable "descriptor1". So, it will be a 64×1 dimensional column vector.

The condition to accept that a match has been found with the nearest neighbor will be given by:

$$SSD\ of\ 1st\ matching\ /\ SSD\ of\ 2nd\ matching < 0.6 \tag{3.15}$$

Where SSD is the Sum of Square Differences.

If a match is found then: match(i) = idx(1).

$$match(i) = idx(1) \tag{3.16}$$

However, if no match is found:

$$match(i) = -1. \tag{3.17}$$

The "knnsearch" function will be executed n_1 times, as in each iteration it considers one different feature point from the variable "descriptor2".

To sum up this section, the *feature_match.m* function can be explained by means of the following pseudocode:

Pseudocode of the "feature_match.m" function

n_1: number of feature points in descriptor1

n_2: number of feature points in descriptor2

match: zeros(n_1,1)

kdtree = KDTreeSearcher(descriptor2')

for i=1 to n_1 **do**

1. desc = descriptor1(:, i)

2. Apply the "knnsearch" function to desc by:
   $[idxD] = knnsearch(kdtree, desc', 'K', 2)$

3. Save the two matches that are obtained. They correspond with the idx(1) and idx(2) columns in the "descriptor2" variable

4. See if the criteria in Equation 3.15 is verified

5. If it is verified, then match(i) = idx(1). If it is not verified, match(i) = -1.

end

3.1.5 Outlier rejection by RANSAC

RANSAC (Random Sample Consensus) is the most habitually used algorithm to perform homography. Homography consists on removing the corners that do not belong to the overlapping area.

The idea of this algorithm is very simple as explained in (Ghosh and Kaabouch, 2016*b*): for an N number of iterations a random sample of 4 correspondences is chosen and a homography matrix H is calculated from those four correspondences. After that, each correspondence will be defined as an inlier or outlier according to their concurrence with the obtained homography matrix. Once all the iterations are performed, the one that had the highest number of inliers is chosen. Finally, H can be recalculated considering only the correspondences that were classified as inliers in that iteration.

When the RANSAC method is applied, there are important issues that need to be considered. On the one hand, it should be chosen a condition in order to classify correspondences as inliers or outliers. On the other hand, it should be chosen also the value of N, that is, the number of times that the algorithm will be run. This number must guarantee that at least one of the samples will have no outliers. In the developed Matlab algorithm, N will be defined as 1000. Then, the algorithm has the following steps:

1. Select N number of iterations.

2. Determine a threshold T: this will be threshold on distance used to determine if the transformed points agree.

3. Choose a random sample of 4 correspondences.

4. Calculate the homography matrix with the chosen sample.

5. Find the number of data items that fit the model, that is, the number of inliers. In order to determine if the transformed points are inliers or outliers it is used the threshold (T) value defined before.

6. It will be defined K as the number of inliers in that iteration.

7. If K is bigger than the ones in previous iterations save that homography matrix.

8. Repeat steps 3-6 N times.

In the developed image mosaicing algorithm in Matlab, the outlier rejection will be performed by means of the Matlab function named as: *ransac_est_homography.m* function, which has the following input and output arguments:

Input arguments:

- x_1, y_1, x_b, y_b: They are the corresponding point coordinate vectors (x, y) of size N$\times$1 of the corner points of the i and the base images respectively. That is, each (x1(i), y1(i)) matches (xb(i), yb(i)) after a preliminary matching.

- threshold: It is the threshold on distance used to determine if transformed points agree.

Output arguments:

- H: It is the 3x3 homography matrix computed in the final step of RANSAC.

- inlier_index: It is the column vector with the indices of the points in the arrays x_1, y_1, x_b and y_b that were found to be inliers.

Computing the plane to plane homography matrix

Now, it will be explained the procedure of computing the homography matrix from source (x,y) to destination (X,Y). Under perspective projection corresponding points are related by:

$$X = Hx \tag{3.18}$$

From equation 3.18 each point correspondence between 2 images provides two linear equations in the H matrix elements. Therefore, for n correspondences it is obtained a system with 2n equations in 8 unknowns. So, if n = 4 then an exact solution is achieved. However, if n $>$ 4, the matrix is over determined and therefore, H would need to be calculated by an adequate minimization scheme.

It should be taken into account that the covariance of the estimated homography matrix H is influenced by two different factors: the errors in the position of the points used for its calculation and the estimation method used.

There are three standard methods for estimating H:

- Non-homogeneous linear solution
- Homogeneous solution
- Non-linear geometric solution

Homogeneous estimation method

If the homography matrix H is written in the following vector form:

$$h = (h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})^T \tag{3.19}$$

the homogeneous equation 3.18 for n points becomes Ah=0, being A the 2n x 9 matrix

defined as:

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 & -X_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1Y_1 & -y_1Y_1 & -Y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -y_2X_2 & -X_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2Y_2 & -y_2Y_2 & -Y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nX_n & -y_nX_n & -X_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -x_nY_n & -y_nY_n & -Y_n \end{bmatrix} \tag{3.20}$$

It is a standard result of linear algebra that the vector h that minimizes the algebraic residuals |Ah|, subject to |h|=1, is obtained by the eigenvector of least eigenvalue of A^TA. It is possible to obtain directly the value of this eigenvalue from the SVD of A. This is performed by the "svd" function that it is already defined in Matlab. In the case of n = 4, h is the null-vector of A and the residuals are zero.

In the developed image mosaicing algorithm the function that calculates H given 4 correspondences is called *est_homography.m*. This function has the following input and output arguments:

Input arguments:

- X,Y: They are the coordinates of the destination points, which in this case correspond to the ones in the base image. Each variable is a column vector of length 4.

- x,y: They are the coordinates of the source points, which in this case correspond to the ones in the base image. Each variable is a column vector of length 4.

Output arguments:

- H: It is the homography matrix that verifies Equation 3.18

3.1.6 Image homographic warping and stitching

Image warping is the step in which the image is digitally manipulated such that any shapes portrayed in the image are significantly distorted. So, warping is performed in order to correct image distortion. In this step, images are aligned into a common coordinate system using the previously calculated geometric transformations (H).

First, the size of the output mosaic is calculated by computing the range of warped image coordinates for each input image. This can be performed easily by mapping four corners of each source image forward and calculating the minimum and maximum x and y coordinates in order to establish the size of the output mosaic.

Then, the x-offset and y-offset values which determine the offset of the reference image origin relative to the output panorama are computed.

The last step is to use the inverse warping for mapping the pixels from each input image to the plane defined by the reference image. Both images are stitched by merging pixel values of the overlapping regions and retaining the pixel values where there is no overlap.

3.1.7 Image blending

Image blending is the final step of image mosaicing algorithms. In this step, pixels colours of the overlapped region are blended in order to avoid seams. As it has been explained in the literature review there are different techniques to perform the blending. However, in this project is has been chosen the feathering-based one because it is easy to implement, fast and it performs well under exposure differences. This technique uses weighted averaging colour values to blend the pixels of the overlapped regions.

Usually it is defined an alpha factor which has its highest value in the centre pixel and becomes zero after decreasing linearly to the border pixels in the image. In the developed algorithm in Matlab a distance map is computed in terms of Euclidean distance of each

valid pixel from its nearest invalid pixel. This is performed by using the *dist2border* function defined inside the function *mymosaic.m*. This function uses the *bwdist* function which is already defined in Matlab.

Therefore, the values of the pixels that are located in the overlapped regions between two images will be calculated as follows. Given two photos Img_1 and Img_2 with some overlapping in the output mosaic, each pixel (x,y) in photo Img_i is represented as:

$$Img_i(x,y) = (\alpha_i R,\ \alpha_i G,\ \alpha_i B,\ \alpha_I) \tag{3.21}$$

where (R,G,B) are the color values at the pixel.

So, it will be computed the pixel value in the position (x, y) in the output mosaic as:

$$Value(x,y) = \frac{(\alpha_1 R,\ \alpha_1 G,\ \alpha_1 B,\ \alpha_1) + (\alpha_2 R,\ \alpha_2 G,\ \alpha_2 B,\ \alpha_2)}{\alpha_1 + \alpha_2} \tag{3.22}$$

In the code, it will be defined a parameter p as follows:

$$p = \frac{dist2border(Img_1)}{dist2border(Img_1) + dist2border(Img_2)} \tag{3.23}$$

Finally, developing Equation 3.22 the following expression that calculates the pixel values of the overlapped regions is obtained:

$$\begin{aligned} Value(x,y) &= \frac{\alpha_1 Img_1 + \alpha_2 Img_2}{\alpha_1 + \alpha_2} \\ &= \frac{\alpha_1}{\alpha_1 + \alpha_2} Img_1 + \frac{\alpha_2}{\alpha_1 + \alpha_2} Img_2 \\ &= p * Img_1 + \frac{\alpha_2 + \alpha_1 - \alpha_1}{\alpha_1 + \alpha_2} * Img_2 \\ &= p * Img_1 + (1-p) p * Img_2 \end{aligned} \tag{3.24}$$

3.2 Mobile tasking

The efficient cooperation of multiple UAVs, which is defined as task allocation, is a fundamental part in order to achieve the mission success. In this research, the mission scenario is an area of high interest in which certain waypoints or tasks have been placed.

Therefore, the aim of the sensor placement problem will be to decide where the sensors should be placed, taking into account that there are M candidate locations, and N<<M sensors. So, each UAV or sensor platform should have allocated some positions in the mission scenario where they should go and take a photograph. In addition, once that each sensor has allocated some positions it should be selected the sequence in which the tasks should be visited by each UAV in order to minimize the travelled distance.

Therefore, the task allocation algorithm can be divided in three steps:

1. Select the most appropriate tasks in order to maximize the amount of information obtained by the sensors in the shortest possible time. This will be performed using a Greedy algorithm which is based, as it will be explained later, in maximizing a submodular function f.

2. Once the required tasks are selected, they should be divided in N clusters, being N the number of UAVs that are being used.

3. Finally, the task sequence inside each cluster should be chosen, with the condition of minimizing the travelled distance by each UAV.

3.2.1 Greedy algorithm

The essential part of Greedy algorithms is to define properly the objective function *f*. In this case, this objective function will be the fundamental part of the integration between the mosaicing and the mobile tasking parts, and therefore, it will be necessary to develop a submodular function which can reflect the performance of mosaicing.

In this study, the objective connected to the purpose of the study is to maximize the amount of information obtained by the images, as this will provide a complete output mosaic with no missing parts.

First, it should be defined what a submodular function f. Submodular functions are functions that exhibit a natural diminishing returns property, which means that the marginal benefit of any given element decreases as more elements are selected. Defining it mathematically, a submodular function $f : 2^V \rightarrow \mathbf{R}$ assigns a subset $A \subseteq V$ a utility value $f(A)$ such that:

$$f(A \cup \{i\}) - f(A) \geq f(B \cup \{i\}) - f(B) \tag{3.25}$$

for any $A \subseteq B \subseteq V$ and $i \in V \setminus B$. Where V is called the ground set.

This definition expresses that adding an element i to a subset A of set B yields at least much value (or more) as if i is added to B. In other words, the marginal gain of adding i to A is bigger or equal to the marginal gain of adding i to B.

In this project, the area of interest will be divided in squares as it can be seen in Figure 3.3, being the corners of these squares the tasks or waypoints where the UAVs could take a picture.

So, in the problem they will be considered a fixed number of UAVs, that will be defined as N, and M tasks.

As it has been mentioned before, in order to apply a Greedy algorithm it is important to define a suitable submodular objective function. In this case, the objective will be defined as maximizing the coverage area obtained by all the images, because logically the bigger the coverage area, the more information will be obtained, and therefore, a better final mosaic will be achieved.

It will be assumed there is a $d_{waypoints}$ distance between waypoints, and that when a UAV takes a picture in the position (x, y) it is covering a square area of of side $2 \times d_{waypoints}$.

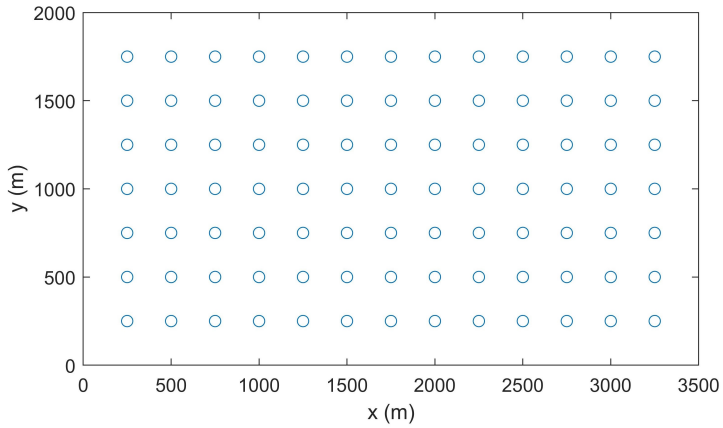


Figure 3.3: Tasks in the area of interest

However, this will not be completely true, since in reality it is covering a bigger size, which is necessary in order to guarantee overlapping regions between the images.

In order to explain the methodology of this step it will be assumed there is a small number of UAVs, for example N = 4, and that all the squares that appear in Figure 3.3 refer to areas that should be covered by the UAVs.

In order to achieve the objective in an optimal way, the following Greedy algorithm will be executed for maximizing the coverage area.

Pseudocode of the Greedy algorithm for maximizing the coverage area

Input: t_1, t_2, t_3,..., $t_M \subseteq$V

Output: S (Selected tasks)

begin

- S $\leftarrow$ Ø It begins with an empty set S.
- V′ = V = $\{t_1, t_2, t_3, ..., t_M\}$
- **while** there exists $t_i \in V'$ such that $\Delta f(t_i \mid S) > 0$ **do**
  (While the marginal gain of adding one element is greater than zero)

- Let t_i be the element of V' maximizing $\Delta f(t_i \mid S)$

  $t_i = \underset{t_i}{arg\ max}\ \Delta f(t_i \mid S)$

- Add t_i to S

  S = S $\cup$ $\{t_i\}$

- Delete t_i from V'

  V' = V' - $\{t_i\}$

- **end**

So, as it can be seen in the above pseudocode the algorithm has only one input argument, which is a set V with all the possible waypoints in which the UAV can take an image. Then, the algorithm starts with an empty set S, and in each iteration it adds the element that maximizes the objective function f while the marginal gain of adding one element is greater than zero. Therefore, this algorithm will stop once all the area of interest is covered by the drones.

Finally, after running the Greedy algorithm for the above case the following tasks that appear in Figure 3.4 are selected.

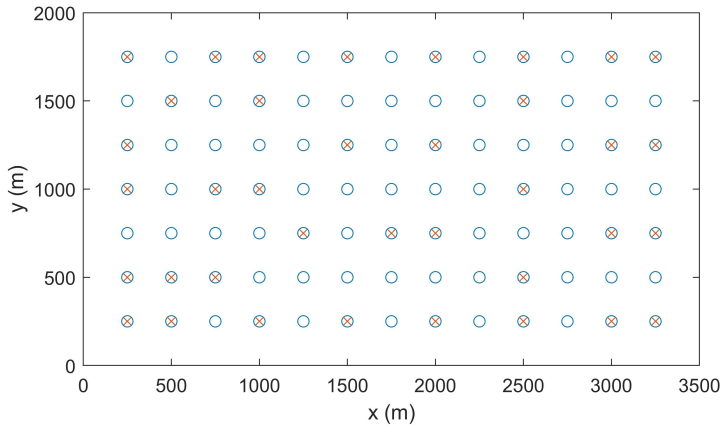


Figure 3.4: Tasks in the area of interest

3.2.2 Clustering

Once all the tasks have been selected, each UAV should be assigned a certain number of tasks. Obviously, this will not be performed randomly, as it is required the mission to last the shortest possible time. In order to achieve this objective, clustering will be a good alternative.

In this project, this step will be performed using K-means clustering. K-means clustering is a kind of unsupervised learning, which is useful when there is some unlabeled data that want to be separated in different groups. So, the objective of this algorithm is to find K groups in the given data. After applying this technique:

1. Each data point will be assigned to a single cluster

2. Each cluster will have a fixed centroid

One of the important parameters when applying this algorithm is to choose the value of K. However, in the developed task allocation algorithm in Matlab K will coincide with the number of UAVs, that is, with N.

The K-means clustering algorithm uses an iterative method to find the final result. The input arguments to the algorithm will be:

- The data set: It will be a list with the coordinates (x,y) of all the tasks that have been selected in the previously run Greedy algorithm.

- Number of clusters K: As it has been mentioned, this will be the same as N.

Now, they will be explained the different steps of this clustering algorithm. First, they are randomly generated the centroids of the K clusters, which will be named as: μ_1, μ_2, μ_3,..., μ_K. However, the result of this algorithm will be influenced by the initial locations of the clusters' centroids. That is the reason why in the developed Matlab algorithm it has been applied a method to select those initial positions, with the objective of achieving good results at the end.

Therefore, the following steps are performed to initialize the K cluster centroids:

1. $\theta = \frac{2\pi}{K}$
2. Define a point $(x_{ref}$,$y_{ref})$ which is located in the centre of the interest area. That is: $x_{ref} = \frac{x_{length}}{2}$ and $y_{ref} = \frac{y_{length}}{2}$.
3. Define a radius $r = \frac{\sqrt{(x_{ref})^2+(y_{ref})^2}}{3}$
4. **for** i=1 to K **do**

   $x_{centroid}(i) = x_{ref} + r * cos\frac{(2i-1)\theta}{2}$

   $y_{centroid}(i) = y_{ref} + r * sin\frac{(2i-1)\theta}{2}$

   end

After that, the algorithm iterates between the two following steps until a defined stopping criteria is met.

1. Assign data to clusters

In this step, taking into account that each cluster is defined by the position of its centroid, each task will be allocated to its nearest cluster centroid. More formally, considering there are m data points, for each of them it should be found the k_i cluster that minimizes the Euclidean distance between the task position and the cluster centroid, that is:

$$\min_{k_i} ||x_i - \mu_k|| \qquad \text{for i=1, 2, 3,...,m} \tag{3.26}$$

2. Update the positions of the centroids

The centroid of each cluster is recalculated. This is performed by computing the mean of the data points that have been assigned to each cluster. So, assuming that S_k is the set of points that have been assigned to cluster k:

$$\mu_k = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad x_i \in S_k \tag{3.27}$$

where n_k are the number of points that have been assigned to cluster k.

So, the K-means clustering algorithm will go iterating between the previous two steps until the stopping criteria is verified. In this case, the algorithm will stop if no cluster changes its data points after one of the iterations.

For the example it is being analyzed above they were already selected the tasks that should be performed. So, the first step for the clustering part was to compute the initial centroids of the clusters, which allows to obtain the following positions indicated in Figure 3.5.

Finally, the K-means clustering algorithm is run, which will provide the final result after two iterations as it can be seen in Figures 3.6 and 3.7.

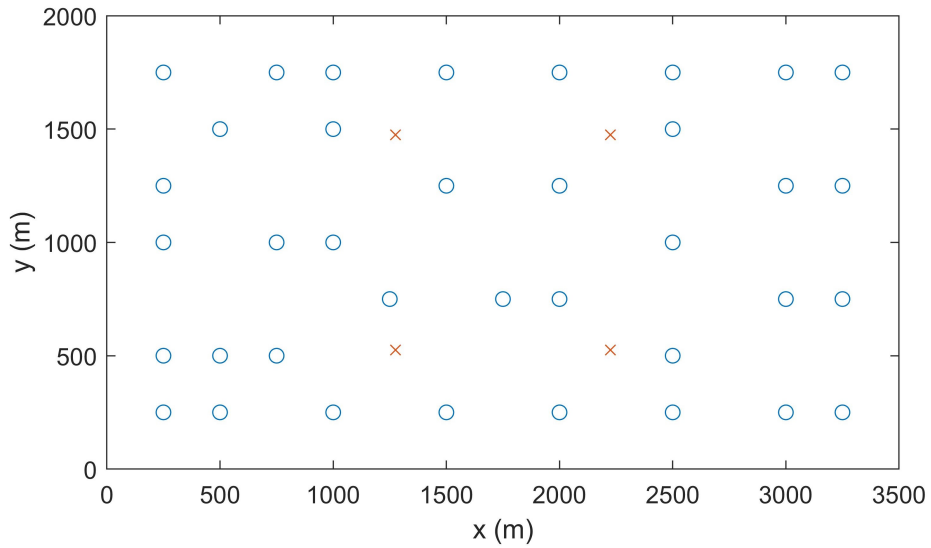


Figure 3.5: Initial centroid positions

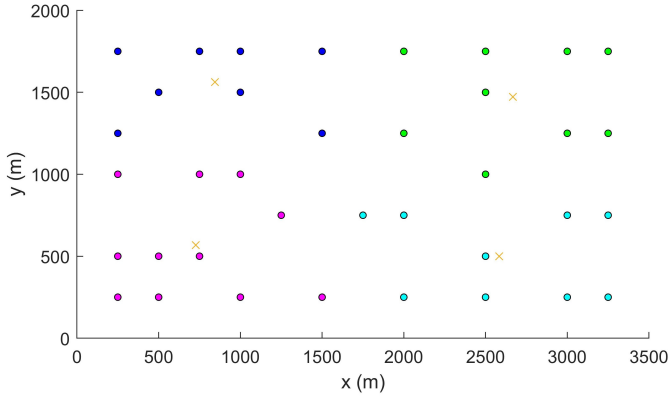


Figure 3.6: Task distribution among clusters after the first iteration

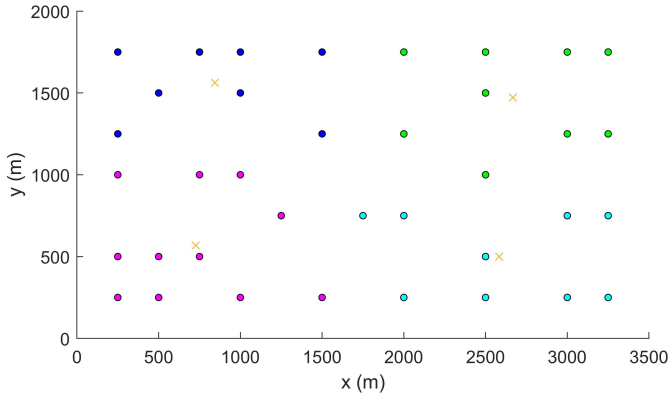


Figure 3.7: Task distribution among clusters after the second iteration

3.2.3 Task sequence selection

Once the task allocation issue is performed each UAV has a set of tasks to perform, that is, a set of positions to visit. As the endurance of the UAVs is limited, it will be necessary to travel to all these positions trying to minimize the total travelled distance by each drone.

This problem will be faced as it was mentioned in the literature review as a case of the Travelling Salesman Problem (TSP). In the developed mobile tasking algorithm in Matlab it has been programmed a code based on the Held-Karp algorithm (Held and Karp, 1961). This algorithm solves the TSP using dynamic programming (DP). Therefore, it is

guaranteed to provide an optimal result, but the number of tasks will be limited as the time complexity of the algorithm is $O(2^n n^2)$.

In the developed Matlab algorithm the function that solves the TSP is called $tsp.m$. This function has the following input and output arguments:

Input arguments:

- tasks: It is an n$\times$2 matrix with the locations of the tasks. These tasks will all correspond to the same cluster.

Output arguments:

- optimaltour: It is a vector with the indices of the tasks, according to the optimal route to perform all the tasks.

- mindistance: The total length of the optimal route.

The Held-Karp algorithm applies the following property of the TSP: *Every subpath of a path of minimum distance is itself of minimum distance.*

Therefore, when trying to find the shortest path, instead of looking at every possible permutation using a simple ”top-down” method, it is used a ”bottom-up” method. The advantage of this technique is that all the intermediate information which is necessary in order to solve the problem will be calculated once and only once.

So, the first step will be the smallest subpath. Then, at each step a larger subpath is solved, making it possible to look up the results of all the smaller subpath problems that have been previously calculated.

Hence, the time required to solve the problem is reduced because it has already been found a solution for all the smaller subpaths and these savings compound exponentially (at each higher subpath level). However, the main disadvantage is that a large memory is

required to solve the problem, as all the intermadiate path results need to be saved until the end.

Here it is a pseudocode of the algorithm it has been implemented in Matlab that was taken from Wikipedia.

Where:

It is assumed that the route begins at task 1.

$d_{i,j}$: distance between task i and task j

n: number of tasks

C(S,k): minimum distance starting in task 1, visiting all the tasks in S and finishing in task k

function algorithm for the TSP (G, n)

for k := 2 to n do

C({k}, k) := $d_{1,k}$

end for

for s := 2 to n-1 do

for all S $\subseteq \{2, \ldots, n\}$, $|S|$ = s do

for all k $\in S$ do

$C(S,k) := \min_{m \neq k,\ m \in S} [C(S \backslash \{k\},\ m) + d_{m,k}]$

end for

end for

end for

$opt := \min_{m \neq 1} [C(\{2,3,...,n\},\ k) + d_{k,1}]$

return(opt)

end

Finally, the TSP algorithm will be run for the example it has been considered. So, after running the *tsp.m* Matlab function N times, that is, one per cluster, the following routes are obtained for each of the UAVs:

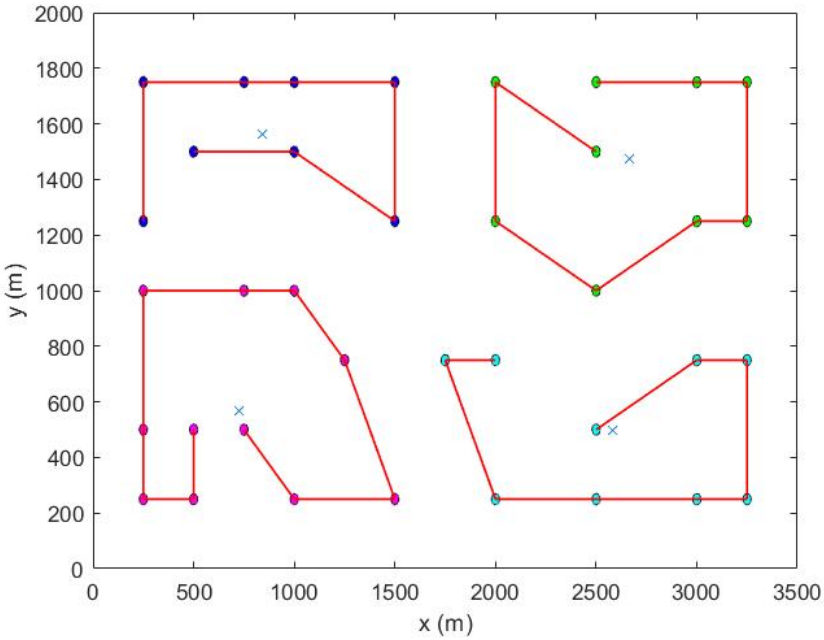


Figure 3.8: Optimal task sequence for each UAV

Chapter 4

Results and Analysis

In this chapter the procedure that has been followed to validate each part of the project will be explained. In order to do that, the obtained results will be shown and analyzed.

4.1 Image mosaicing algorithm

First, the performance of the image mosaiciing algorithm will be validated. This can be easily done by proving that the algorithm returns a well built final mosaic when a number of images of the same scene with some overlapping areas are introduced as the input data of the algorithm.

The algorithm was run first with the following input images of Figure 4.1. As it can be seen, all the images correspond to the same scene and were taken from the same point but pointing to different directions. Therefore, once they are stitched they should complete a panoramic image.

After running the algorithm the output mosaic of Figure 4.2 is obtained. As it can be seen all the feature points of each image must have been well identified and matched with their corresponding points of other figures, as the images have been perfectly stitched among them. Therefore, as this mosaic represents perfectly the complete scene it has been demonstrated that the image mosaicing code works properly with this kind of images.

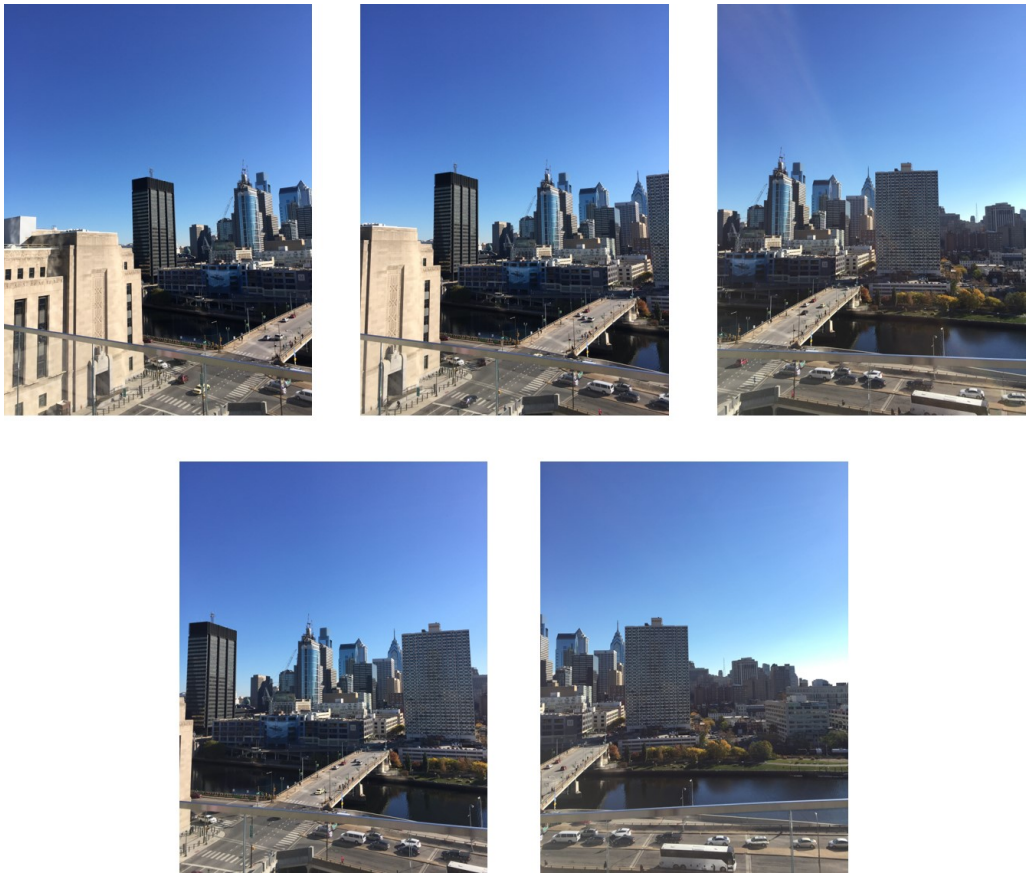


Figure 4.1: Input images to the image mosaicing algorithm

Figure 4.2: Obtained output mosaic from the image mosaicing algorithm

The above considered images have a lot of features points, as the image shows different buildings with very noticeable corners on them. This fact makes it easier for the algorithm to find a good final solution. As it has been explained in the introduction, this project is based in the Thales Watchkeeper, which is an unmanned aircraft system designed specially for military purposes. Then, the mission areas in which it operates are not usually the centre of a city. That is the reason why it has been decided to validate the image mosaicing algorithm for images with more sparse features like the ones shown in Figure 4.3.

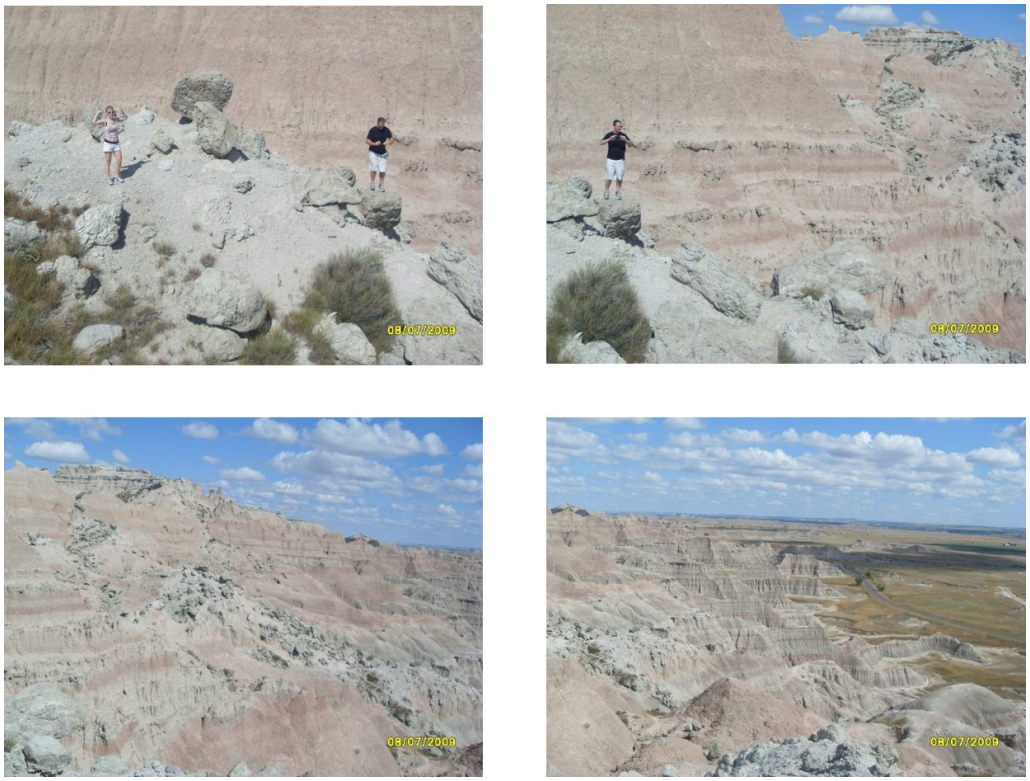


Figure 4.3: Input images (with more sparse features) to the image mosaicing algorithm

Running the algorithm the mosaic which is shown in Figure 4.4 is obtained. In addition, even if this images have much less quality than the ones considered previously, the algorithm has been proven to continue working.

Figure 4.4: Obtained output mosaic from the image mosaicing algorithm (input images with more sparse features)

4.2 Task allocation

4.2.1 Validation of the task selection algorithm

As it has been explained in Chapter 3, given a certain mission area the task allocation algorithm consisted on three steps:

1. Task selection algorithm: A certain number of tasks are selected trying to maximize the coverage area.

2. Task allocation to each UAV

3. Selection of the shortest route to be performed by each UAV

In this section it will be validated the proposed Greedy algorithm for the first step. This algorithm selects tasks with the objective of maximizing the coverage area by all the UAVs. In addition, one of its advantages is that given a certain mission area and a distance between these waypoints, the algorithm is capable of calculating the minimum number of tasks which are necessary to cover all the area.

The validation will be carried out by means of a comparison, that is the performance of the results obtained with the proposed algorithm will be compared by the one obtained

by a randomized approach. That is, in the randomized approach there should be selected the same number of tasks (m) that in the Greedy algorithm, but their positions should be randomly chosen among all the possible tasks. As there is an element of randomness in the simulation, the Monte Carlo method will be used in order to asses its the performance.

Let be assumed the following mission area of Figure 4.5. A number M of tasks have been defined inside that area, being M in this case 91. Therefore an m number of tasks should be selected until all the interest area is covered by the UAVs. After running the proposed Greedy algorithm 37 tasks have turned to be chosen as it is shown in Figure 4.6. So, it can be stated that given that mission scenario it is possible to achieve a complete coverage area by suitably selecting 37 tasks.

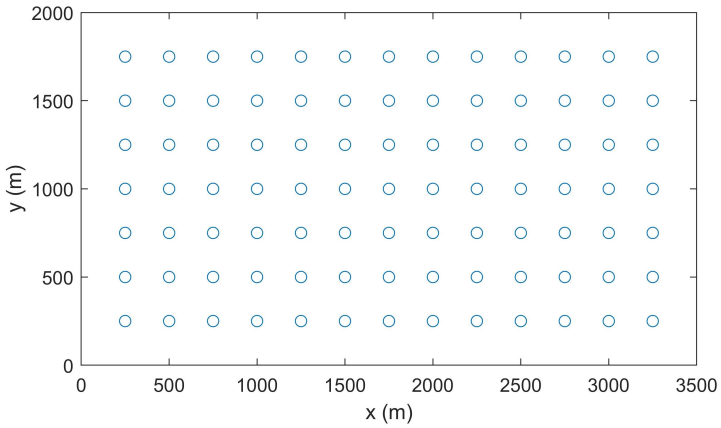


Figure 4.5: Tasks in the area of interest

Now, the following Monte Carlo simulation will be applied. They will be selected as well 37 tasks among all, but in this case their positions will be randomly selected assuming that all the tasks have the same probability to be chosen. Once these tasks have been chosen the percentage of the total mission area they cover, that is, their coverage area, will be calculated. This Monte Carlo simulation has been performed for N=200 different cases, and the following results have been obtained.

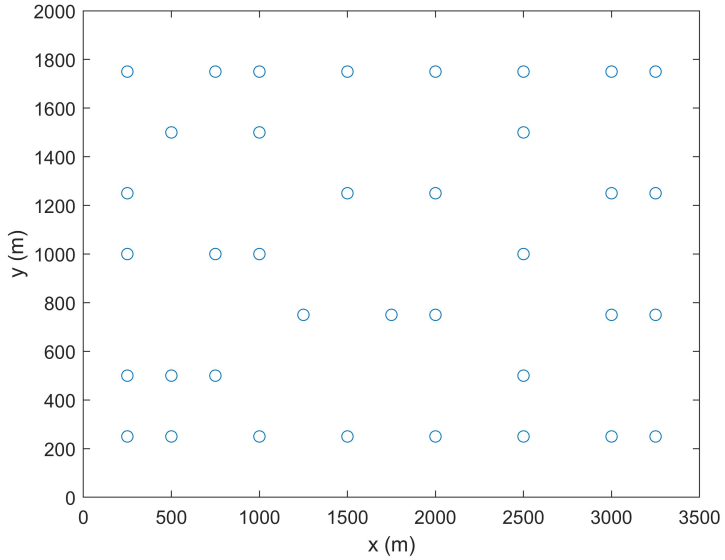


Figure 4.6: Selected tasks in the area of interest

The mean coverage area was resulted to be a 71%, which is much less than the one obtained from the Greedy algorithm. In addition, the following graph was obtained, in which the probabilities of achieving each percentage of covered area are indicated.

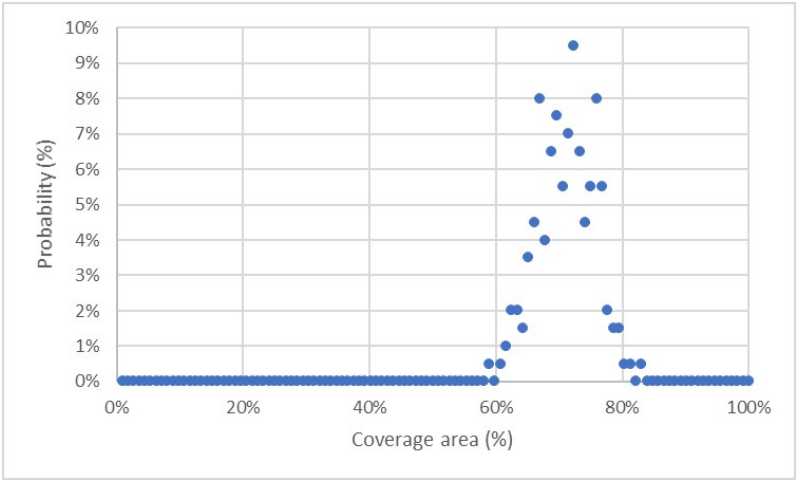


Figure 4.7: Probability of achieving a certain coverage area

As it can be seen in N=200 iterations there was no case in which a 100% coverage area was achieved. The maximum coverage area was resulted to be 83%. So, in the performed Monte Carlo simulation the coverage are was resulted to be between the 59% and the

83%.

It can be concluded then, than the proposed algorithm to select the positions in which the UAVs should be placed performs much better than the approach of randomly choosing these positions.

4.3 Integration of image mosaicing and mobile tasking

4.3.1 Simulation

Finally, the complete project, that is, the integration of image mosaicing and task allocation, should be validated. As there was no time for hardware demonstration with real UAVs, it was decided to validate the proposal by means of a Matlab simulation.

The simulation will be performed in the following way. Let be assumed there is a certain mission area. In this case, the mission area will correspond with an image of dimensions H$\times$W. After running the task allocation algorithm in that image, there will be selected some waypoints among all. In addition, each UAV will have a trajectory assigned, in which it will be indicated which are the positions where it should stop in order to take a picture. So, it will be made the UAVs fly over the image taking pictures in the positions that were selected by the task allocation algorithm. After that, a set of images will be obtained by each UAV which will need to be stitched. In order to do that, first they will be stitched the images obtained by each UAV, that is, all the images that correspond to the same cluster will be stitched together. Once this N mosaics have been obtained, being N the number of UAVs, the image mosaicing algorithm will be run one last time with these N mosaics as input arguments in order to obtain the last complete mosaic. If the simulation performs properly, the obtained last mosaic should coincide with the image that was considered to be the mission area.

Now, different simulations will be done in order to analyze the validity of the project and find the limitations that this has. That is, the drones will need to coordinate in an

optimal way, trying to cover all that area in the shortest possible time.

Simulation 1:

First, the following Notting Hill (London) image will be considered as the "mission area".

Figure 4.8: Mission area for simulation 1

The dimensions of the area that are introduced in the task allocation algorithm correspond with the dimensions of this image. So, in this case, as it is a 609×1200 pixel image, it will be assumed a 600×1200m^2 mission area. As it can be seen, the input dimensions in the task allocation algorithm will be slightly smaller. However, this will suppose no inconvenient because when a UAV takes a picture in a certain point (x, y), the area it is covering in reality is a bit bigger than the one considered in the task allocation algorithm. This fact guarantees that images that are taken in positions that are next to each other will have a certain overlapping. In addition, the inserted dimensions value will also have the advantage of obtaining an easier waypoint distribution later, as it is easier to find a $d_{waypoints}$ value, which is the distance between waypoints which are next to each other, that matches perfectly with dimensions X and Y.

After inserting the above mentioned two values for the X and Y dimensions, it has been defined $d_{waypoints}$ as 100 metres. Therefore, the following waypoint distribution of Figure 4.9 has been obtained in the considered mission area.

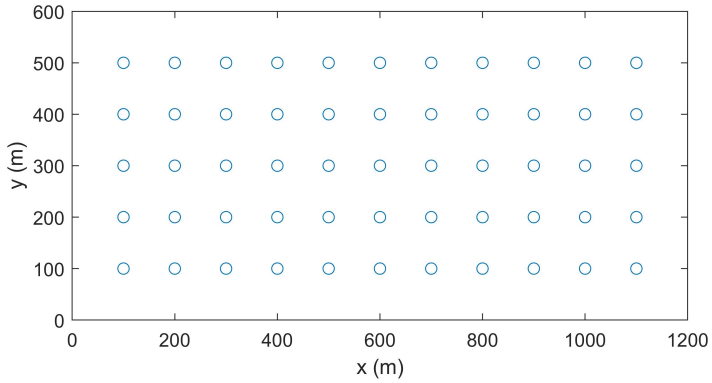


Figure 4.9: Waypoint distribution over the mission area for simulation 1

It will be assumed there are 4 UAVs, that is, N=4. So, at the beginning they will be placed those 4 drones in 4 different positions in the mission area. These will be considered the starting points of each UAV. These positions are indicated in red in Figure 4.10.

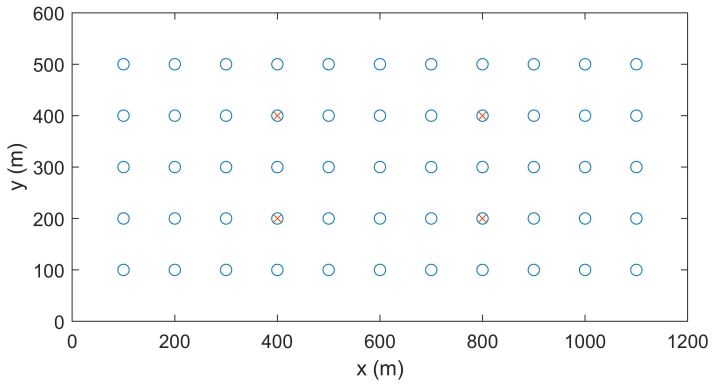


Figure 4.10: Initial UAV positions in the mission area for simulation 1

After that, the developed Greedy algorithm will select the set of tasks that need to be performed in order to cover all the mission area trying to minimize obviously the number of selected tasks. As it has been explained in Chapter 3, the objective function that should be maximized while running the Greedy algorithm is the coverage area. Therefore, in each iteration of this algorithm it is selected the task that maximizes the objective function f while the marginal gain of adding one element is greater than zero. As a consequence of

this last condition, this algorithm will stop once all the area of interest is covered.

Anyway, it should not be forgotten that in order the mosaicing algorithm to work it will be necessary to guarantee an overlapping area between the images that are obtained. This parameter will be analyzed at the end of this validation step, as they will be compared the results obtained with different overlappings.

In this case, after running the Greedy algorithm it is obtained a set of 23 tasks that are indicated in Figure 4.11. This means that the UAV should go to those positions in the mission area and take a photo there.

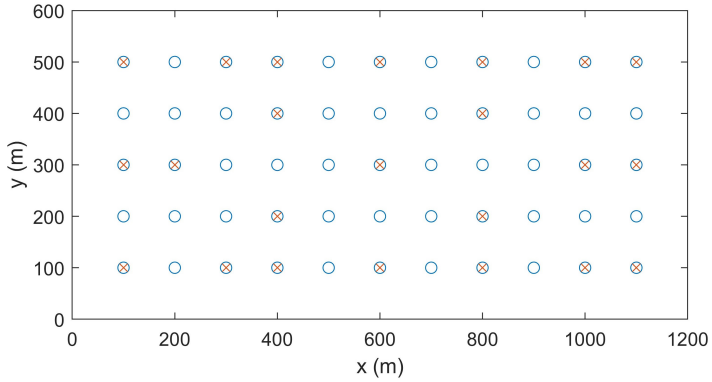


Figure 4.11: Selected tasks by the Greedy algorithm for simulation 1

Once it has been decided which are the positions in which the images should be taken, the next step will be to allocate a set of tasks to each UAV. In this case, as there are 4 drones the tasks that were obtained from the Greedy algorithm should be divided in 4 groups, according to the proximity they have among them. That is, tasks should be divided in 4 clusters applying the K-means clustering algorithm. The initial positions of the clusters were defined as they are indicated in Figure 4.12.

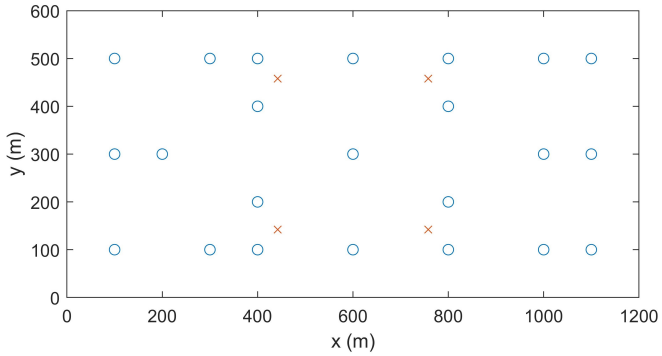


Figure 4.12: Initial positions of the clusters' centroids in simulation 1

As it has been explained previously, the K-means clustering method implies various iterations, until the positions of the centroids do not vary from the obtained ones in the previous iteration. In this case, they have been required 3 iterations until the final cluster distribution is achieved.

In Figures 4.13, 4.14 and 4.15, the task distribution in clusters and their corresponding centroid positions in each iteration can be seen.

In addition, it is clearly shown that Figures 4.14 and 4.15 are exactly the same. This makes sense, because as it was explained earlier, the algorithm finishes when there is no change in the assigned tasks to each cluster from one iteration to another.

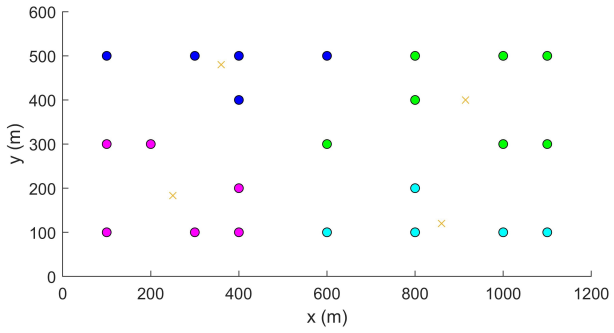


Figure 4.13: Distribution of tasks in clusters (after iteration 1) in simulation 1

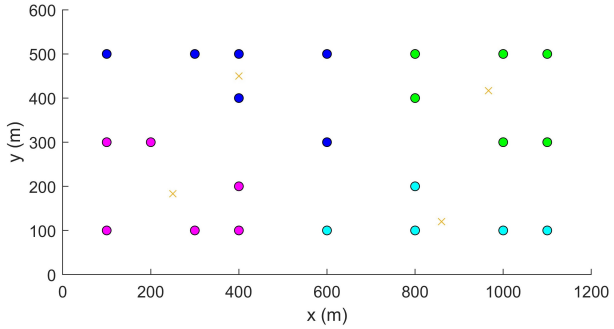


Figure 4.14: Distribution of tasks in clusters (after step 2) in simulation 1

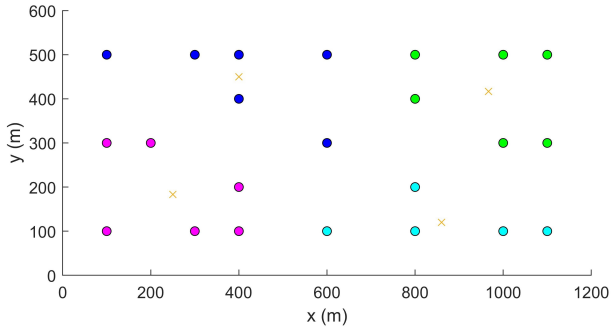


Figure 4.15: Distribution of tasks in clusters (after step 3) in simulation 1

After that, the Held-Karp algorithm is applied in order to obtain the trajectory that each UAV should follow so that the time used by each of them to go over all the tasks is the minimum one. This trajectories are indicated in Figure 4.16

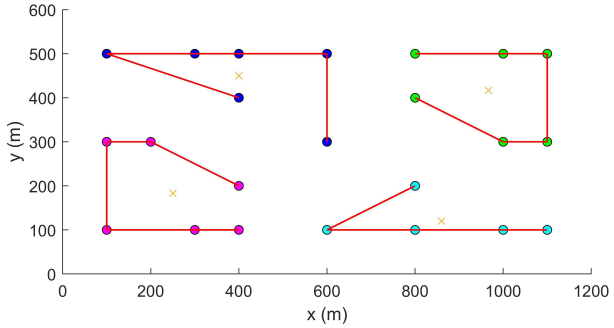


Figure 4.16: Final cluster distribution and minimum path for each UAV in simulation 1

Once all the task allocation algorithm has been run, each UAV will fly over the image of Figure 4.8. Therefore, each selected task will be related with one position in the image. These positions are indicated in Figure 4.17.

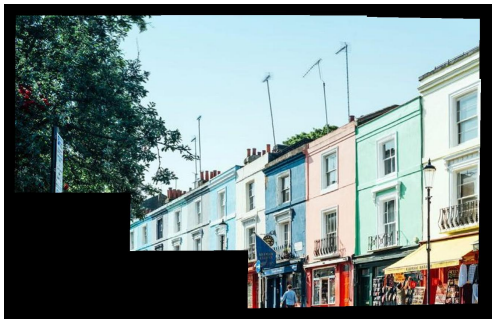
Figure 4.17: Positions in which the UAVs will take a photograph during the simulation 1

So, in this case there will be obtained 23 images (see Appendix A) that should be stitched with the developed image mosaicing algorithm. This will be performed in two different steps:

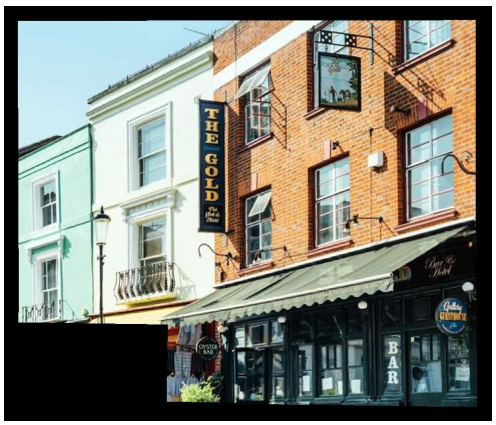
1. It will be formed a mosaic with the corresponding images of each UAV. That is, the mosaicing algorithm will be run N=4 times.

2. Once the N mosaics which correspond to the different clusters are obtained, it will be executed the mosaicing algorithm one last time in order to obtain the final mosaic.

It can be seen in Figure 4.18 the results obtained after the first step, that is, the obtained mosaic for each of the clusters.



(a) Cluster 1 (dark blue)



(b) Cluster 2 (green)

(c) Cluster 3 (pink)

(d) Cluster 4 (cyan)

Figure 4.18: Obtained mosaic for each of the clusters in simulation 1

After performing the second step, the final mosaic of Figure 4.19 has been obtained.

Figure 4.19: Final mosaic in simulation 1 [1]

It should be remarked that this procedure has been developed numerous times with different overlapping areas between images. After performing an analysis about the required overlapping in order to obtain a good final mosaic, it has been concluded that it is required a minimum overlapping of 50% between two images that are located next to each other to guarantee the quality of the final result.

Finally, comparing images from Figures 4.8 and 4.19, it can be affirmed that the obtained final mosaic covers all the mission area, as it is a perfect copy of that Notting Hill street it has been used for the simulation. Therefore, the integration of image mosaicing and task allocation has been validated.

Simulation 2:

Now, it will be carried out a second simulation in order to prove that good results are obtained for a wide variety of images.

The next considered mission area for the simulation, will be the image in Figure 4.20.

[1] Image obtained from https://www.theresident.co.uk/homes-interiors/move-over-shoreditch-notting-hill-is-proving-west-is-best/

This image is very different to the one considered in the previous simulation, as it is a landscape. In addition, as it is a 2108× 3360 pixel image, it will be assumed a 2000× 3500m^2 mission area. So, after inserting these two values for the X and Y dimensions it has been defined the $d_{waypoints}$ value as 250 metres. Therefore, the following waypoint distribution of Figure 4.21 has been obtained in the mission area.

Figure 4.20: Mission area for simulation 2 [2]

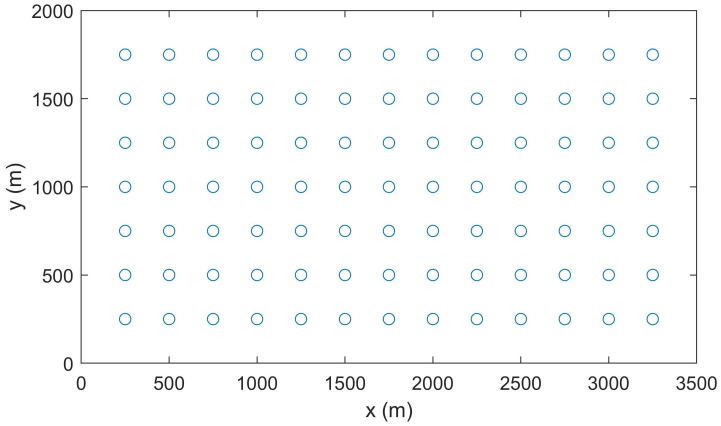


Figure 4.21: Waypoint distribution over the mission area for simulation 2

It will be assumed as well that there are 4 UAVs, that is, N=4. So, at the beginning

[2]Image obtained from https://www.pexels.com/search/landscape/

they will be placed those 4 drones in the following 4 positions in the mission area, which are indicated in red in Figure 4.10.

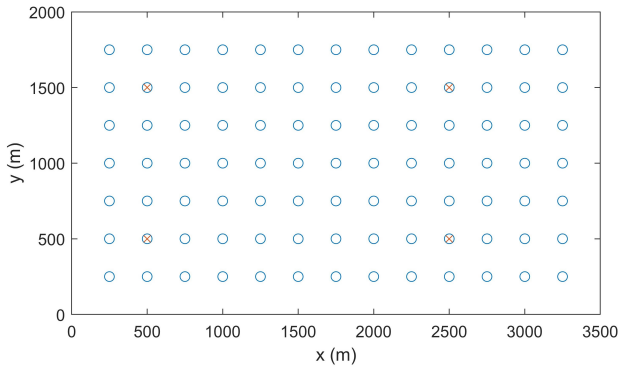


Figure 4.22: Initial UAV positions in the mission area for simulation 2

After that, the developed Greedy algorithm will select the set of tasks that need to be performed in order to cover all the mission area in an optimal way, that is, by using the minimum number of tasks. In this case, it was obtained a set of 37 tasks that are indicated in Figure 4.23.

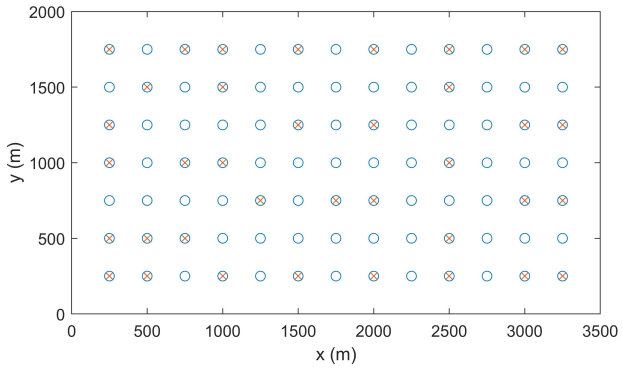


Figure 4.23: Selected tasks by the Greedy algorithm for simulation 2

The next step is to allocate a certain number of tasks to each UAV, which is done using the clustering method. Therefore, as there are 4 UAVs, the previously obtained tasks will be divided in 4 clusters. The initial positions of the clusters were defined as they are indicated in Figure 4.24.

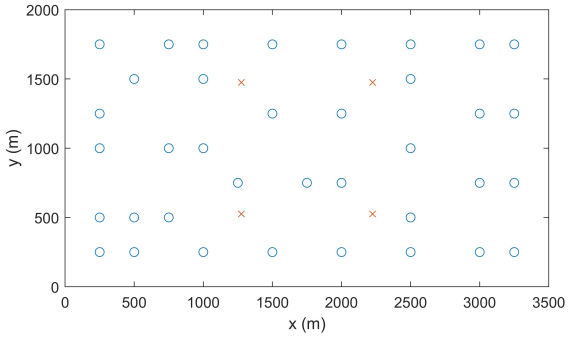


Figure 4.24: Initial positions of the clusters' centroids in simulation 2

In this case, they have been required 2 iterations until the final cluster distribution is achieved. In Figures 4.25 and 4.26 the task distribution in clusters and their corresponding centroid positions in each iteration can be seen.

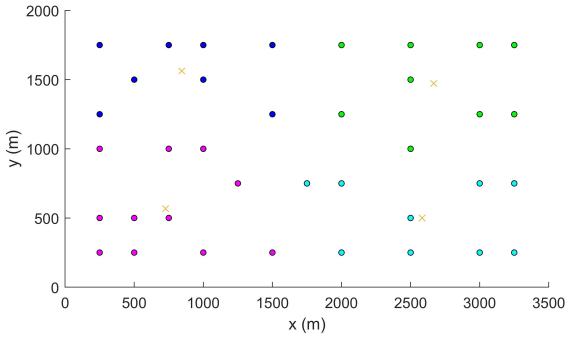


Figure 4.25: Distribution of tasks in clusters (after iteration 1) in simulation 2

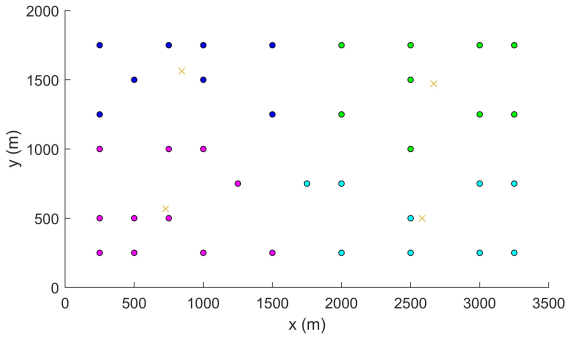


Figure 4.26: Distribution of tasks in clusters (after iteration 2) in simulation 2

To end with the task allocation algorithm, the trajectories that each UAV must perform are calculated with the Held-Karp algorithm. These trajectories are indicated in Figure 4.27.

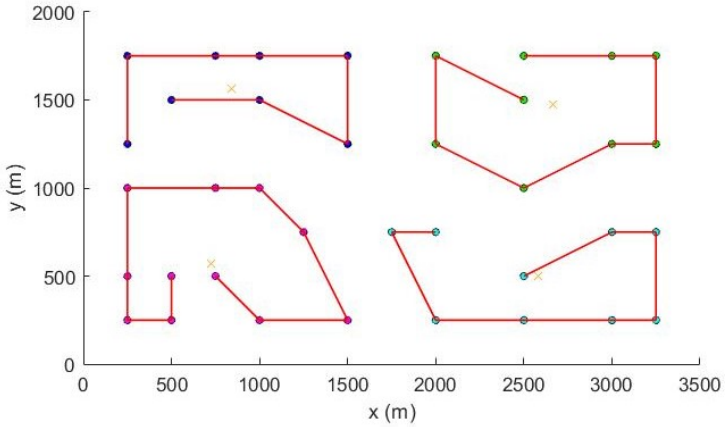


Figure 4.27: Final cluster distribution and minimum path for each UAV in simulation 2

Then, the simulation where the UAVs ”fly” over the image starts. As it can be seen in Figure 4.28, each task will be related with one position in the image.

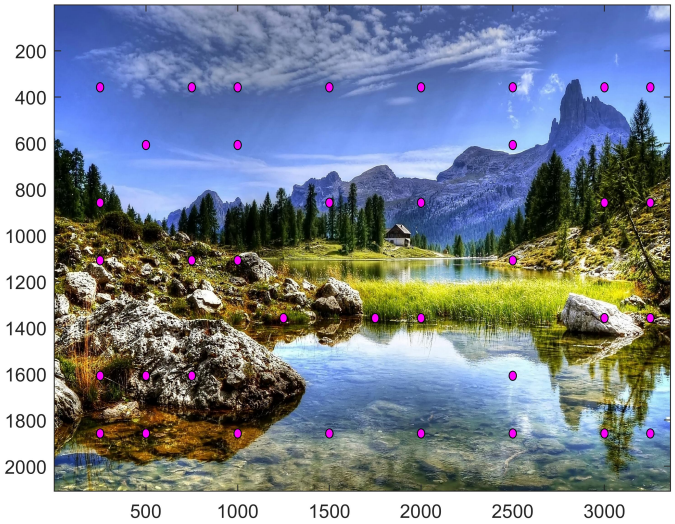


Figure 4.28: Positions in which the UAVs will take a photograph during simulation 2

So, in this simulation they will be obtained 37 images (see Appendix B) that should

be stitched together in order to obtain the final mosaic.

It can be seen in Figure 4.29 the mosaic obtained for each of the clusters, and in Figure 4.30 the final mosaic.

(a) Cluster 1 (dark blue)

(b) Cluster 2 (green)

(c) Cluster 3 (pink)

(d) Cluster 4 (cyan)

Figure 4.29: Obtained mosaic for each of the clusters in simulation 1

Figure 4.30: Final mosaic in simulation 2

Finally, comparing images of Figures 4.20 and 4.30 it can be affirmed that the mosaic represents perfectly the original image, and therefore, the integration between image mosaicing and task allocation has been validated.

4.3.2 Changes in the number of UAVs to deal with broken drones

In this section, it is going to be proved that the system can continue working when for some reason one of the UAVs in unable to contribute. This might happen when a drone gets lost or needs to be charged.

So, let assume the mission area of the first simulation, that is, the Notting Hill image of Figure 4.8. At the beginning it was considered there were N=4 UAVs, but suddenly one of them stops working, and so, it will not be able to perform any task. In this situation, the algorithm is capable of reprogramming everything in order to divide the same selected tasks by the greedy algorithm in (N-1) clusters, that is, in this case they will be divided in 3 clusters.

The procedure followed from this point on is exactly the same to the previous simulations. First, the initial positions of the clusters are defined as they are indicated in Figure 4.31.

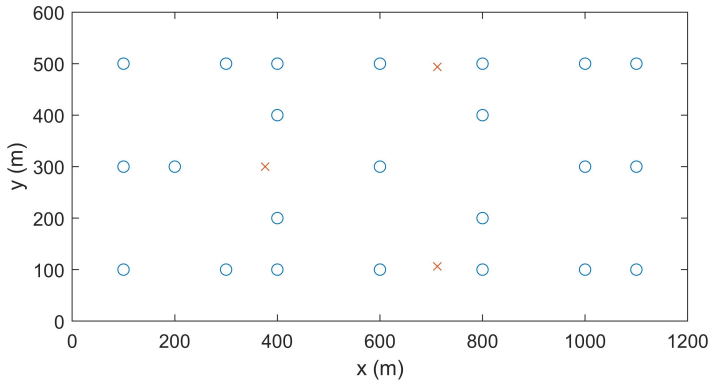


Figure 4.31: Initial positions of the clusters' centroids in simulation 3

After that, the K.means clustering method will be applied. In this case, the final cluster distribution is achieved after two iterations as indicated in Figures 4.32 and 4.33.

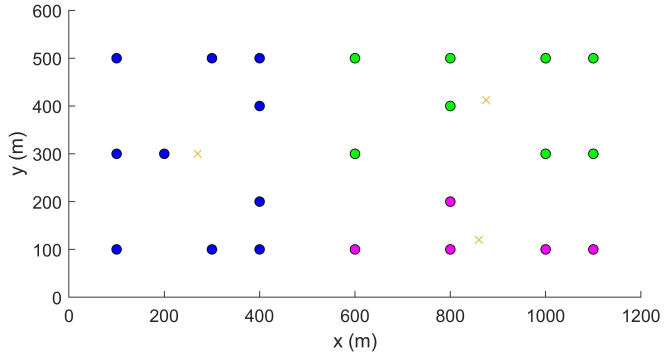


Figure 4.32: Distribution of tasks in clusters (after iteration 1) in simulation 3

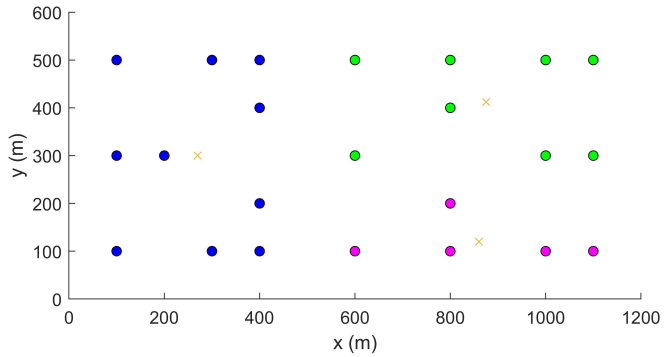


Figure 4.33: Distribution of tasks in clusters (after iteration 2) in simulation 3

And after applying the Held-Karp algorithm the following trajectories are obtained for each of the UAVs.

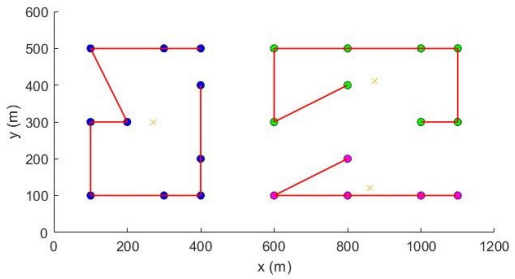
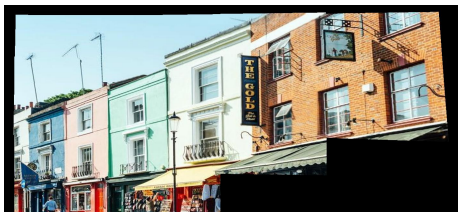


Figure 4.34: Final clusters and minimum path for each UAV in simulation 3

(a) Cluster 1 (dark blue)



(b) Cluster 2 (green)

(c) Cluster 3 (pink)

Figure 4.35: Obtained mosaic for each of the clusters in simulation 3

Then, the simulation where the UAVs ”fly” over the image starts, and the mosaics of Figure 4.35 will be formed with the images that are taken inside each cluster.

Finally, the mosaic of Figure 4.36 is obtained. As it can be seen, it has been possible to cover all the mission area with one less drone. However, it should be paid attention to the number of tasks that are assigned to each cluster with this new clustering distribution. In this case it has been no problem as the number of tasks per city was less than 14. Nevertheless, if the number of tasks inside a cluster exceeds this value, the Held-Karp algorithm will not be able to find a trajectory in a reasonable runtime.

Figure 4.36: Final mosaic in simulation 3

Chapter 5

Conclusions

In this chapter, the overall conclusions of the whole projected are commented. In addition, possible future work about this thesis and ways of improving the system performance are suggested.

In the last decades, the presence of UAVs has increased widely in the military word, as they are able of monitoring or attacking an area of conflict without endangering human lives. One UAV example is the Watchkeeper X unmanned aircraft system, which is capable of transmitting high quality images and video securely and reliably to different locations. However, the main disadvantage of this system is that it is quite big and expensive. That is why, the trend now is to use multiple smaller UAVs with single cameras incorporated on them working together as a swarm. This provides several advantages such as:

- Cost reduction
- Reduction in the time required to fulfill the mission, as the same number of tasks is performed using a higher number of UAVs.
- More flexible solution: The system can continue working even if a couple of drones get lost or are unable to contribute for some reason.

In order to obtain high quality images of a certain area by using multiple agents working together as a swarm, it will be necessary to coordinate their positions in an optimal way, that is the reason why developing suitable task allocation algorithm for the mission becomes so important.

The aim of this thesis has been to design a robust system that using a fixed number of drones with single cameras on them delivers a good resolution picture comparing with the ones that are obtained from expensive systems. The overall project has been developed in a simulation environment using Matlab.

The carried work can be summarized in the following steps. First, an image mosaicing algorithm that stitches multiple overlapping images has been designed. Secondly, given a certain mission scenario area, a task allocation algorithm that assigns each UAV to certain positions in the area has been developed. In addition, this algorithm defines the trajectory that each UAV should follow. After that, the different parts of the project have been validated independently. And finally, the integration of the image mosaicing and task allocation algorithms was proved to work properly by means of a simulation in Matlab.

The results obtained will be summarized below.

1. Image mosaicing algorithm

The results obtained in the validation from the image mosaicing algorithm proved that the developed algorithm worked with a wide variety of images, that is, with images that had lots of feature points and also with images that had more sparse features. When the mosaic consisted on images that should just be stitched all of them one after the other in the same direction, the final result was achieved perfectly. That is, it was enough with entering all the images at the same time. In addition, the algorithm performed well both for small and bigger overlapping areas between images.

2. Task allocation: Greedy algorithm

The proposed Greedy algorithm which selected the most suitable tasks in order to cover

all the mission area was proved to perform better than the method of selecting the same number of tasks randomly. That is, the Greedy algorithm selects the most suitable tasks in order to cover all the mission area by using the minimum possible number of UAVs. So, after performing a Monte Carlo simulation for one of the cases, it was demonstrated that with 200 iterations, there was no iteration in which the 100% coverage was achieved. The coverage area resulted to be between the 59% and the 83%, resulting the mean coverage are to be a 71% of the total one.

3. Integration of image mosaicing and mobile tasking

The integration of these two parts of the algorithm has been performed by means of a simulation in Matlab, in which the mission area was assumed to be an image of dimensions H$\times$W. So, it was made the UAVs fly over the image taking pictures in the positions that were selected by the task allocation algorithm.

In order to stitch correctly all the obtained images, different techniques were tried. However, the most adequate one resulted the one of forming the mosaic in two steps. Supposing there are N UAVs, the first step consisted on forming first N mosaics, one with the images of each cluster. After that, the N obtained mosaics were stitched together running one last time the image mosaicing algorithm.

The integration between both algorithms was proved to work perfectly when the system is formed by 4 UAVs, as the obtained final mosaics represented perfectly the images that were considered to be the mission area. However, in this part, as it was required to stitch together multiple images inside an area that were not located in the final mosaic just one after the other, it was required a bigger overlapping between images. This could be also influenced by the fact that all the input images were cut outs of the original image around a selected point, so if the chosen overlapping region between images was too small, they would not be enough feature points in the resulting cut out images, and therefore, the matching step would not be performed appropriately.

Finally, the system has also been proven to be flexible when suddenly a UAV is unable

to contribute. When this happens the system is capable of reprogramming the clustering algorithm so that it separates the selected tasks in N-1 clusters, being N the number of UAVs that were at the beginning. Therefore, it is guaranteed that all the tasks will be performed.

5.1 Future Work

To end, they will me mentions some possible steps to follow with this research research project:

- **Dynamics of the UAVs:** In this project it has been assumed that UAVs move from one position of the space to another one, but it has not been defined the agents' dynamics. That is the reason why, in order to do a more real simulation it would be necessary to define their movement dynamics and also their stopping dynamics for the instant in which the photograph is taken.

- **Increase the number of UAVs:** The proposed algorithms have been proven to work appropriately when the number of UAVs that are considered is small. However, it should be analyzed what happens when this number becomes high. In addition, having too many agents requires a perfect coordination among them, as it is easier for them to obstruct each others trajectories.

- **More realistic scenario:** In this simulation it has been assumed that all the mission area should be photographed. However, in reality there might be some zones which are more interesting than others. This could be simulated by defining some probability density function over the plane that reflects the probability of having some important information in some of the positions of the mission area.

- **Propose other methods to select the most appropriate trajectory inside a cluster:** The idea of finding the shortest path in order to fulfill all the tasks inside a cluster in the minimum possible time has been faced as a case of the Travelling Salesman

Problem, which has been proven to be NP-hard. The developed algorithm has been based on the Held-Karp algorithm that uses dynamic programming to solve this problem. However, because of the large memory required to solve the problem using this method, it is not suitable when the number of tasks becomes bigger than 20. That is the reason why it will be convenient to look in the future for other alternative methods to solve the TSP which are more suitable when the number of tasks increases.

- **Implementation in real scenarios**: As a consequence of the limited time that has been available to develop the project, it has not been possible to test the algorithm in a real environment. Therefore, a future step would be to implement the proposed algorithms in a real environment. However, it should be taken into account that in real scenarios there are many more factors that should be taken into account such as the environmental conditions, the flight height, the characteristics of the cameras...

Appendix A

Extra data from simulation 1

These are the cropped images that were obtained from the first simulation.

1

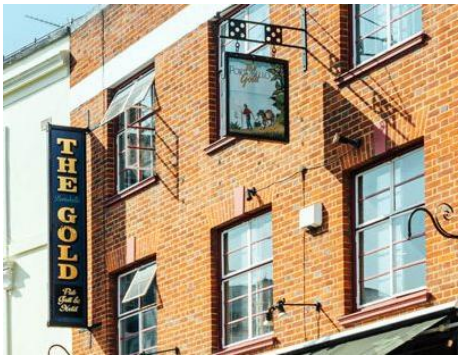
2

3

4

Figure A.1: Obtained cropped images (1-4) from simulation 1

5



6

7

8

9

10

Figure A.2: Obtained cropped images (5-10) from simulation 1

11

12

13

14

15

16

Figure A.3: Obtained cropped images (11-16) from simulation 1

17

18

19

20

21

22

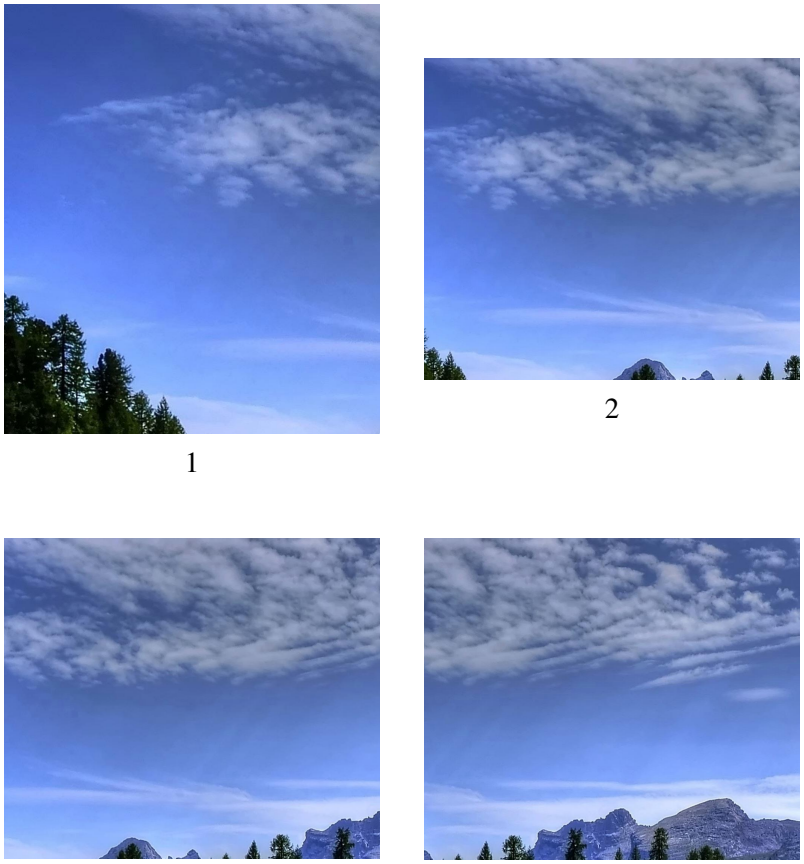
23

Figure A.4: Obtained cropped images (17-23) from simulation 1

Appendix B

Extra data from simulation 2

These are the cropped images that were obtained from the second simulation.



1 2 3 4

Figure B.1: Obtained cropped images (1-4) from simulation 2

5

6

7

8

9

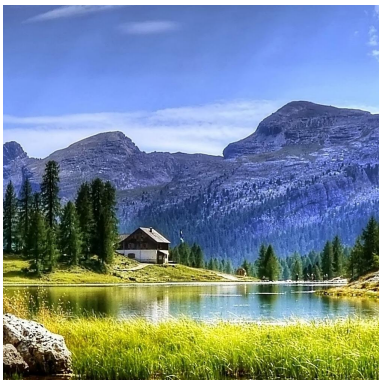
10

Figure B.2: Obtained cropped images (5-10) from simulation 2

11

12

13



14

15

16

Figure B.3: Obtained cropped images (11-16) from simulation 2

17

18

19

20

21

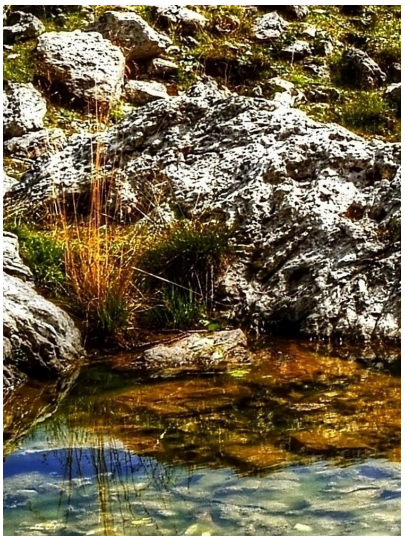
22

Figure B.4: Obtained cropped images (17-22) from simulation 2

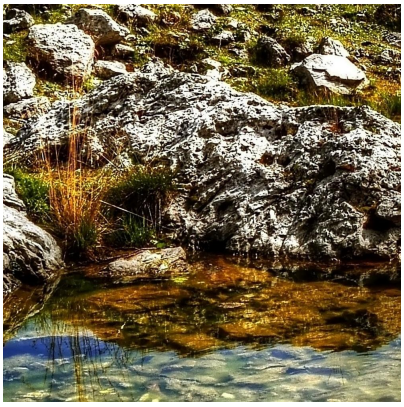
23

24

25



26



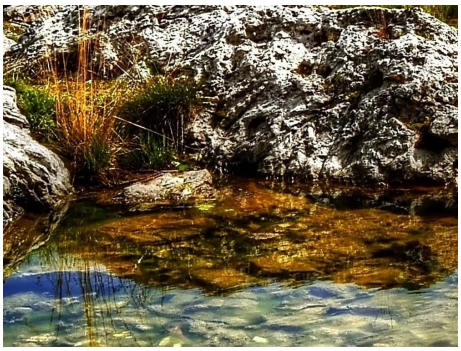
27

28

Figure B.5: Obtained cropped images (23-28) from simulation 2

29

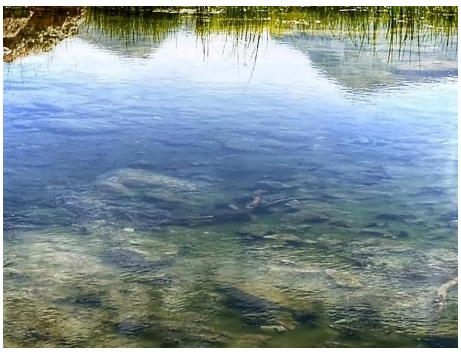
30



31

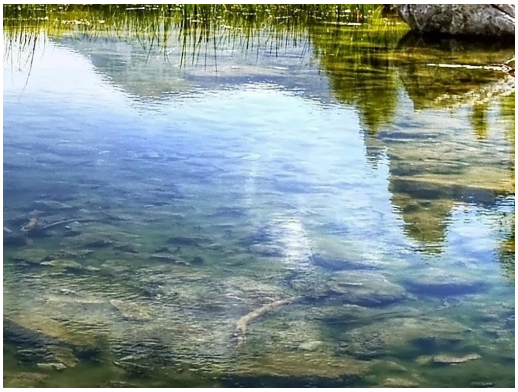
32

33

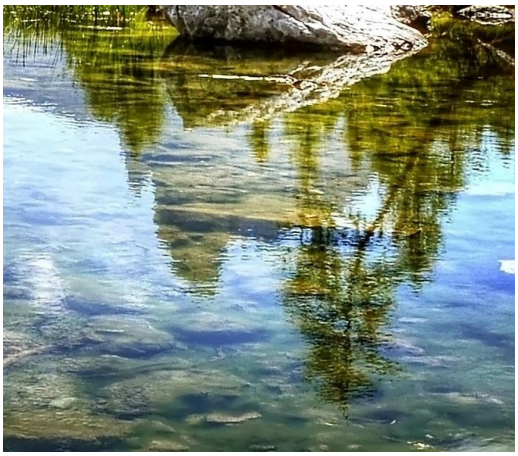


34

Figure B.6: Obtained cropped images (29-34) from simulation 2



35



36

37

Figure B.7: Obtained cropped images (35-37) from simulation 2

Bibliography

Berg, A. and Malik, J. (2001), Geometric blur for template matching, Vol. 1, IEEE Comput. Soc, pp. I–607–I–614.
URL: *http://ieeexplore.ieee.org/document/990529/*

Brown, M., Szeliski, R. and Winder, S. (2005), Multi-Image Matching Using Multi-Scale Oriented Patches, Vol. 1, IEEE, pp. 510–517.
URL: *http://ieeexplore.ieee.org/document/1467310/*

Ghannam, S. and Lynn, A. (2013), 'Cross Correlation versus Mutual Information for Image Mosaicing', *International Journal of Advanced Computer Science and Applications* **4**(11).
URL: *http://thesai.org/Publications/ViewPaper?Volume=4Issue=11Code=IJACSASerialNo=13*

Ghosh, D. and Kaabouch, N. (2016*a*), 'A survey on image mosaicing techniques', *Journal of Visual Communication and Image Representation* **34**, 1–11.
URL: *http://linkinghub.elsevier.com/retrieve/pii/S1047320315002059*

Ghosh, D. and Kaabouch, N. (2016*b*), 'A survey on image mosaicing techniques', *Journal of Visual Communication and Image Representation* **34**, 1–11.
URL: *http://linkinghub.elsevier.com/retrieve/pii/S1047320315002059*

Goyal, S. (n.d.), 'A Survey on Travelling Salesman Problem', p. 10.

Hahsler, M. and Hornik, K. (2007), '**TSP** - Infrastructure for the Traveling Salesperson

Problem', *Journal of Statistical Software* **23**(2).
URL: *http://www.jstatsoft.org/v23/i02/*

Harris, C. and Stephens, M. (1988), A Combined Corner and Edge Detector, Alvey Vision Club, pp. 23.1–23.6.
URL: *http://www.bmva.org/bmvc/1988/avc-88-023.html*

Hartley, R. and Zisserman, A. (n.d.), 'Multiple View Geometry in Computer Vision, Second Edition', p. 673.

Held, M. and Karp, R. M. (1961), 'A Dynamic Programming Approach to Sequencing Problems', *Proceedings of the 1961 16th ACM National Meeting* pp. 71.201–71.204.
URL: *http://doi.acm.org/10.1145/800029.808532*

Heng, L., Gotovos, A., Krause, A. and Pollefeys, M. (2015), Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments, IEEE, pp. 1071–1078.
URL: *http://ieeexplore.ieee.org/document/7139309/*

Islam, B. and Kabir, M. J. (n.d.), 'A New Feature-Based Image Registration Algorithm', p. 6.

Jichao Jiao, Baojun Zhao and Shasha Wu (2011), A speed-up and robust image registration algorithm based on FAST, IEEE, pp. 160–164.
URL: *http://ieeexplore.ieee.org/document/5952825/*

Krause, A. and Golovin, D. (2013), Submodular Function Maximization, *in* L. Bordeaux, Y. Hamadi, P. Kohli and R. Mateescu, eds, 'Tractability', Cambridge University Press, Cambridge, pp. 71–104.
URL: *https://www.cambridge.org/core/product/identifier/CBO9781139177801A031/type/book*$_{p}$*art*

Little, J. D. C., Murty, K. G., Sweeney, D. W. and Karel, C. (1963), 'An Algorithm for the

Traveling Salesman Problem', *Oper. Res.* **11**(6), 972–989.

URL: *http://dx.doi.org/10.1287/opre.11.6.972*

Nemhauser, G. L., Wolsey, L. A. and Fisher, M. L. (1978), 'An analysis of approximations for maximizing submodular set functions—I', *Mathematical Programming* **14**(1), 265–294.

URL: *http://link.springer.com/10.1007/BF01588971*

Okumura, K.-i., Raut, S., Qingyi Gu, Aoyama, T., Takaki, T. and Ishii, I. (2013), Real-time feature-based video mosaicing at 500 fps, IEEE, pp. 2665–2670.

URL: *http://ieeexplore.ieee.org/document/6696732/*

Shin, H.-S. and Segui-Gasco, P. (2014), UAV Swarms: Decision-Making Paradigms, *in* R. Blockley and W. Shyy, eds, 'Encyclopedia of Aerospace Engineering', John Wiley & Sons, Ltd, Chichester, UK, pp. 1–13.

URL: *http://doi.wiley.com/10.1002/9780470686652.eae273*

Singh, A., Krause, A., Guestrin, C. and Kaiser, W. J. (n.d.), 'Efficient Informative Sensing Using Multiple Robots', p. 50.

The Effect of Using Inter-Frame Coding with Jpeg to Improve the Compression of Satellite Images (2017), *IRAQI JOURNAL OF SCIENCE* **58**(4A).

URL: *http://www.ijs.scbaghdad.edu.iq/issues/Vol58/No4A/Vol58Y2017No4APP1970-1978.pdf*

Vaghela, D. and Naina, K. (n.d.), 'A Review of Image Mosaicing Techniques', p. 6.