1 2 9 0

UNIVERSIDADE D
COIMBRA

Diogo Filipe Rodrigues Gonçalves

IMPACT OF IMAGE ACQUISITION GEOMETRY AND SfM-MVS PROCESSING PARAMETERS ON THE 3D RECONSTRUCTION OF COASTAL CLIFFS

Dissertação no âmbito do Mestrado em Engenharia de Informação Geoespacial orientada pelo Professor Doutor Gil Rito Gonçalves e apresentada ao Departamento de Matemática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2020

Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Impact of image acquisition geometry and SfM-MVS processing parameters on the 3D reconstruction of coastal cliffs

(Utilização de drones na reconstrução 3D de arribas costeiras: impacto da geometria de aquisição de imagem e dos parâmetros do processamento fotogramétrico)

Diogo Filipe Rodrigues Gonçalves

Dissertação de Mestrado na área científica da Engenharia de Informação Geoespacial orientada pelo Professor Doutor Gil Rito Gonçalves e apresentada ao Departamento de Matemática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2020

1 2 9 0

UNIVERSIDADE D COIMBRA

Agradecimentos

Ao meu orientador Professor Doutor Gil Rito Gonçalves pela ajuda, conselhos e esclarecimentos ao longo de toda a dissertação.

Ao Instituto de Engenharia de Sistemas e Computadores de Coimbra, por me ter proporcionado a experiência de uma bolsa de investigação.

À minha noiva, Andreia, pelo carinho, paciência e suporte. Por me ter feito crescer como homem e ver a vida com outros olhos.

Aos meus "cunhados", Raquel e João, pela amizade, ajuda e sugestões nalguns pontos da dissertação.

Aos meus pais, Bélina e Emídio, por me terem transmitido os valores corretos, nunca terem desistido de mim e acreditarem nas minhas capacidades.

Abstract

Due to the danger of rockfall inherent to erosion, coastal cliffs arouse high interest in its monitoring. It is important to perform a 3D reconstruction in order to identify, measure and prevent collapse. With the high technological advances in the scope of photogrammetry and 3D modelling, new areas of research have been developed, mainly in the utilization of drones for the acquisition and processing of data. Due to its complexity, prior planning is necessary in order to minimize occlusions in the 3D model (points not visible in at least two images).

This dissertation aims to study exhaustively a commercial photogrammetric processing software (Agisoft Metashape) in order to optimize a 3D reconstruction of a coastal cliff. In this context, two unmanned aerial systems will be used for the image acquisition, namely a fixed-wing (Ebee Sensefly) and a multirotor (DJI Phantom 4 Pro). To orient the images blocks, the main parameters under test will be the limit of the tie points and key points, the weights of the control points and tie points in the bundle block adjustment (BBA), and the best spatial location for the control points. Then, with the reprojection error of the tie points, points that contribute to higher errors can be filtered and removed. Thus, the densification of tie points can result in a more precise dense cloud. Finally, the automatic identification of areas without data resulting from occlusions or insufficient overlapping of images will be made using a voxelization implementation in MATLAB.

The results reflect an appropriate use of the limits of the key points and tie points and it is not advantageous not to place a limit on the latter (where all tie points are chosen). The overwhelming majority of tie points (sparse cloud) are only visible in 3 or less images and have a reprojection error concentrated around 0.1 pixels. In the two types of acquisition geometry, the point cloud density of the Ebee Sensefly presents values around 200 points per m^3 whereas in Phantom 4 Pro the values are aroumd 1300 points per m^3. In terms of zones without data, for the fixed wing, with spatial resolutions of 1 m, 0.5 m and 0.25 m, volumes of 50 m^3, 56.75 m^3 and 60.74 m^3 were identified, respectively. For the multirotor using the same combinations, 0 m^3, 1.75 m^3 and 1.94 m^3 are identified as zones without data.

For Agisoft Metashape, the processing parameters do not influence the accuracy of the 3D model it being adequate to use the default parameters. For both acquisition geometries, the point cloud from multirotor has a points density much higher than that of the fixed wing (difference around 1100 points per m3). Regarding voxelization, the results are promising because most voxels without data are identified. This method is sensitive to the spatial resolution of voxels, since we have point clouds that already include zones without data (gap zones). Therefore, the detection of this gap zones depends on the spatial resolution.

With these indicators, it was concluded that the acquisition of Phantom 4 Pro is better for the 3D reconstruction of a vertical cliff in terms of the accuracy of 3D model, points density and zones without data.

Keywords: coastal cliffs, key points and tie points, 3D reconstruction, voxelization, gap zones

Resumo

Devido ao perigo de quedas de rochas inerentes à erosão, as arribas costeiras despertam elevado interesse de monitorização. Torna-se importante efetuar uma reconstrução 3D de forma a identificar, medir e prevenir possíveis derrocadas. Com os elevados avanços tecnológicos no âmbito da fotogrametria e modelagem 3D, desenvolveram-se novas áreas de investigação, principalmente na utilização de drones para a aquisição e processamento de dados. Devido à sua complexidade, é necessário efetuar um planeamento prévio de forma a minimizar oclusões no modelo 3D (pontos não visíveis em pelo menos duas imagens).

A presente dissertação tem como objetivo estudar exaustivamente um software comercial de processamento fotogramétrico (Agisoft Metashape) de forma a otimizar a reconstrução 3D de uma arriba costeira. Neste contexto, serão utilizados dois sistemas aéreos não tripulados para a aquisição de imagens, nomeadamente um asa fixa (Ebee Sensefly) e um multi rotor (DJI Phantom 4 Pro). Para efetuar a orientação do bloco de imagens, os principais parâmetros em teste serão o limite dos *key points* e *tie points*, os pesos dos pontos de controlo e *tie points* no ajustamento por feixe de perspetiva, a melhor localização espacial para os pontos de controlo. De seguida, com o erro de reprojeção dos *tie points*, podemos filtrar e remover os pontos que contribuem com erros mais elevados. Assim, a densificação dos *tie points* pode resultar numa nuvem densa mais precisa. Por fim, será feita a identificação automática de zonas sem dados resultantes de oclusões ou insuficiente sobreposição de imagens recorrendo a uma implementação de voxelização em MATLAB.

Os resultados obtidos refletem uma utilização adequada dos limites dos *key points* e *tie points* não sendo vantajoso a não colocação de limite neste último (onde são escolhidos todos os *tie points*). A esmagadora maioria dos *tie points* (nuvem esparsa) são apenas visíveis em 3 ou menos imagens e apresentam um erro de reprojeção concentrado em torno de 0.1 pixels. Para os dois tipos de geometrias de aquisição, a densidade de pontos do Ebee Sensefly apresenta valores em torno de 200 pontos por m^3 enquanto que no Phantom 4 Pro os valores estão situados em torno de 1300 pontos por m^3. Em termos de zonas sem dados, para o asa fixa e com as resoluções espaciais de 1 m, 0.5 m e 0.25 m, foram identificados volumes de 50 m^3, 56.75 m^3 e 60.74 m^3, respetivamente. Para o multirotor utilizando as mesmas combinações, foram identificados 0 m^3, 1.75m^3 e 1.94 m^3 como sendo zonas sem dados.

Para o Agisoft Metashape, os parâmetros de processamento não influenciam a precisão do modelo 3D sendo adequado a utilização dos parâmetros por defeito. Para as duas geometrias de aquisição, a nuvem de pontos resultante do multirotor tem uma densidade de pontos muito superior à do asa fixa (diferença de cerca de 1100 pontos por m^3). Em relação à voxelização, os resultados são promissores pois são identificados grande parte dos *voxels* sem dados. Este método é sensível à resolução espacial do *voxels* dado que temos presente uma nuvem de pontos que por si só já contempla zonas sem dados. Por isso a deteção destas áreas depende da resolução espacial.

Com estes indicadores, concluiu-se que a geometria de aquisição do Phantom 4 Pro é melhor para a reconstrução 3D de uma arriba em termos de precisão do modelo 3D, densidade de pontos e zonas sem dados.

Palavras-chave: arribas costeiras, *key points* e *tie points*, reconstrução 3D, voxelização, zonas sem dados

Abbreviations

API	Application Programming Interface
BBA	Bundle Block Adjustment
CC	CloudCompare
CHP	Check Point
DOP	Dilution of Precision
GCP	Ground Control Point
GNSS	Global Navigation Satellite System
GUI	Graphical User Interface
H	3D Histogram
HSDV	Horizontal Standard Deviation
ICO	Image Count Observation
IDE	Integrated Development Environment
MPA	Metashape Python API
MVS	Multi View Stereo
RE	Root Mean Square Reprojection Error
RMSE	Root Mean Square Error
SfM	Structure from Motion
TLS	Terrestrial Laser Scanning
UAS	Unmanned Aerial System
VSDV	Vertical Standard Deviation
XML	Extensible Markup Language

Contents

1 Introduction

Coastal cliffs are specific coastal landforms characterized by steep rock walls that provide wide range of habitats for plants and animals (Westoby et al., 2018; Young & Carilli, 2019). The actual sea level rise induced by climatic change is intensifying erosional processes of coastal cliffs (Westoby et al., 2018; Young & Carilli, 2019), therefore it is essential to monitor the morphological changes of these landforms. The erosional processes of cliffs are mainly caused by the notching at their base by marine processes, and/or by the collapse of the cliff face induced by a combination of atmospheric and marine processes (Westoby et al., 2018; Young & Carilli, 2019).

In the context of coastal cliff monitoring, unmanned aerial system (UAS) has been proved to be a valuable tool for reconstruction of cliff geometry (Esposito et al., 2017; Gómez-Gutiérrez & Gonçalves, 2020; Jaud et al., 2019; Ružić et al., 2014; Warrick et al., 2017). Photogrammetric techniques applied on the high-resolution images collected by UAS allowed, for instance, to identify the erosional processes of cliffs at the Adriatic sea (Ružić et al., 2014) and to measure cliff recession rate on the Tyrrhenian Sea and California coasts (Esposito et al., 2017; Warrick et al., 2017).

The 3D reconstruction of a coastal cliff from a UAS survey consists in generating a dense cloud of points from the collected images. However, this is a challenging task since it requires an appropriate drone flight planning and camera setting. A good choice of parameters like flight height and speed, along with camera tilt angle and image overlap, influence the capacity of reconstructing the cliff face surface and can avoid blind spots (Gómez-Gutiérrez & Gonçalves, 2020; Jaud et al., 2019). Nevertheless, the choice of all involved technical parameters is not yet standardized, as environmental conditions (e.g., cliff geometry, ecc.) must be considered (Tmuši et al., 2020). Structure-from-motion (SfM; Ai et al., 2015; Smith et al., 2015) and multi-view stereo (MVS; Ahmadabadian et al., 2013; Remondino et al., 2013) algorithms are usually applied to the acquired aerial images using specific softwares. Agisoft Metashape is one of the most used software for this purpose.

A detailed insight about how each parameter influences the 3D reconstruction accuracy by Metashape is still lacking (Mayer et al., 2018). For instance, understanding how the choice of the key points (points of interest) when matching images through tie points affects the reprojection error is a valuable knowledge. Jaud et al., (2019) reported that image overlap, spatial resolution and illumination conditions are the main parameters to consider for tie point detection, however, further investigation is needed in this regard.

Also due to the absence of purely three-dimensional methodologies for the analysis of point clouds, dimensionality reduction (for 2D) where the point cloud projection on a plane is calculated, is frequently used as an approach. In this topic of coastal cliffs, this methodology can be used when there is high verticality (Tonini & Abellan, 2014). However, in some cases, the terrain profile does not allow it and is necessary to split the dense cloud (Gómez-Gutiérrez & Gonçalves, 2020).

The main objective of this work is to evaluate the influence of the processing parameters in the 3D reconstruction of a vertical cliff. Images acquired by two distinct UASs, namely a multi-rotor (DJI Phantom 4 Pro) and a fixed-wing (Ebee Sensefly), were used for this aim. The initial step involves assessing the quality of Phantom 4 Pro image overlap reduction through the use of a specific Agisoft Metashape function (*Reduce overlap*). Back to the SfM process, the key points limit, the tie points limit, the camera calibration parameters, the control points placement and the interior and exterior orientation parameters optimization are assessed separately in order to highlight the best settings of Agisoft Metashape. The strategy includes the development of tools exploring the functionality of its Python application programming interface (API). After that, with a dense cloud optimized for each dataset, a method based on voxelization was developed on MATLAB to locate and quantify volumetrically zones without data due to occlusions and/or lack of overlap between images, thus surpassing the drawbacks of nowadays methodologies.

This work is divided into six major chapters where we can highlight: (1) an introduction to contextualize the problems of the reconstruction of a coastal cliff; (2) the presentation of the study area; (3) methods inherent to the 3D reconstruction of a coastal cliff, with all the hypotheses under study; (4) exposure of all results calculated by the applied methods and discuss about the variables under test and critical analysis of the results; (6) main conclusions and future studies.

2 Study area and UAS image acquisition

2.1 Study Area

The study area is Praia do Porto da Calada, a beach located in the central Portuguese coast facing the Atlantic Ocean (Figure 1a). This beach is an embayment bounded by two cliffs with an approximate altitude of 100 m. The cliff to be studied has an orientation SW-NE. A concrete sea wall was built at the base of the cliff (Figure 1b).to increase its stability, as several landslides occurred over the last decades. As the characteristics and history of this cliff represent dangerous situations where continuous monitoring is needed, it offers great conditions to develop this study.

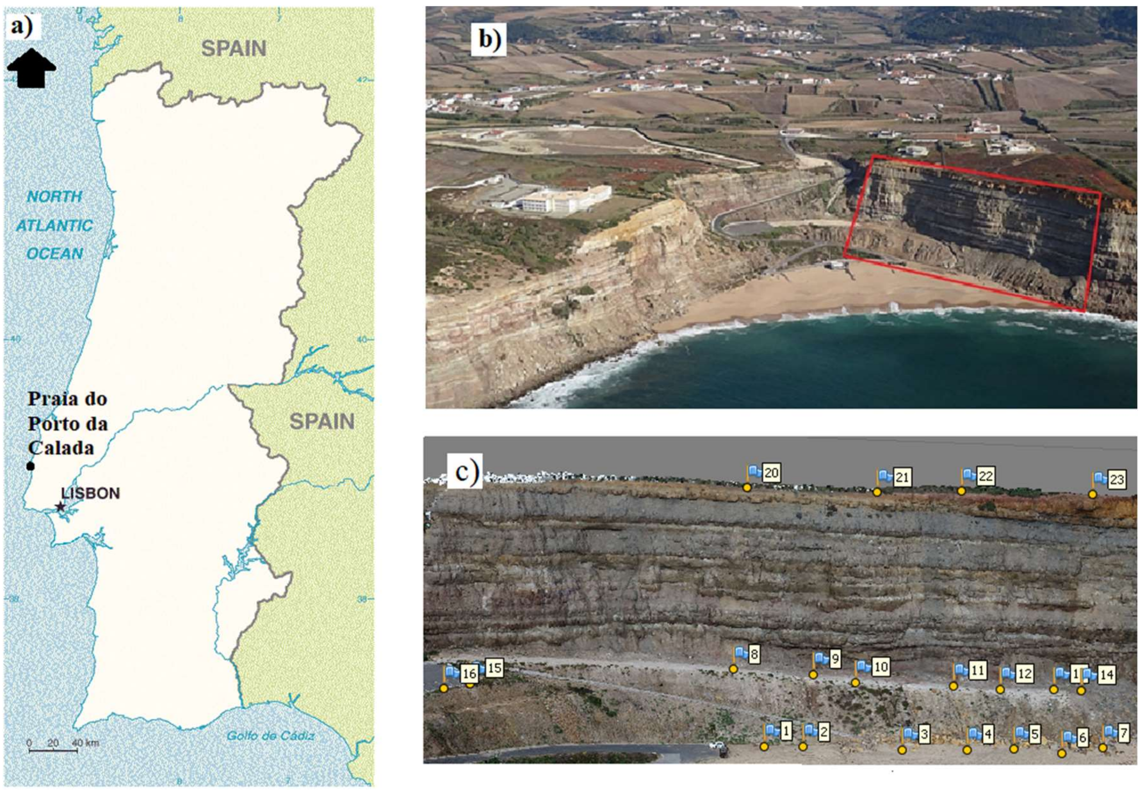


Figure 1. Study site. a) Praia do Porto da Calada location on the Portuguese map; b) aerial physical picture of the cliff in exam (red rectangle); c) placement of the Ground Control Points (GCPs, numbered blue flags) on the coastal cliff

2.2 UAS flights planning and image acquisition

Two flights were performed at the study site, using two different unmanned aerial vehicles, namely a multi-rotor (DJI Phantom 4 Pro) and a fixed-wing (Ebee Sensefly). The ground control points(GCPs) were distributed across all areas being placed 7 GCPs on the bottom of the cliff, 9 in the intermediate step and 4 on the top (Figure 1c).

The fixed-wing (Ebee Sensefly) was equipped with a Sony WX220 camera, 4.45 mm focal length with the images containing 18 MP. The Ebee Sensefly does not have a gimbal to rotate the camera axis (pitch and yaw axis), therefore the drone flew with an off-nadir angle. Overall, the Ebee Sensefly collected 53 images (see Figure 2a)

The multirotor DJI Phantom 4 Pro was equipped with a Sony FC6310 camera, 8.80 mm focal length with the images containing 20 MP. The flight plan was set on the Drone Harmony software. The flight path was parallel to the cliff, with the camera off-nadir angle (pitch) equal to 90º. The drone went up and down on the same line varying, about 10º, the vertical axis (yaw). The DJI Phantom 4 Pro collected 448 images (see Figure 2b).

2.3 Reducing the number of images

Given the high number of images in the Phantom data, some of the images may be redundant. The "Reduce Overlap" tool identifies the valuable images (Agisoft, 2020). The image reduction aimed to optimize the time spent in dense cloud construction and also close the number of images between Ebee Sensefly and Phantom 4 Pro dataset. The time spent in the sparse cloud densification was objectively reduced by 35 % (approximately 4 hours). In summary, from 448 images, the tool selected only 258 images (Figure 2c) that will be tested in order to assess the loss or not of information (see section 3.6.1).

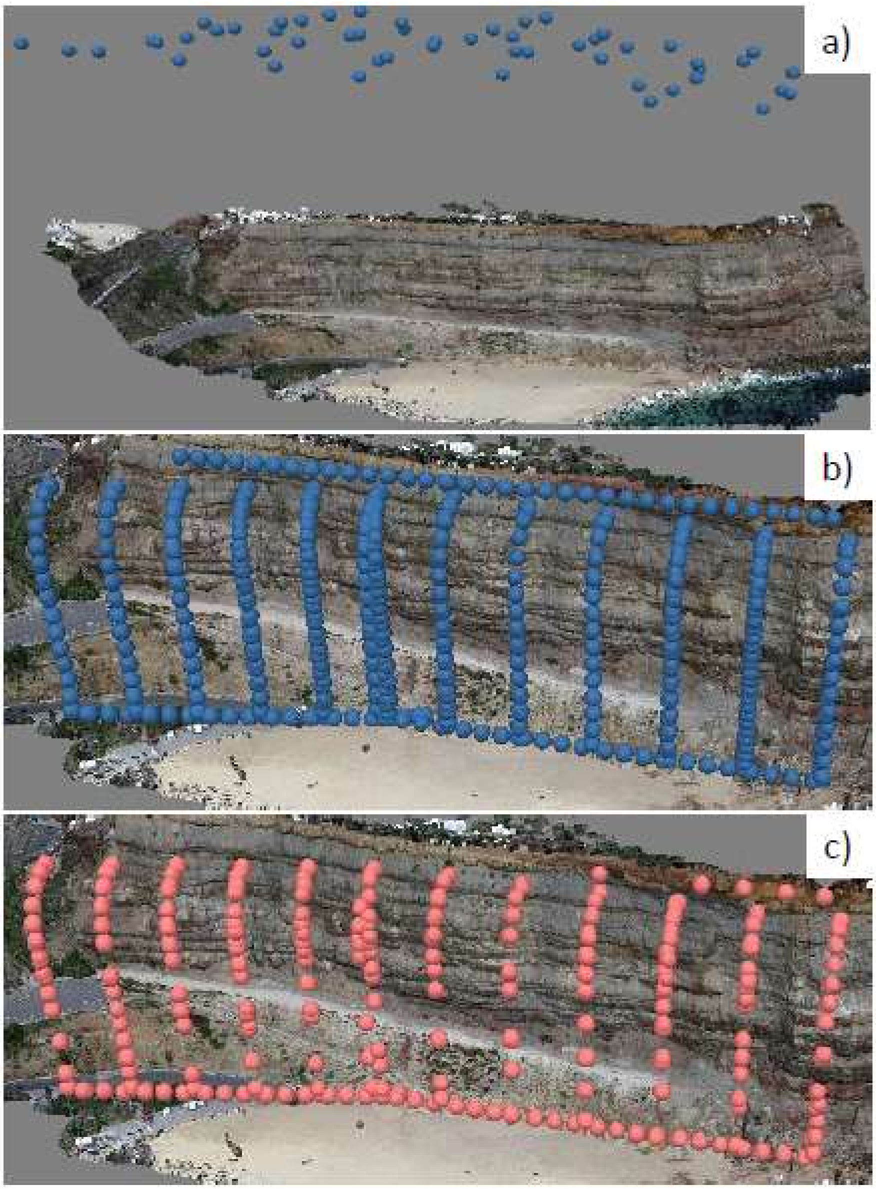


Figure 2. Flight plans and image acquisition. a) Camera position during the flight of Ebee Sensefly; b) Camera position during DJI Phantom 4 Pro flight; c) Reduced camera position of the DJI Phantom 4 Pro flight

3 Methods

The tasks to be carried out includes the study of several topics inherent to the photogrammetry assisted by drones. In this context, the workflow of a generic SfM-MVS will be explained (section 3.1.1) and applied to the Metashape workflow (section 3.1.2). Then, the main processing parameters that influence the orientation of a block of images resulting from the BBA will be presented. Subsequently, it will be discussed how the acquisition geometry of the image dataset influences the reconstruction of vertical cliffs, namely, the error of the model adjustment related to the control points, the global density of the point cloud and the possible appearance of zones without points (gap areas). The use of morphological analyses arises for the identification of gap zones. Finally, 7 sequential phases will be presented for an in-depth study of Metashape functionalities in the optimization of a dense cloud construction (Figure 10). Measurements in dense clouds such as density and quantification of gap areas are also inferred.

3.1 3D reconstruction of coastal cliffs based on UAS photogrammetry

3D reconstruction represents the generation of three-dimensional digital information (points or geometric shapes) of an object or area where all its forms are preserved. This method allows the use of spatial analysis, such as, for example, the stability evaluation of structures with high complexity and generation of Digital Elevation or Surface Models (DEM or DSM).

In the cliffs context, there is a huge lack of information due to inexistent monitoring, therefore compromising the prediction of possible hazards. It is known that the cliffs have very complex structures and the use of TLS techniques can be dangerous when collecting data. The use of UAS becomes important because, in most cases, ensures the protection of operators on site. These aircrafts collect images easily and provide information for the 3D reconstruction of a cliff. From two-dimensional images, the 3D points cloud is calculated using techniques of: (i) SfM to orient the images and generate a sparse cloud with points common to images; (ii) MVS to densify the sparse cloud using techniques of computer vision.

All the methodology inherent to a typical SfM-MVS workflow will be presented and explained (section 3.1.1) and then, compared step by step with the Metashape photogrammetric workflow (section 3.1.2).

3.1.1 General SfM-MVS workflow

The 3D reconstruction of a coastal cliff from a UAS survey consists in building a dense point cloud from the collected images. The procedure requires the application of SFM-MVS technique (Figure 3).

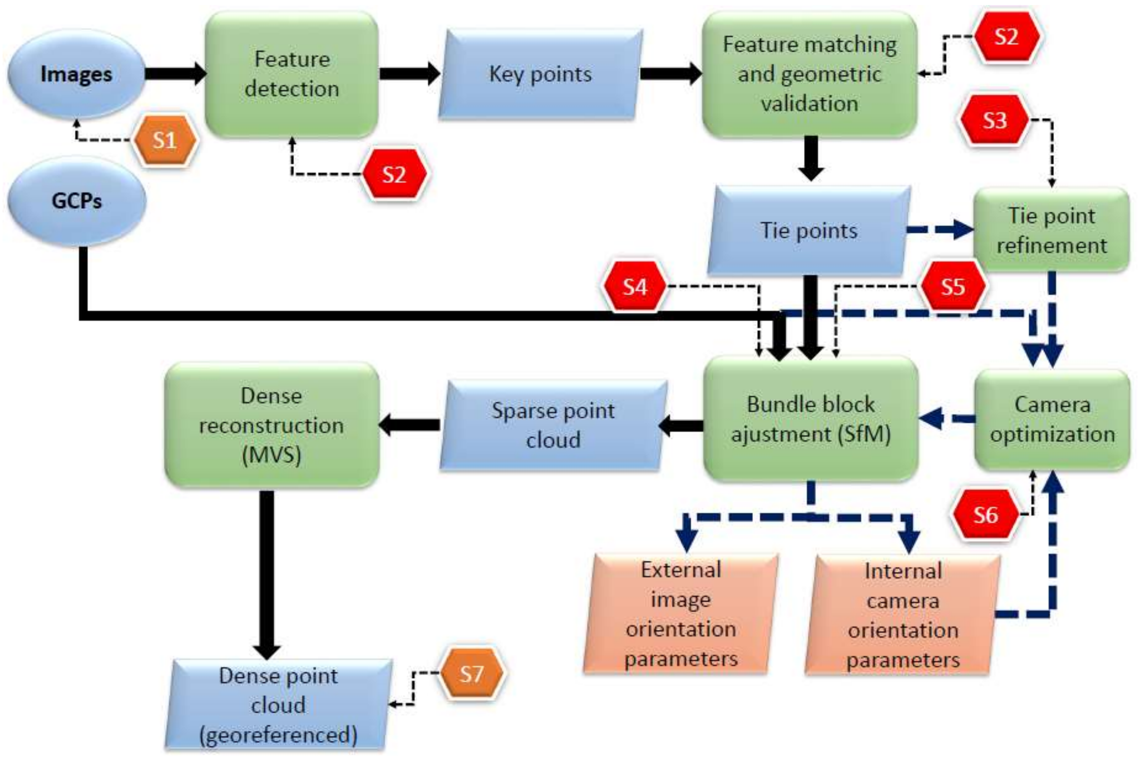

Images
Feature detection
Key points
Feature matching and geometric validation
S2
S1
S2
S3
GCPs
Tie points
Tie point refinement
S4
S5
Dense reconstruction (MVS)
Sparse point cloud
Bundle block ajustment (SfM)
Camera optimization
S6
External image orientation parameters
Internal camera orientation parameters
Dense point cloud (georeferenced)
S7

Figure 3. Workflow of a generic SfM-MVS process associated with the seven Metashape phases that are followed for an in-depth study of the generated dense point cloud (orange and red hexagons; section 3.6)

From the images collected by the drone, the first stage consists in "Feature detection" and "Feature matching and geometric validation" (Figure 3). The images are uploaded in the software that starts with the detection of the key point. Key points are scale-invariant features filtered by the location accuracy and sensitivity to noise (Lowe, 2004). As that key points are well identified in all images, they are matched by the respective identifier. These points are named tie points.

The second stage is the "Bundle Block Adjustment (SfM)" (BBA; Figure 3) that computes the interior camera parameters and the exterior orientation parameters of each image in the image block. The input parameters are the tie points coordinates selected in the first stage, the interior orientation camera parameters and the image set. Thus, the linearization of the collinearity equation in the least square method is applied to refine the orientation parameters from the image block because the linearization lacks initial approximations.

The third stage is the “Camera optimization” (Figure 3) that aims to improve the orientation camera parameters previously found by the BBA procedure. In this procedure, the reprojection error (see section 3.2.4.2) quantifies the difference between points on the sparse cloud and the same points reprojected using the camera parameters. The reprojection error is associated to each point, allowing removal of the points with the highest reprojection error. However, this isolated action does not change the error because the parameters calculated in the second stage are adjusted for the unfiltered sparse cloud. So, the process must be optimized for the filtered sparse cloud points.

The fourth stage is the “geo-referencing process”. This process is a 3D transformation between the model system and the object system that returns seven parameters (three for rotation, three for translations, and one scale factor).

The fifth and last stage is the “Dense reconstruction” (Figure 3). The initial task is to apply one of the Multi-View Stereo (MVS) algorithms to calculate depth maps from collinearity or projective equations (Remondino et al., 2013).

3.1.2 Metashape workflow

Metashape adapts well to a general SfM-MVS workflow. The corresponding stages are straightforward and natural.

First, the weights are attributed to the control points (markers) and tie points accuracy (see section 3.2.2). The control points accuracy is represented in the object and images referential that have as default 0.005 meters and 0.5 pixels, respectively. For instance, in the object referential a control point has a positional uncertainty of 0.005 meters, which is a bit overrated. In image referential, 0.5 pixels represents the confidence that the operator identifies a control point in an image. The tie points accuracy represents the residual of the tie points location in the image. By default, tie points accuracy has 1 pixel of tolerance. With these weights adjusted, four main phases are considered to build a dense cloud: 1. Align Photos; 2. Geo-referencing; 3. Refinement procedure; 4. Build Dense Cloud (Figure 4).

After importing the images, the first stage of Metashape workflow is “Align Photos”, which calculates the interior and exterior orientation parameters (Mayer et al., 2018). This stage performs the key points detection and matching (Smith et al., 2015) that corresponds to “Key Points Detection” and “Key Points Matching” (Figure 4). “Align

Photos" also computes the orientation parameters through BBA. The main user-defined parameters are:

- Accuracy: images are downscaled by a given factor. There are 5 possible values (highest, high, medium, lower, lowest) that reflect scaling down twice the values 0, 1, 2, 4, 8. Note that the highest precision uses the original size of the images. The less the subscale factor is, the greater is the computational time and the precision of the camera parameters;
- Key Points Limit: maximum number of key points calculated in each image. Perhaps, the number of key points has an upper limit related to image dimensions. The default value is 40000 key points.
- Tie Points Limit: maximum number of tie points selected among the key points matched in the images. The default value is 4000 tie points. If the user chose the zero value than Metashape will obtain all tie point that it can find.
- Adaptive Fitting: enable to estimate radial and tangential distortion parameters. If this option is disabled, Metashape only estimates focal length, principal point, radial distortion parameters, and tangential distortion parameters (Agisoft, 2020; see section 3.2.3).

Let's see how these parameters are used in SfM processing. The parameters of the BBA are implicit here because the limit of key and tie points and precision do not enter the BBA. That is, the photo coordinates of the tie points that are input parameters in the BBA together with the internal orientation parameters of the camera. In the Python API this concept is present because "Align Photos" is divided into two functions: 1. *matchphotos*: calculation of the photo coordinates of the tie points; 2. *Aligncameras (BBA)*: calculation of the orientation parameters of the image block plus the object coordinates of the tie points (sparse cloud). This sparse cloud lacks georeferencing because in this transformation between references systems (object and image) it is used the coordinates of the camera centers. This fact is present when the coordinate systems of the project, images, and GCPs are defined (see Figure 5b). In other words, the transformation is done on the camera reference system, which is transformed to system coordinates.

The "Geo-referencing" step represents a 3D transformation from an arbitrary system (or model system), and an object system. Usually, the object system reflects a geographic datum (e.g., the project system PT-TM06/ETRS89 in Portugal). Successively, GCPs are manually identified to compute the transformation parameters (Westoby et al., 2012).

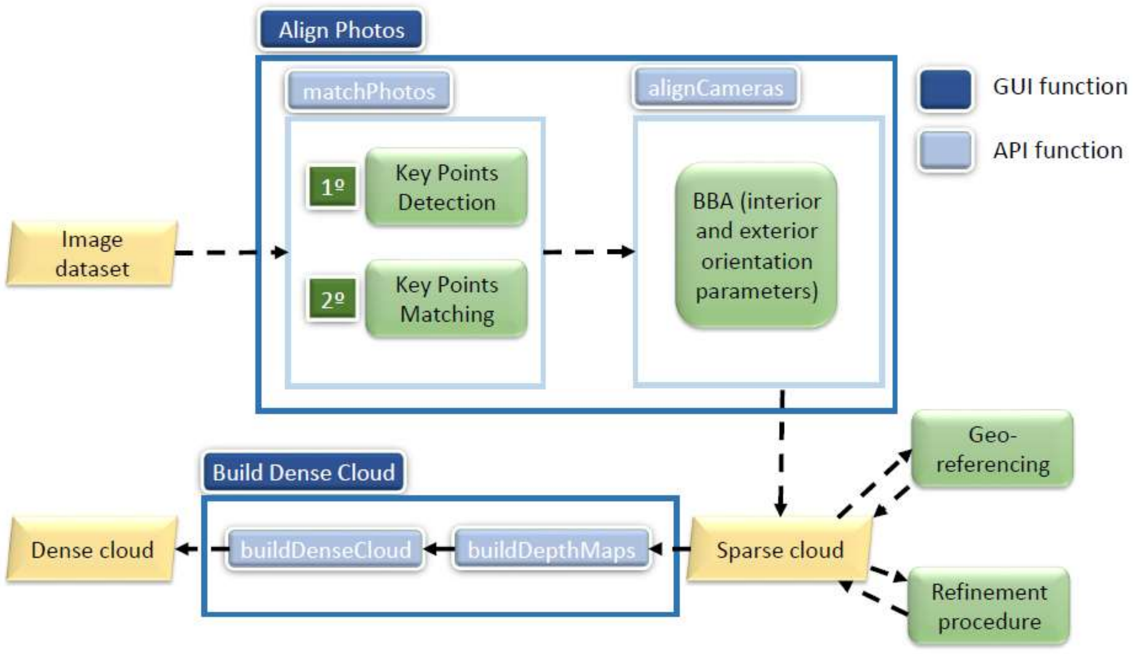


Figure 4. Flowchart for a typical Metashape workflow in Graphical User Interface (GUI) and in Python API

3.2 Impact of the SfM processing parameters on the BBA

The choice of input parameters for a BBA is not straightforward and can lead to results that are not representative of reality. For example, Mayer et al., (2018) reported that the weight of 5 mm for GCPs on the ground may be too optimistic. In this sense, the parameters under test and the metrics to evaluate the combination of several parameters will be exposed.

3.2.1 Number of key points and tie points

The selection of the key points number is related to the image resolution given that key points are calculated in the image space. Initially, a feature description is applied to calculate these points with a high correlation. Key points are used to connect each image through tie points. Therefore, a tie point is an intersection of several key points that represent the same feature. Increasing the number of key points does not mean that the tie points will also increase.

3.2.2 Weights used for tie points and GCPs

The GCPs accuracy on the object coordinate system is given by the position accuracy of the Real-Time Kinematic GNSS. The accuracy of the positioning represents the influence of the satellite constellation in the GCPs accuracy (Acharya, 2014) and is named as the dilution of precision (DOP). The individual standard deviation errors derived by the pseudo-distance correction (distance between the satellites and receiver) are given by (Acharya, 2014)

$$\sigma_x = \sqrt{H_{11}}\sigma_r$$

$$\sigma_y = \sqrt{H_{22}}\sigma_r$$

$$\sigma_z = \sqrt{H_{33}}\sigma_r$$

$$\sigma_t = \sqrt{H_{44}}\sigma_r$$

where $H_{4\times4}$ is the covariance matrix between a point position and the satellites positions, $H_{i\times i}$ is the variance of element i in $\{\sigma_x^2,\ \sigma_y^2,\ \sigma_z^2,\ \sigma_t^2\}$ and the σ_r is the receiver error. The geometric error given by

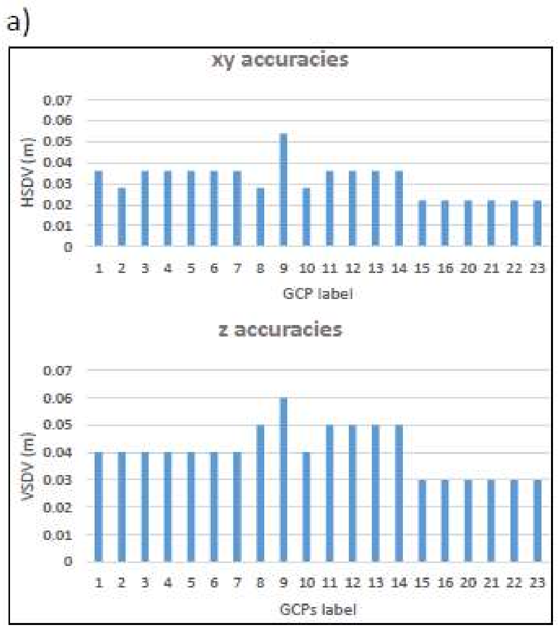
$$\sigma_G = \sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2 + \sigma_t^2} = GDOP \times \sigma_r \tag{1}$$

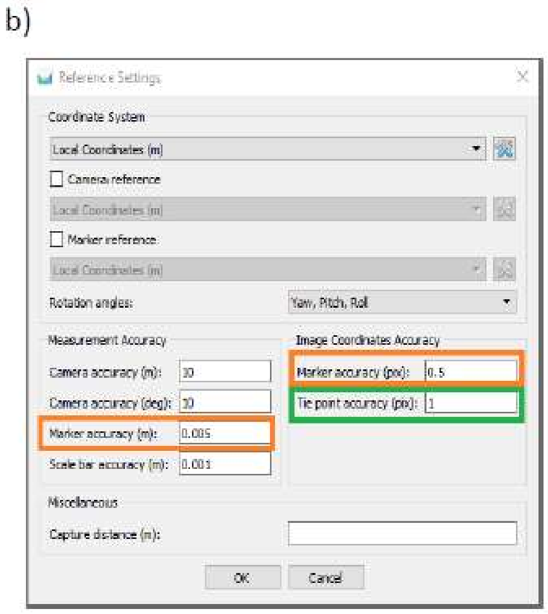
where $GDOP = \sqrt{H_{11} + H_{22} + H_{33} + H_{44}}$ is the Geometric DOP (GDOP). From equation (1), it can be particularized for horizontal and vertical standard deviation (HSDV and VSDV, respectively)

$$HSDV = \sqrt{\sigma_x^2 + \sigma_y^2}$$

$$VSDV = \sigma_z$$

Weights represent the confidence established for the observations (Casella et al., 2020). In “Reference Settings” of Metashape GUI, all these weights may be changed (Figure 5b). For the xy and z positions, the weights for GCPs in the terrestrial system were assigned using the HSDV and the VSDV (Figure 5a). Image accuracy represents the tolerance, in pixels, that GCPs (precision of the observation) and tie points (precision of the key points matches) can have.





Coordinate System

Local Coordinates (m)

☐ Camera reference

Local Coordinates (m)

☐ Marker reference

Local Coordinates (m)

Rotation angles: Yaw, Pitch, Roll

Measurement Accuracy

Camera accuracy (m): 10

Camera accuracy (deg): 10

Marker accuracy (m): 0.005

Scale bar accuracy (m): 0.001

Image Coordinates Accuracy

Marker accuracy (pix): 0.5

Tie point accuracy (pix): 1

Miscellaneous

Capture distance (m):

Figure 5. Weights associated to the GCPs and tie points. a) HSDV and VSDV obtained by the RTK GNSS survey; b) "Reference Settings" of Metashape GUI where the weights can be changed. Note that the "Marker accuracy (m)" tab will be changed for the weights of a)

3.2.3 Camera optimization

In the BBA, there are two different methodologies to calculate camera calibration parameters: pre and self-calibration (Griffiths & Burningham, 2019). For a typical photogrammetric camera, a periodic calibration (i.e. pre-calibration) is frequently made and the corresponding calibration report is provided. This is possible because the materials that make up these cameras are not predisposed to deformations due to weather conditions. In another way, UASs have low-cost cameras and can have several fluctuations during aerial coverage.

When a UAS camera is not pre-calibrated, the calibration parameters are calculated by Brown's distortion model (Gonçalves & Henriques, 2015). The mathematical model to perform self-calibration is an extension of the collinearity equations (Fraser, 1997). The camera lenses induce radial and decentering distortions, which need to be corrected. It is worth noting that the collinearity principle assumes that the center of exposure in the terrain $O = (X_0, Y_0, Z_0)$, a point in the image coordinate system $a = (x_a, y_a)$ and the terrain coordinate system $A = (X_A, Y_A, Z_A)$ lay on the same line (Figure 6a). For the point $a' = (x'_a, y'_a)$, Δr and Δt are the radial and decentering distortions, respectively (Figure 6b). In the image coordinate system, the mathematical model is given by:

$$\begin{cases} x'_a = x_a(1 + k_1r^2 + k_2r^4 + k_3r^6 + k_4r^8) + (p_1(r^2 + 2x^2) + 2p_2xy)(1 + p_3r^2 + p_4r^4) \\ y'_a = y_a(1 + k_1r^2 + k_2r^4 + k_3r^6 + k_4r^8) + (p_2(r^2 + 2y^2) + 2p_1xy)(1 + p_3r^2 + p_4r^4) \end{cases}$$

where $r = \sqrt{x_a^2 + y_a^2}$, (k_1, k_2, k_3, k_4) are the radial distortion coefficients and (p_1, p_2, p_3, p_4) are decenter distortion coefficients.

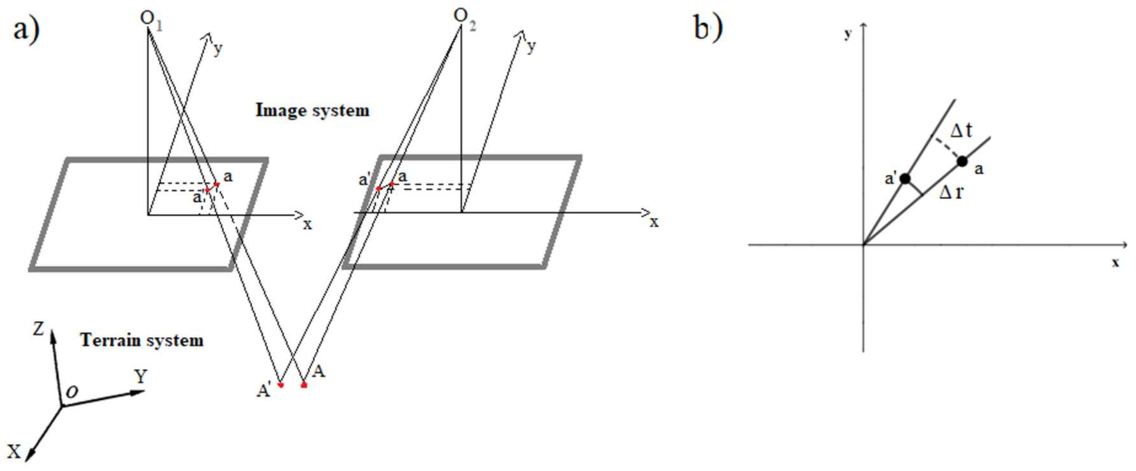


Figure 6. Lens-inducted distortions. a) Impact of the camera calibration on the terrain point location; b) radial and decentering distortion, Δr, and Δt respectively, on the point image location

The optimization of the camera distortion parameters aims to minimize the reprojection error (see section 3.2.4.2; Zhou et al., (2012)). Eliminating points with greater error, the camera distortions need to be adjusted to new points (sparse filtered cloud – see section 3.6.6). Thus, with the refined parameters, fewer errors are obtained in the calculation of the 3D model.

In particular, the self-calibration included in the Alignment of the Metashape workflow requires the choice or not of the adaptive model (see section 3.1.2). The difference lies in the inclusion of the affinity and orthogonality (b_1 and b_2 respectively) of the pixels when the adaptive model is selected.

3.2.4 Evaluation metrics

Given a SfM processing, it is necessary to design metrics to evaluate the parameters obtained in the BBA. The use of statistical indicators that measure, for example, the variability between estimated points and their true values is a key factor to validate the estimation (case of RMSE assessment in check points (CHPs) and GCPs – section 3.3.1).

3.2.4.1 Root mean square error

Root mean square error (RMSE) is a measurement to evaluate the difference between the projected and the real values. For a one-dimensionally variable, the RMSE is given by

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i - x_i)^2} \tag{2}$$

Where $\hat{x}_i$ and x_i are the i-th predicted and real value, for a given operation. To evaluate the alignment of a model consisting of 3D points of the form $p_i = (x_i, y_i, z_i)^T$ and extending the equation (*2*), we obtain (Taddia et al., 2020)

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2 + (\hat{z}_i - z_i)^2]}$$

3.2.4.2 Root mean square reprojection error

The reprojection error is an important metric to assess the accuracy of the 3D reconstruction (Nguyen et al., 2012). Reprojection error quantifies the difference between points on the sparse cloud and the same points reprojected using the camera parameters. In other words, the distance between a point observed in an image and its reprojection is measured through the collinearity equations. In practice, the measurement consists in a derivation of (*2*) considering image projection and reprojection coordinates. So, RMS Reprojection Error (RE) is given by

$$RE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left[\left(x_i^{reproj} - x_i^{proj}\right)^2 + \left(y_i^{reproj} - y_i^{proj}\right)^2\right]}$$

3.3 Impact of the image acquisition geometry

The acquisition geometry of vertical coverages can be a key factor in occlusions reduction. In this context, the most common geometries are derived from either laser techniques or photogrammetric methods. The laser techniques, known as Light Detection and Ranging (LiDAR), might be placed in the terrain (depending on the profile, one placement may not be enough) or in an aircraft. The photogrammetric methods are currently using drones because of the difficult accesses of the terrain and for the safety of the operator.

The use of nadiral images, for the reconstruction of a cliff, proved to be irrelevant since the final point cloud presented many zones without points (Jaud et al., 2019). The possibility that drones offer to change the angle of the camera becomes fundamental for the imaging of vertical structures. In fact, the variation of the angle of inclination of the camera has been adopted in several contexts such as in urban areas (Rupnik et al., 2014; Xiao et al., 2012) and coastal cliffs (Gómez-Gutiérrez & Gonçalves, 2020; Jaud et al., 2019). Combining nadiral and oblique images reflects on the improvement of accuracy of the 3D model (Manfreda et al., 2019; Rossi et al., 2017) and for stepped areas (Trajkovski et al., 2020). On the other hand, by placing the 90º angle off nadir for vertical cliffs, a high density of points in the reconstruction is obtained (Gómez-Gutiérrez & Gonçalves, 2020). In short, the image acquisition geometry of a cliff has impact on the geometric accuracy of BBA, on the density of points, and the reduction of zones without data.

3.3.1 Geometric accuracy of the BBA

Regarding GCPs errors after alignment, the key of the analysis is to evaluate 3D transformation from different reference systems. There are three reference systems: image system, ground system, model system. When the operator identifies the location of control points in each image it is saved the image coordinates from GCPs. This action represents the addition of another pair of equations to the mathematic model. Furthermore, the 3D transformation parameters are calculated from the external orientation parameters of each image. The quality of this transformation is evaluated through CHPs that are points in the same coordinate system as GCPs distributed for the study area but independent of the transformation, that is, CHPs are not used in the optimization. In summary, the RMSE of both GCPs and CHPs are calculated from the back projection between the object and image system (see section 3.2.4.1).

3.3.2 Point cloud density and accuracy

In terms of point cloud analysis, the methods are not simple and can be derived, for instance, with a reduction of dimensionality (Tonini & Abellan, 2014). The best scenario is to choose or implement a three-dimensional process in order to maintain credibility and consistency of results. For example, in the reduction of dimensionality points that belong to a line orthogonal to the projection plane may overlap.

One of the most adopted density approaches is the density based on the nearest neighborhood. Given a length for a sphere diameter it is required to count the neighbor points that fall in this sphere and repeat this process to all points keeping the same diameter (e.g. CloudCompare – see section 3.6.7). Another possibility is to create a 3D grid across the points and count how many points fall in an element (see section 3.4).

3.3.3 Point cloud gaps

Due to shadows, vegetation or lack of overlap, a given area in the object is not visible in at least two images (blind spots). Thus, in point cloud arises gaps that are areas without points. Of course, that a point could always have gaps, by definition. Therefore, the identification of meaningful gaps is directly linked to points distribution and density.

3.4 Voxelization of a 3D point cloud

When it is pretended to analyze a 3D point cloud volumetrically, voxelization becomes more appropriate (Alsadik et al., 2014). The idea of turning points into voxels is made by calculating the 3D histogram (H) of the point cloud (APPENDIX A - *voxelization.m* function). The bins will be vectors that represent the partition of the points along the XYZ directions with equal spacing between them. H represents a three-dimensional matrix in which each element H_{ijk} contains the number of points in the bins. Thus, two variables can be derived from this result: (i) 3D binary images (Figure 8c); (ii) voxels density (e.g. Figure 26c).

To transform H into a 3D binary image, it is only necessary to assign each element 1 (true) if it contains points or 0 (false) otherwise. This image is used for the application of the method of morphological analysis (see section 3.5).

To calculate the voxels density, it is, firstly, computed the volume of a simple voxel (using the spacing between the bins). Then all elements of H are divided by the voxel density resulting in a 3D float matrix (D) in which each element D_{ijk} contains the density of the voxel ijk.

3.5 Morphological analysis for gap detection

Morphological analyses are topological operations that focus on the definition of set. Particularly, in digital images, these operations are implemented either with a given pixel characteristic or with all pixels. The objective is to calculate the closest path between two groups of pixels using the distance transform. Thus, the pixels that define the path will be assigned as gap pixels. First, some basic concepts about binary images such as pixel neighbourhood, path definition, and city block distance will be explained.

In binary images, there are only two possible values for the pixels, let's say $V = \{0,1\}$. Let $T: P \rightarrow V$ defined by

$$T(P) = \begin{cases} 1, & if\ each\ p_i \in P\ satisfy\ C \\ 0, & otherwise \end{cases}$$

where P is a set of all pixels and C is a given condition for the pixels in P. Two pixels in $T(P)$ are connected if they are adjacent and share the same value. Usually, it is used the adjacency considering 4 or 8 neighbours like is presented in Figure 7. For example, p_1 is assigned as neighbour of e considering 8-connectivity (Figure 7b) and with 4-connectivity p_1 is not assigned as a neighbour of e.

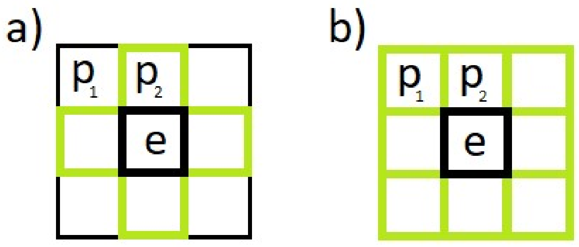


Figure 7. Pixel connectivity a) with 4 neighbours; b) with 8 neighbours

Deriving from this definition comes the concept of path. For all a and b pixels in $T(P) = 1$ and $a \neq b$, it is possible to define a path starting at a, passing only points in $T(P) = 1$, and ending in b given some connectivity. For instance, in Figure 8a-b are described the two possible scenarios for the pixels display where it is visible that the image from Figure 8c does not satisfy the definition of path.

Finally, the definition of distance between pixels ($d(a,b)$) follows the 3 axioms of the classic definition of distance: non-negativity ($d(a,b) = 0$ if and only if $a = b$), commutatively and $d(a,c) \leq d(a,b) + d(b,c)$. From a group of metrics, it can be highlighted the "City Block" distance defined by

$$d_{CB}(a,b) = |a_1 - b_1| + |a_2 - b_2|$$

where $a = (a_1, a_2)$ and $b = (b_1, b_2)$. This metric can be used to calculate the transform of a binary image distance that represents the calculation for each false pixel of the distance to the nearest true pixel (Maurer et al., 2003).

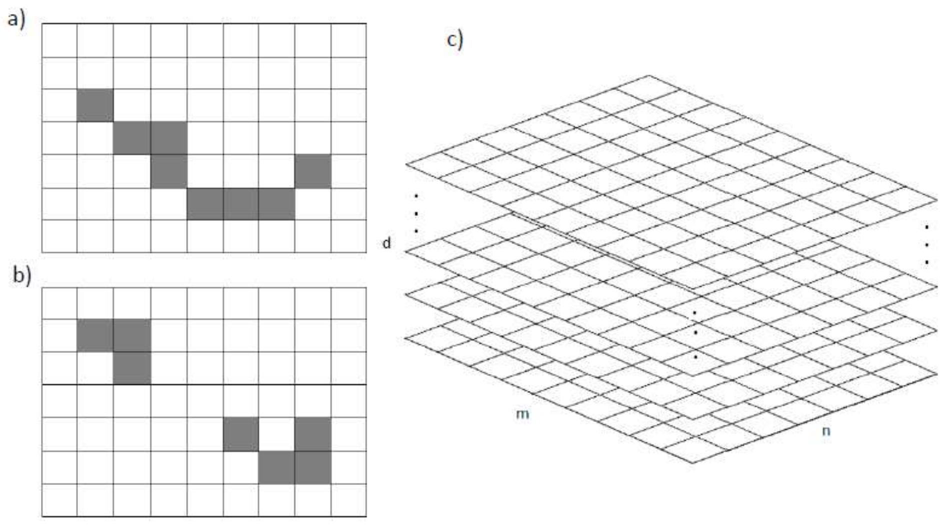


Figure 8. a) binary 2D image with one group of pixels; b) binary 2D image with two groups of pixels; c) 3D matrix that represents a 3D image. Note that the grey pixels are the true values

With these well-defined concepts, a methodology can be created to identify the gap. Given a 3D binary image, created by the voxelization method (section 3.4), with m×n×d elements (Figure 8c), we start by testing for each k level of d (number of partitions across z axis - altitude) if there is only a group of "true" pixels. For levels that do not satisfy this condition, we take groups two to two and estimate the shortest path until there is only one group (Figure 9).

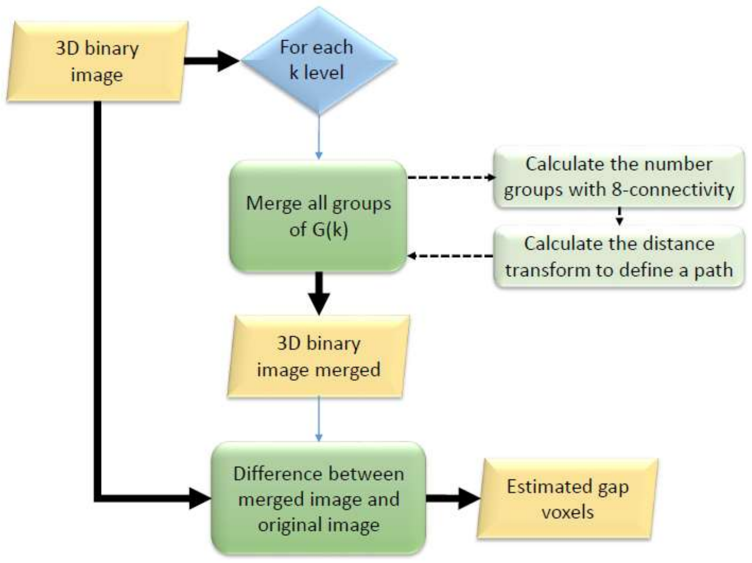

3D binary image
For each k level
Merge all groups of G(k)
Calculate the number groups with 8-connectivity
Calculate the distance transform to define a path
3D binary image merged
Difference between merged image and original image
Estimated gap voxels

Figure 9. Flowchart to represent the methodology for detecting gaps. Notation: | G(k) | is the number of groups of a level k.

This shortest path is calculated by taking two consecutive groups and calculate the distance transform, for each group individually (D_1 and D_2; Soille (2003)). Adding D_1 and D_2, a 2D array (D_{sum}) is obtained. It represents the cost surface between these groups.

Finally, all elements of D_{sum} are merged with the minimum values in order to travel between the two groups with 8-Conectivity. These new pixels are classified as gap voxels.

3.6 Optimizing Metashape workflow for 3D reconstruction

For an in-depth study about the impact of the image acquisition geometry and Metashape processing parameters on the 3D reconstruction of a dense cloud, seven phases are considered and described in the next sections (Figure 10).

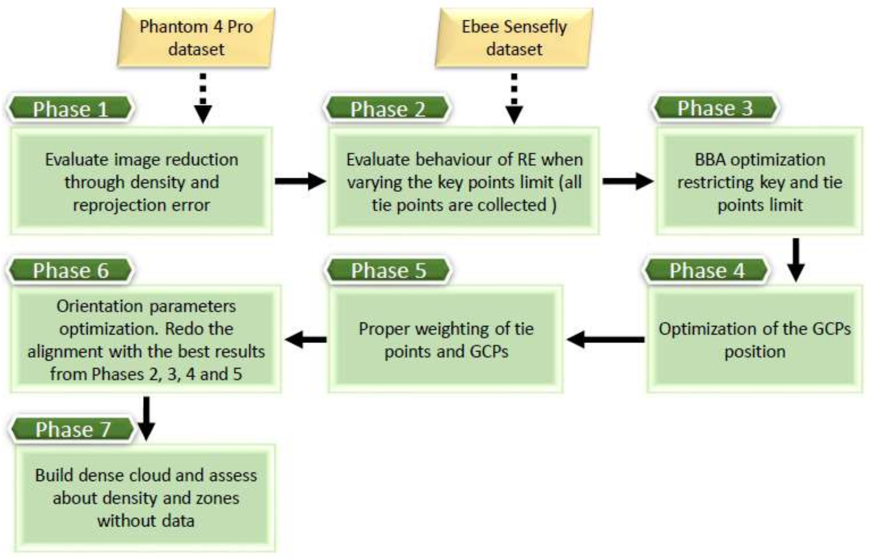

Phantom 4 Pro dataset
Ebee Sensefly dataset
Phase 1
Evaluate image reduction through density and reprojection error
Phase 2
Evaluate behaviour of RE when varying the key points limit (all tie points are collected)
Phase 3
BBA optimization restricting key and tie points limit
Phase 4
Optimization of the GCPs position
Phase 5
Proper weighting of tie points and GCPs
Phase 6
Orientation parameters optimization. Redo the alignment with the best results from Phases 2, 3, 4 and 5
Phase 7
Build dense cloud and assess about density and zones without data

Figure 10. Flowchart for the optimization of the 3 D reconstruction of a coastal cliff using two different UAS (see section 3.6)

Looking at the high number of images and the need to create several projects in Metashape, it was chosen to use the Metashape Python API (MPA; Figure 4). In each aerial survey a default project was created to pick the control points in the images. Then, a XML file (in Metashape File>Export>Export markers) was exported with all the information needed to extract the image coordinates of control points. This process saved operation time and provided consistency on the image coordinates because they were always the same in all projects (APPENDIX B).

In MPA, photo alignment is composed by two functions: 1) *matchPhotos*; 2) *alignCameras*. These two functions represent, in the graphical user interface (GUI), the stage of Align Photos (see Figure 4). Thus, with the image coordinates from the GCPs, it is enabled to make the dense matching of the sparse cloud with the application of two functions, successively, *buildDepthMaps* and *buildDenseCloud* (in GUI Build Dense Cloud).

3.6.1 Phase 1: Image number reduction

Given the Phantom 4 Pro dataset, the image reduction was evaluated applying the reduction of images with the Metashape default parameters (see section 3.1.2). A polygonal mesh from the sparse cloud (exterior and interior orientation parameters of the image block) was built to identify the useful images and remove the redundant images.

Besides, the dense clouds were obtained from the reduced and non-reduced images sets. The functionality of the "Reduce overlap" tool was assessed through the impact of the RE (section 3.2.4.2) and RMSE in the control points (section 3.3.1) and the volumetric density between dense clouds (see section 3.6.7).

3.6.2 Phase 2: RMS Reprojection error and tie points number

Initial images datasets were oriented varying key points limits and find all tie points in each image. The process was evaluated counting the maximum number of points in the sparse cloud for a given megapixel number and computing the RMS reprojection error (RE) with and without filtering. Successively, a refinement procedure eliminated the points with higher reprojection error and adjusted the parameters to the filtered points (see Section 3.6.6). The refined output was compared with the unfiltered sparse cloud and the time spent in alignment. Finally, the number of cloud points of the sparse cloud in each iteration was also computed. Main settings of RE evaluation are summarized in Table 1 with markers and tie points weights values as default.

Table 1: Processing strategies for evaluating reprojection error in "align photos" step

	Settings
Accuracy	High
key Points Limit	From 10000 to 145000 (step is 5000)
Tie Points Limit	0
Model Fitting	False

3.6.3 Phase 3: Effect of the tie and key poins limit on the RMSE of GCPs

In Phase 3, the orientation parameters of images block were calculated changing (i) tie points limit to a set of 7 elements ($tp_{set} = [1000,2000,3000,4000,5000,6000,7000]$) and (ii) key points limit to a set of 5 key points($cp_{set} = [30000,40000,50000,60000,70000]$). We evaluated the transformation between model and object system. All points stay as control points and no checkpoints were used. The “adaptive camera model fitting” was also tested (Table 2). When the model fitting is true, affinity and orthogonally are also calculated.

Table 2: Processing strategies for evaluating RMSE in “align photos” step

	Process 1	Process 2
Accuracy	*High*	*High*
Tie Points Limit	tp_{set}	tp_{set}
Key Points Limit	cp_{set}	cp_{set}
Model Fitting	True	False

3.6.4 Phase 4: GCPs distribution and 3D model accuracy

Given the optimum tie points and key points limit, the influence of GCPs distribution on the model was assessed. So, the idea is to distribute the points in three levels (Figure 1b). The first two approaches only considered GCPs on the beach and on the top of the cliff. One more approach considered the GCPs within monitored area (Table 3). The selection criteria aimed to find the best GCPs and CHPs combination in terms of RMSE. For this purpose, we calculated the mean weight between the GCPs and CHPs, that is, the weights were attributed by the number of each point type. For instance, in Approach 1, the weights for GCPs and CHPs are 5 and 15, respectively.

Table 3. Combination of different GCPs and checkpoints to evaluate the RMSE on geo-referencing of the sparse cloud

Scenario	*GCPs label*	*CHPs label*
1 (5 GCPs, 15 CHPs)	2,4,6,20,23	1,3,5,7,8,9,10,11,12,13,14,15,16, 21,22
2 (10 GCPs, 10 CHPs)	1,2,3,4,6,7,20,21,22,23	5,8,9,10,11,12,13,14,15,16
3 (15 GCPs, 5 CHPs)	1,2,3,4,6,7,8,10,12,14,15,20,21, 22,23	5,9,11,13,16

3.6.5 Phase 5: GCPs and tie points weights on BBA

With the best GCPs distribution derived for optimum tie points and key points limit, the weights of the GCPs (w_{GCP}) and tie points (w_{tp}) were assessed. In terms of GCPs, in the ground system, weights were imported as coordinates of the GCPs and assigned through your HSDV and VSDV (Figure 5a). The other two weights were grouped into two sets that were combined. By default, Metashape uses 0.5 and 1.0 pixels to weight the image coordinates of GCPs and tie points, respectively. The idea is to evaluate the impact of those weights in the GCPs and CHPs RMSE and, then, chose the appropriate values. So, the weights in test are $w_{tp} = [0.1,0.5,1.0,1.5,2.0,2.5]$ and $w_{GCP} = [0.1,0.5,1.0,1.5]$

3.6.6 Phase 6: Interior orientation parameters optimization

With appropriate parameters to minimize GCPs and CHPs RMSE, we assessed the best interior orientation parameters. A threshold of 0.4 was applied to remove tie points with higher reprojection error. After that, the orientation parameters were readjusted using the default model fitting.

Tie points visible in three or fewer images may contribute with higher reprojection errors. In this context, the impact of image count observation (ICO) was evaluated in terms of RE. For example, points with 3 or less ICO, will be removed by applying a filtering (or refinement) operation. In summary, the proposed workflow will include two additional stages:

1. Refinement of tie points;
2. Readjustment of orientation parameters

3.6.7 Phase 7: Point cloud density and gaps

Given the SfM-MVS optimized for 3D reconstruction of the cliff, the aim was to evaluate the volumetric density of the two point clouds resulting from Ebee Sensefly and Phantom 4 Pro as well as the existence of gap areas.

In CloudCompare software (CC), the “Density” tool calculates some measurements to evaluate the spatial density of points. There are two input methods and only one can be choose of the following options: ”Precise”- radius r of a circle or sphere; ”Approximate”- distance to the nearest neighbor (particular case for just one neighbor). So, in theory, there is a set of M points and for each p_i is calculated the number of neighbors falls into a circle or sphere of radius r centered in p_i, named N. After this, three outputs are possible:

1) only the N neighbors (only available for “Approximate” method);
2) neighbors per area (A)

$$S = \frac{N}{A}\ , \text{ where } A = \pi r^2$$

3) neighbors per volume (V)

$$S = \frac{N}{V}\ , \ where\ V = \frac{4}{3}\pi r^3$$

Note that each p_i, in the final results, has a new attribute that represents one of these possible outputs. Thus, the comparison of volumetric density between the two dense clouds was done with the CC's "Density" tool calculating the number of neighbors per m^3.

In terms of point cloud gaps, the objective is to locate and quantify volumetrically the gaps considering point cloud density. The idea was to transform the points into voxels applying the methodology explained in section 3.4. Three spatial resolutions were chosen for the voxels. (1 m, 0.5 m, and 0.25 m). Then, with the use of morphological, the gap voxels were detected and quantified (see section 3.5). The visualization of all voxels was made by their respective centers.

Considering that the gap identification will be implemented using voxelization method, the density provided by this method was compared with CC density because the neighborhood shape (cubes (voxels) versus spheres) was different.

To visualize the density provided by voxelization method, 3D plots were created according with CC visualization. In CC, the color of each point is assigned based on its density. For the density of voxelization method, points that fall into a voxel have the same density. Thus, it was necessary to locate and assign them the density of the respective voxel. To visualize the gap voxels location, each voxel was represented by its centroid.

4 Results and discussion

In this section, all the results covered in section 3.6 will be exposed and discussed. Due to the processing time in BBA, phase 1 was applied only to the set of images from Phantom 4 Pro. It was proved that, with the reduction of images, the results were the same as having used all of them (section 4.5.1).

The workflow scenarios involved several software (Agisoft Metashape, MATLAB and a Python Integrated Development Environment – Spyder IDE; APPENDIX 0 and B) and implementation strategies. Metashape API combined with Spyder IDE allowed to couple multiple operations like iterative processes and statistical analysis. In MATLAB, the voxelization method was implemented where functions of 3D spatial analysis are well defined.

4.1 Impact of the number of key and tie points

4.1.1 Reprojection error

With the initial alignment of images, the relationship between key points limit and reprojection error was assessed (Figure 11). The number of tie points tended to remain constant after 30000 and 80000 key points, for Ebee Sensefly and Phantom 4 Pro respectively. The time spent on the calculation of orientation parameters indicated that processing times increased while reprojection error tended to remain constant.

After the application of the optimization procedure (section 3.6.6), the reprojection error reduced almost 0.1 pixels, although the behavior of the curves was inverse (decreasing for Ebee Sensefly (Figure 11a) and increasing for Phantom 4 Pro (Figure 11b)).

The comparison between key points and sparse cloud point numbers shows that the curves increased until a common global maximum of 130000 and 110000 points in sparse cloud, for Ebee Sensefly and Phantom 4 Pro, respectively. With filtering, the number of points in the sparse cloud remained almost the same (Figure 11c-d).

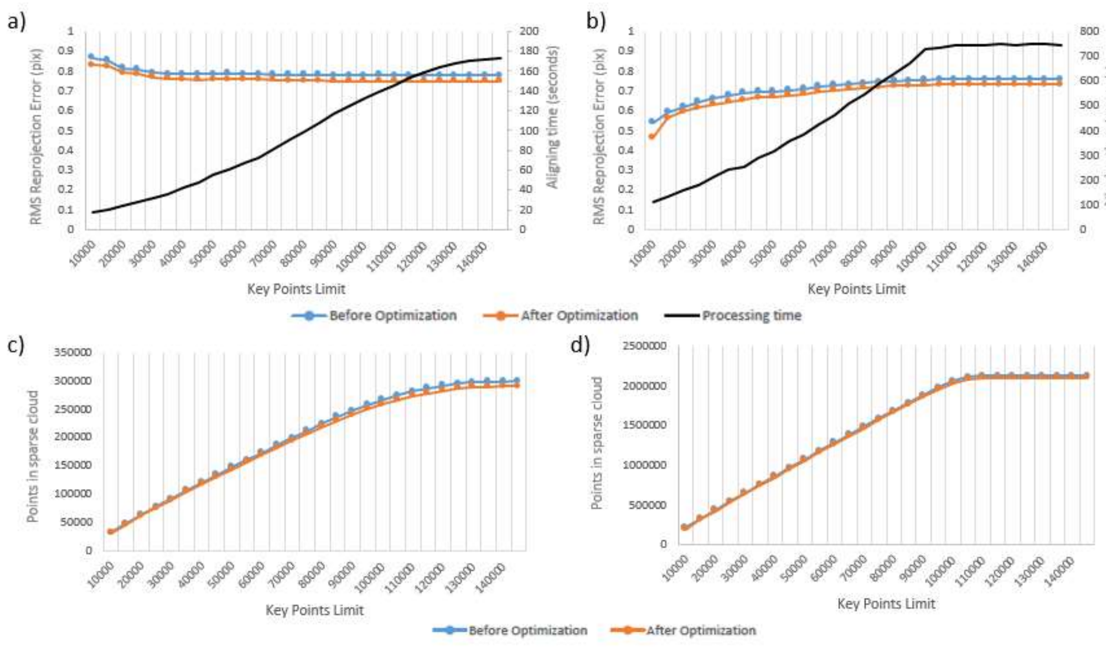


Figure 11. Reprojection error for tie points considering the tie points limit on: a) Ebee Sensefly data; b) Phantom 4 Pro data. The number of points in the sparse cloud considering the key points limit recording for: c) Ebee Sensefly data; d) Phantom 4 Pro data. In this case, it was applied the interior orientation parameters optimization (see Section 4.3.6)

Considering results from Phases 2 and 3, several key points limit were used to detect features of interest in the images. To assess the effect of key and tie points number on the 3D model accuracy, the RMS reprojection error and the RMSE of phases 2 and 3 were merged and grouped by the key points limit (Figure 12). Unexpectedly, phase 2 showed lower errors than phase 3 except for Phantom Pro RMSE where it remained constant. The owners of Agisoft Metashape software said that in general tie points limit does not influence the quality of 3D model. In fact, the RMSE of Phantom 4 Pro remains invariant along each combination. This is not true for Ebee Sensefly, where RMSE only remains the same in phases 2 and 3 in the highest tie point limits (50000, 60000 and 70000).

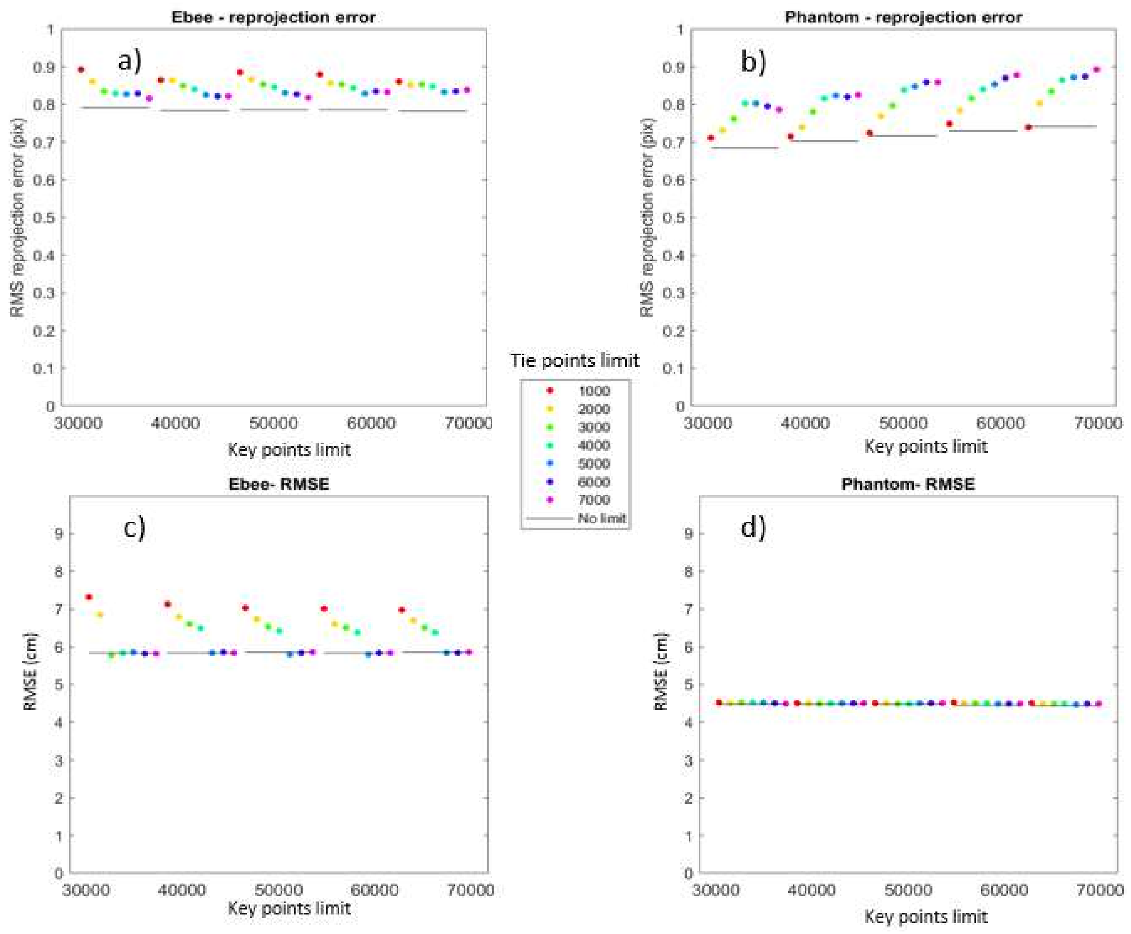


Figure 12. Comparison between the results of Phases 2 and 3. a) and b) represents the reprojection error of each adjustment for Ebee Sensefly and Phantom 4 Pro data respectively. c) and d) represents the RMSE on GCPs for Ebee Sensefly and Phantom 4 Pro data respectively. Note: back horizontal lines corresponds to the phase 2 and color points to the phase 3

The fact of not assigning a limit to tie points implies that the time in the alignment will be related to key points limit. As expected, the times are much higher in Phantom 4 Pro considering the number of images and the percentage of the cliff imaged in each image. Given the flying characteristics of the two aircrafts, the image geometry are different in both datasets (Figures Figure 2a-c and Figure 13) which leads to better results in Phantom 4 Pro. Ebee Sensefly needs a safety distance from the object to operate contrasting with Phantom 4 Pro that fits well for facades fights (Gómez-Gutiérrez & Gonçalves, 2020).

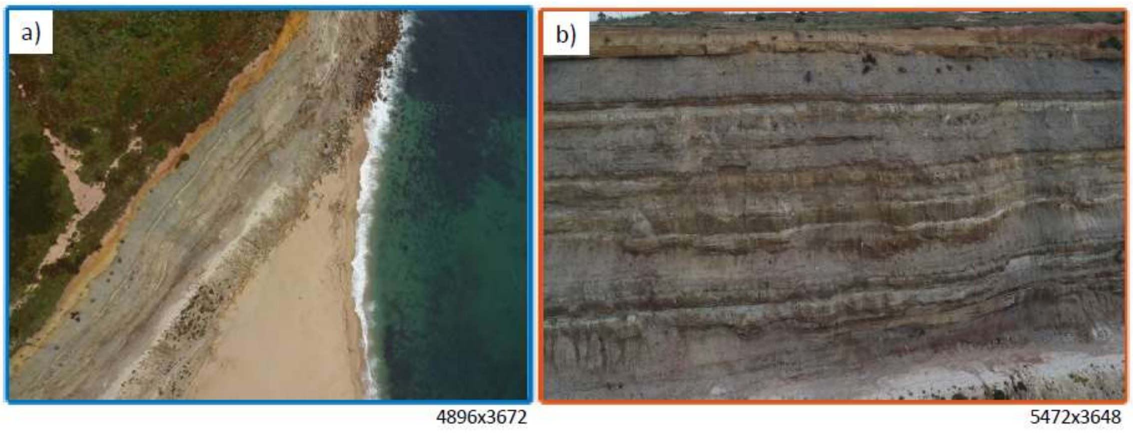


Figure 13. Type and dimensions of images acquired by: a) Ebee Sensefly; b) Phanthom 4 Pro

4.1.2 Camera orientation

Camera self-calibration was implemented in each iteration of phase 3 (section 3.6.3) providing variations in the coordinates of exposure center higher on the Ebee Sensefly., while for Phantom 4 Pro the coordinates were much less scattered around centroids (Figure 14). The groups formed by Processes 1 and 2 are well defined but for Ebee Sensefly they are a bit accurate but not very precise whereas Phantom 4 Pro are accurate and precise.

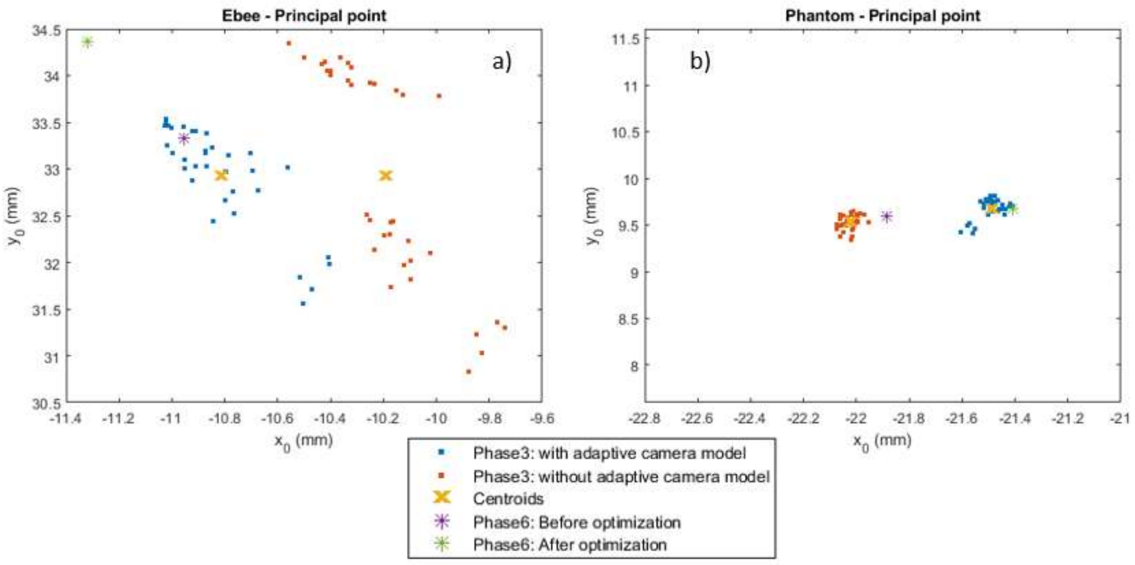


Figure 14. Principal point position combining different key points and tie points limits described in Phase 3 and the unfiltered and filtered sparse cloud of Phase 6. a) refer to Ebee Sensefly data; b) refer to Phantom 4 Pro data

Radial distortions were higher on Phantom 4 Pro than on Ebee Sensefly (Figure 15a-b), while decenter distortions were much lower on Phantom 4 Pro than on Ebbe Sensefly (Figure 15c-d). Process 1 gave about the same results as Process 2, except for radial distortions (coefficients k_2 and k_3) of Phantom 4 Pro which showed differences of approximately 0.06 mm and 0.1 mm, respectively.

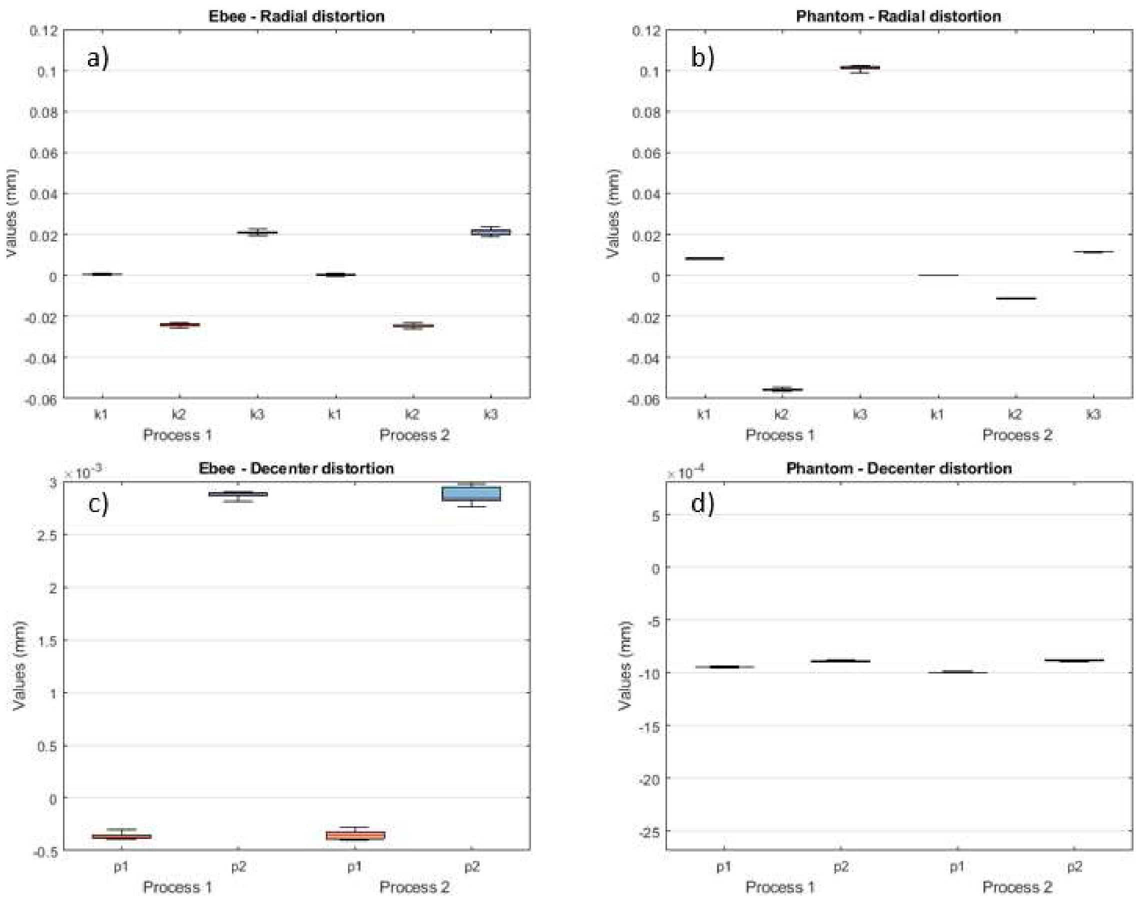


Figure 15. Radial and decenter distortions for the processes 1 and 2: a) and c) concerns to Ebee Sensefly data; b) and d) concerns to Phantom 4 Pro data (see section 3.2.3)

The estimation of camera distortions along each iteration provided a better understanding about the functionality of adaptive model fitting option. As mentioned in section 3.2.3, the affinity and orthogonality are calculated with the choice of the adaptive model fitting. The influence of these two parameters is visible in the principal point position (Ebee and Phantom) and radial distortions (Phantom). The difference in the sensors geometry and the number of pixels in images explain the discrepancies in principal point location and the decenter distortions.

4.1.3 RMSE on GCPs

The interior and exterior orientation parameters were optimized considering the GCPs coordinates. The RMSE was calculated locally and the model was adjusted based on the input GCPs. The RMSE was not varying through the iterations for both Ebee Sensefly and Phantom 4 (Figure 16). The effect of using and not using the adaptive model fitting in GCPs is almost null in both aircrafts, giving the idea of similarity between processes 1 and 2. From the errors displayed in Figure 16, the mean and the standard deviation were computed (Table 4). For the mean values, Ebee Sensefly showed higher errors than Phantom 4 Pro. The error variability was more pronounced, again on Ebee Sensefly, representing 0.30 cm and 0.49 cm when using and not using adaptive model fitting, respectively.

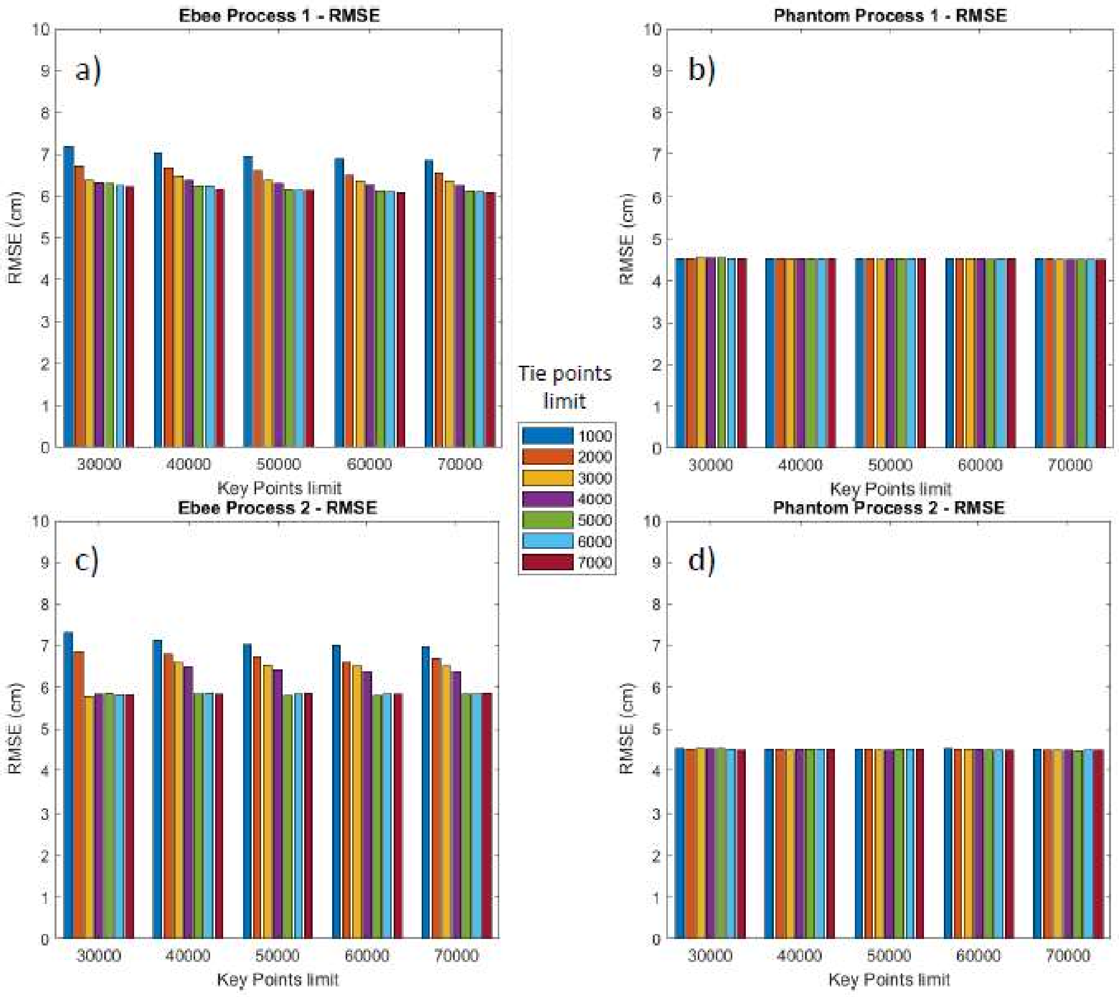


Figure 16. Root Mean Square Error (RMSE) in cm for GCPs recording to Table 2 for: a) Ebee Sensefly-Process 1; b) Ebee Sensefly-Process 2; c) Phantom 4 Pro-Process 1; d) Phantom 4 Pro-Process 2

The relationship between the variation of key and tie points number suggests a limit impact on the RMSE of GCPs. The errors have more influence in the geometry of Ebee images. With the application of the adaptive model fitting, the RMSE of GCPs remains equal. The variability of camera calibration parameters (see section 4.1.2) in processes 1 and 2 does not influence the RMSE except on Ebee when considering 4000 or less tie points (Figure 16a-c).

Table 4. Mean and Standard Deviation of the processes described in Figure 16

	Mean(cm)	*Standard Deviation(cm)*
Ebee Process 1	6.40	0.30
Ebee Process 2	6.29	0.49
Phantom Process 1	4.53	0.00
Phantom Process 2	4.51	0.01

4.2 Impact of GCPs distribution

Using the best parameters tested in Section 3.6.3, the RMSE on the GCPs and CHPs was summarized in Table 5. Overall, the CHPs error was higher than the GCPs error. The errors from Ebee Sensfly were slightly higher than the one produced by Phantom 4 Pro dataset.

While the errors associated to the 2D coordinates (X and Y) were limited to 2 and 3 cm, respectively, error in elevation (Z) was the major contribution for the Total error for both drones (Figure 17). The Z error was higher for 10 CHPs set (8 cm and 6 cm for Ebee Sensefly and Phantom 4 Pro, respectively).

Table 5. RMSE on the GCPs and CHPs for different control points combination for Ebee Sensefly and Phantom 4 Pro regarding to the Table 3. The "Total" column represents the weighted mean (number of points) between the RMSE of the GCPs and CHPs

Aircraft	*Approach*	*GCPs error (cm)*	*CHPs error (cm)*	*Total (cm)*
Ebee Sensefly	1	2.3	6.5	5.4
	2	2.8	7.7	5.3
	3	4.3	7.3	5.0
Phantom 4 Pro	1	2.9	5.5	4.9
	2	3.4	5.8	4.6
	3	4.1	5.4	4.4

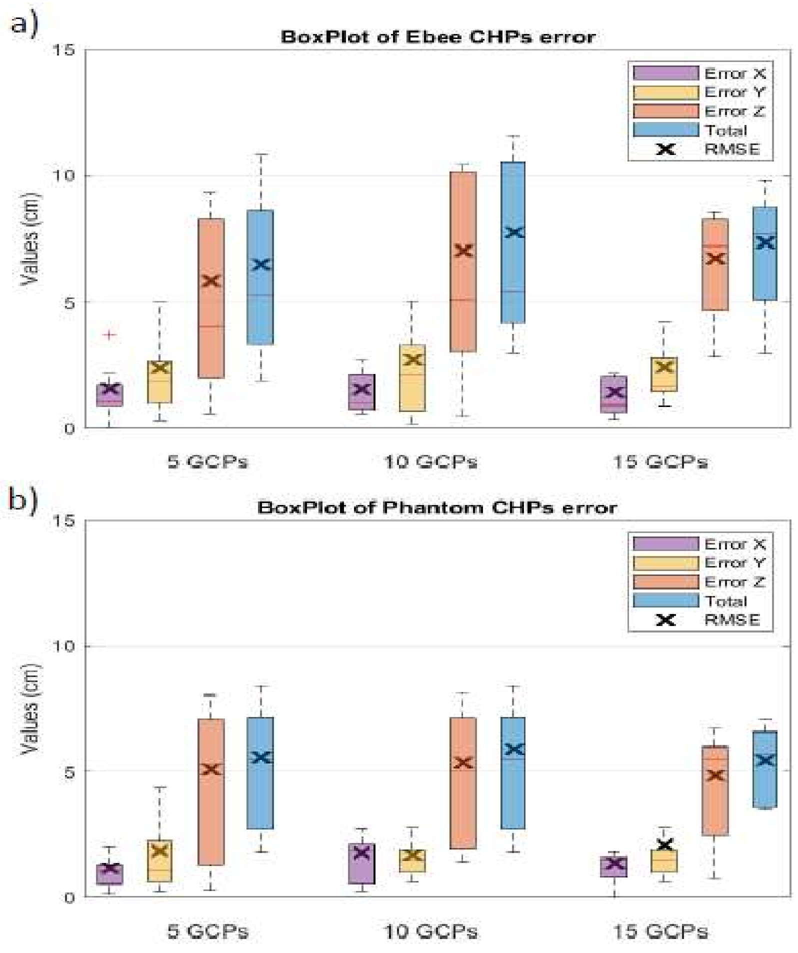


Figure 17. Boxplots that represent the error between the real coordinates and reprojected coordinates for: a) Ebee Sensefly; b) Phantom 4 Pro. The blue crosses represent the RMSE on CHPs along the respective axis

GCPs are an important element to evaluate the geo-referencing quality of the dense cloud that must be spread along the study area (Rangel et al., 2018). In cliffs, given the terrain profile and for the safety of the operator, the GCPs are placed where is possible. This study area is an exceptional case because there is an intermediate passage that allows measure control points. Recently, Gómez-Gutiérrez & Gonçalves (2020) concluded that, for the same cliff of Praia do Porto da Calada, there must be at least one GCP at the top and bottom of the cliff to substantially reduce the error in the CHPs. From this principle, the three scenarios always used GCPs at top and bottom of the cliff.

In the beginning, with fewer GPCs the error is suitable to changes while with more GCPs the error is more robust. By weighting RMSE between GCPs and CHPs, this means that we assigned more importance according to the number of GCPs or CHPs contribution. The weighted RMSE showed lower results considering 15 GCPs and 5

CHPs. This final GCPs set includes points in the 3 levels, which reveals a better adjustment of the model to the entire study area.

4.3 Impact of camera optimization and tie points refinement

The global reprojection error of the tie points, among others, gives the quality of orientation parameters. After the BBA, the two sparse clouds were filtered with a reprojection limit of 0.4 pixels ($RE_{0.4}$).

For Ebee Sensefly, the percentage of tie points removed was 5.6 % (3159 points; Table 6). From this set of points, the tie points that were observed in three or two images (I_3) was calculated representing only 2.3%. It is visible in Figure 18a that the number of image that tie points are observed is concentrated in the lower image count. In particular, in I_3, 50% of the points have RE distributed along the interval 0.043 – 0.175 pixels, and the outliers points were removed (Figure 18c).

For Phantom 4 Pro, the percentage of tie points that not satisfied the $RE_{0.4}$ condition was 15.8% (16939 points; Table 6). The points of I_3 that satisfied $RE_{0.4}$ condition was 3.2%. Finally, the distribution of the tie points observed in the images is allocated at the first 8 numbers and 50% of the I_3 values have a RE distributed along with the interval 0.057 – 0.117 pixels (Figure 18b-d).

Table 6. Statistics for the two image sets before and after the orientation parameters optimization

Aircraft	*RE Before (pix)*	*RE After (pix)*	*Tie points number*	*Tie points removed (%)*	*I_3 (%)*	*I_3 and $RE_{0.4}$ (%)*
Ebee Sensefly	0.78	0.73	56328	5.6	71.5	2.3
Phantom 4 Pro	0.81	0.69	107302	15.8	38.4	3.2

Interior and exterior orientation parameters are crucial elements to decrease the RMSE on the GCPs and CHPs (section 4.1.3). With tie points filtering, the reprojection error showed improvements. The idea that tie points of I_3 set contribute with more reprojection error is false. Without any filtration, Ebee Sensefly sparse cloud contains more than 70 % I_3 points and only 2.3 % with $RE_{0.4}$. This makes sense since the concentration of I_3 values is around 0.1 pixels (Figure 18c). Thus, the points removed of I_3 represent almost all outliers while the other removed points belong to the dual of I_3 set.

The Ebee Sensefly has a stronger imaging geometry. In terms of tie points, the greater the number of images is, where tie points are observed, more likely is the reprojection error of being high.

The Multirotor results follow the same principle of the fixed-wing, but in this case, the number of tie points removed is more representative related to total. The number of I3 is about 38% that is linked to the image overlap, the possibility of a point to be visible in more images is higher.

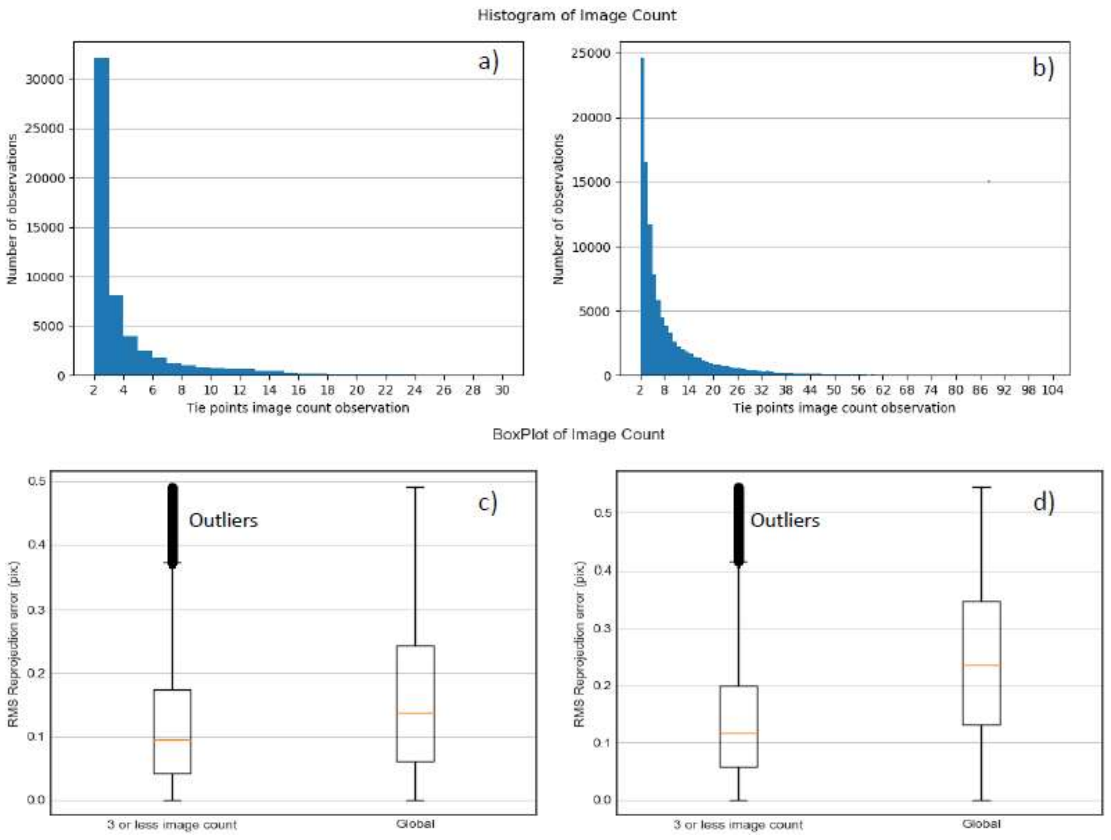


Figure 18. The number of images where tie points are visible and the RMS reprojection error of tie points visible in three or fewer images compared with the overall error.a) and c) for Ebee Sensefly and b) and d) for Phantom 4 Pro, respectively

Although the adaptive model fitting does not influence the RMSE, the camera calibration parameters presents variations mainly in principal point position (Figure 14). Comparing the use of the adaptive model fitting (section 3.6.3) and the sparse cloud filtration (section 3.6.6), some differences between the camera calibration parameters can be observed. To assess these differences, plots with the mean values of radial and decenter distortions from Figure 15 and the distortion of the phase 6 (section 3.6.6) were created (Figure 19).

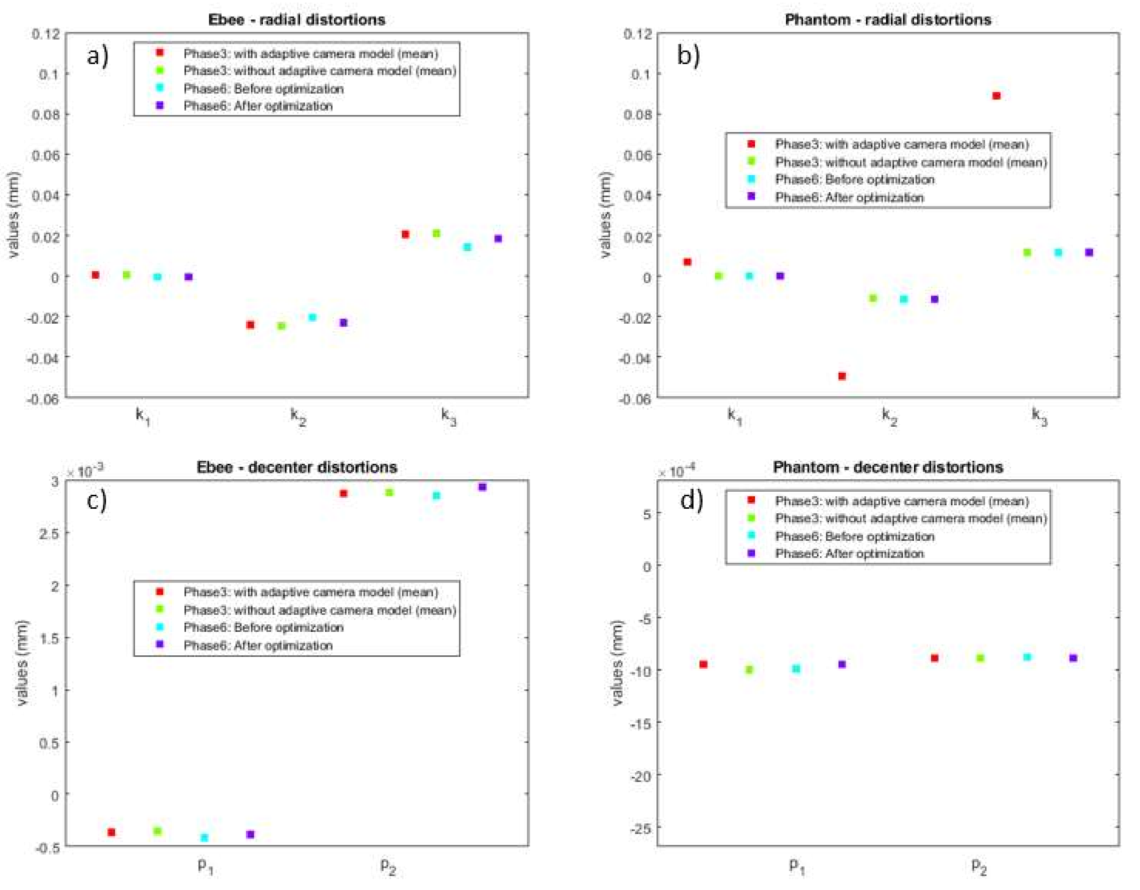


Figure 19. Radial and decenter distortions obtained in phases 3: a) and c) Ebee Sensefly; b) and d) Phanthom 4 Pro

The results suggest a higher influence of adaptive model fitting in Ebee Sensefly dataset in the principal point position (Figure 14), while in Phantom data, the impact of using adaptive model fitting is limited to the radial distortions (Figure 19b). In general, the reprojection error is influenced by the principal point position since the radial and decenter distortions remain equal.

4.4 GCPs and tie points weights on BBA

GCPs and tie points weights were assessed considering a group of 15 GCPs and 5 CHPs (Figure 1c), described in Table 3. For Ebee Sensefly, the CHPs error showed a variation of 1 cm (from 7.2 cm to 8.2 cm), while for Phantom 4 Pro, the CHPs error remained constant (5.4 cm, Figure 20c-d). For both Ebee Sensefly and Phantom 4 Pro data, the GCPs error decreased when the GCPs accuracy increased (Figure 20a-b). Note that, in general, the results were replicated when the control points accuracy was set not variable.

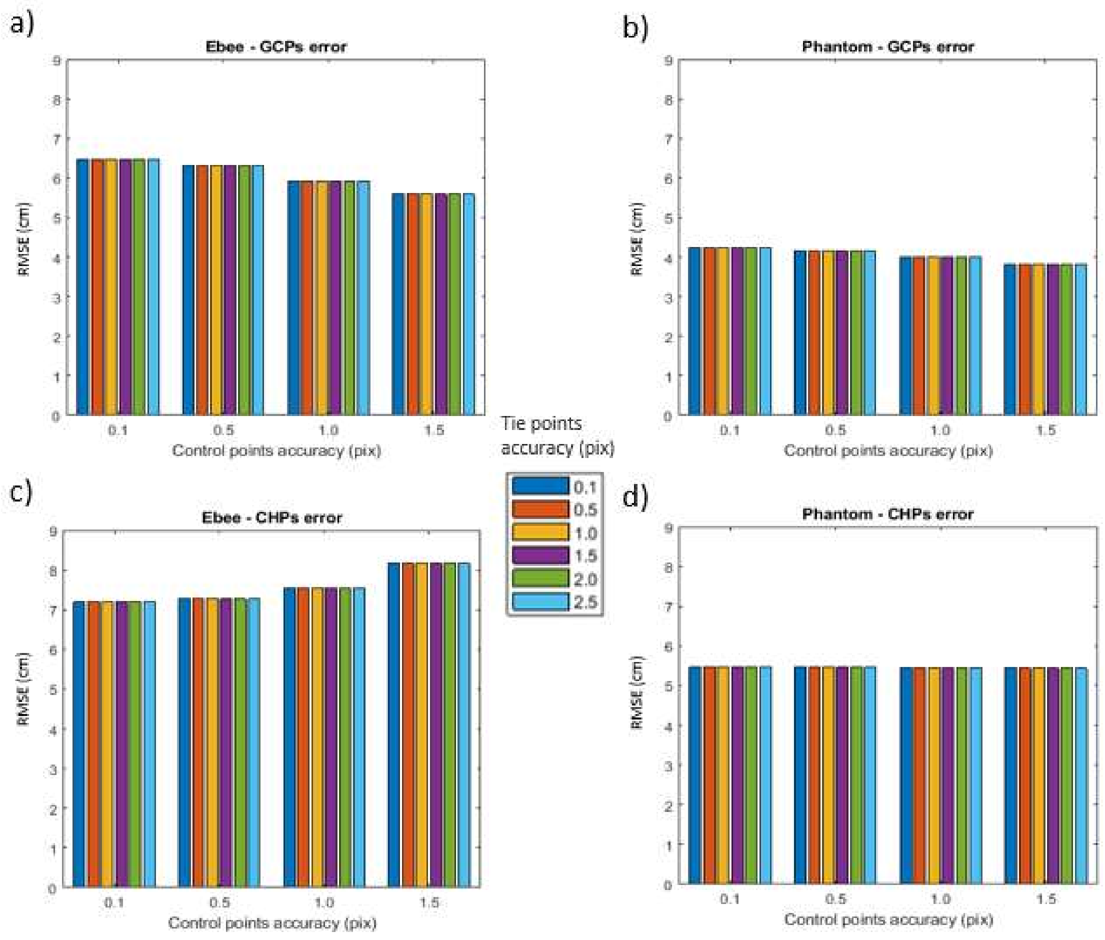


Figure 20. RMSE, for 15 GCPs and 5 CHPs using different GCPs and tie points image accuracy for Ebee Sensefly (a) and c)) and Phantom 4 Pro (b) and d))

The weighs used in BBA represent the accuracy of the control points (in the ground and image system) and tie points (in the image system). Mayer et al., (2018) concluded that the tie points accuracy influences the RMS reprojection error. In our case, this fact was not satisfied and the error was always constant (Figure 21). Therefore, the reprojection error does not depend on the accuracy of the tie points and control points.

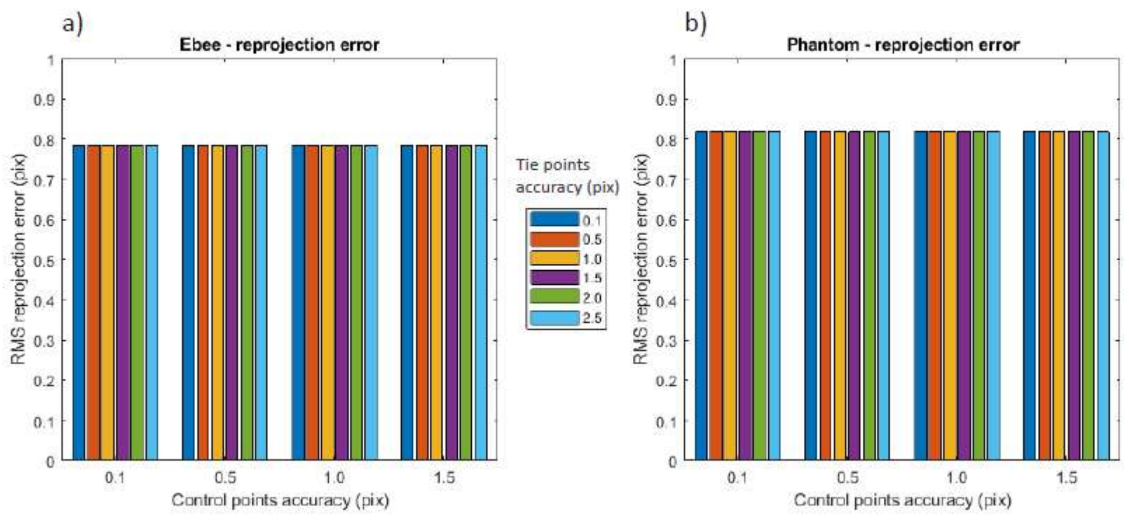


Figure 21. Impact of the tie points and control points accuracies on RMS reprojection error: a) Ebee data; b) Phantom data

On the other hand, the errors resulting from the GCPs and CHPs show some differences. In Figure 20, we can see clearly a pattern in the bar plots (errors change when the control points accuracy changes). As explained before, the control points accuracy represents the confidence which the operator marks each point. For example, saying that control points have an accuracy of 1.5 pixels means that coordinates can oscillate 1.5 pixels from those marked by the operator. Therefore, the GCPs RMSE decreases if we increase the control points accuracy.

4.5 Impact of image number reduction and point cloud density

4.5.1 Reducing the number of images

The procedure of image reduction reduced the initial dataset (448 images) to 258 images (42% reduction, Table 7). Considering all images, the reprojection error did not improve significantly (about 0.01 pixels).

Table 7. Statistics to evaluate reduced images set

	Images number	*RE (pix)*	*Tie points number*	*Dense cloud points number*
All images	448	0.80	102677	20572275
Reduced images	258	0.81	107302	20221285

The voxelization (Section 3.4) allowed to implement a function to calculate the volumetric density per m^3. Besides, we used CC with the same parametrization to compare the results (Figure 22). The volumetric density computed by voxelization function had values comprised between 0 and 1500 p/m^3, with the peak at about 800 p/m^3. The histogram of density computed with CC was instead comprised between 500 p/m^3 and 2000 p/m^3, with the major number of observations around 1400 p/m^3.

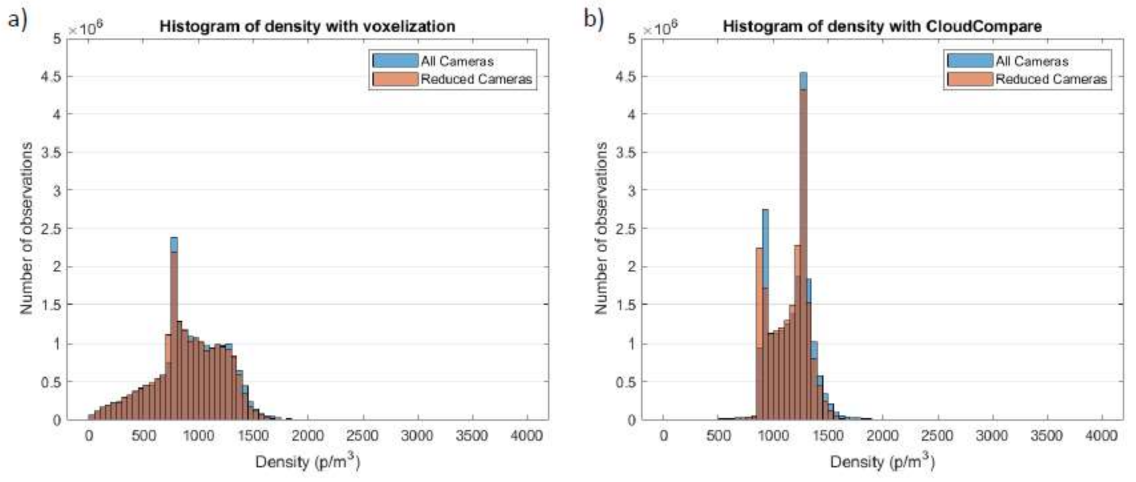


Figure 22. Histogram count for the volumetric density with equal bins spacing for Phantom4 Pro data using all images and the reduced images set: a) for voxelization density; b) for CloudCompare volumetric density

Overall, the histograms produced considering all images and the reduced images number did not differ significantly. For this reason, the results of the final local densities were similar (Figure 23 and Figure 24).

Initially, the objective of reducing the number of Phantom images was to decode the functionality of Metashape "Reduce Overlap" tool and to improve the time in SfM-MVS process. By choosing a moderate overlap, the error and the density were preserved. The final dataset counts with less than 190 images that improved substantially the processing time (section 2.3). For instance, saving 4 hours in one process means saving 140 hours in 35 processes (number of iterations of phase 3 – section 3.6.3).

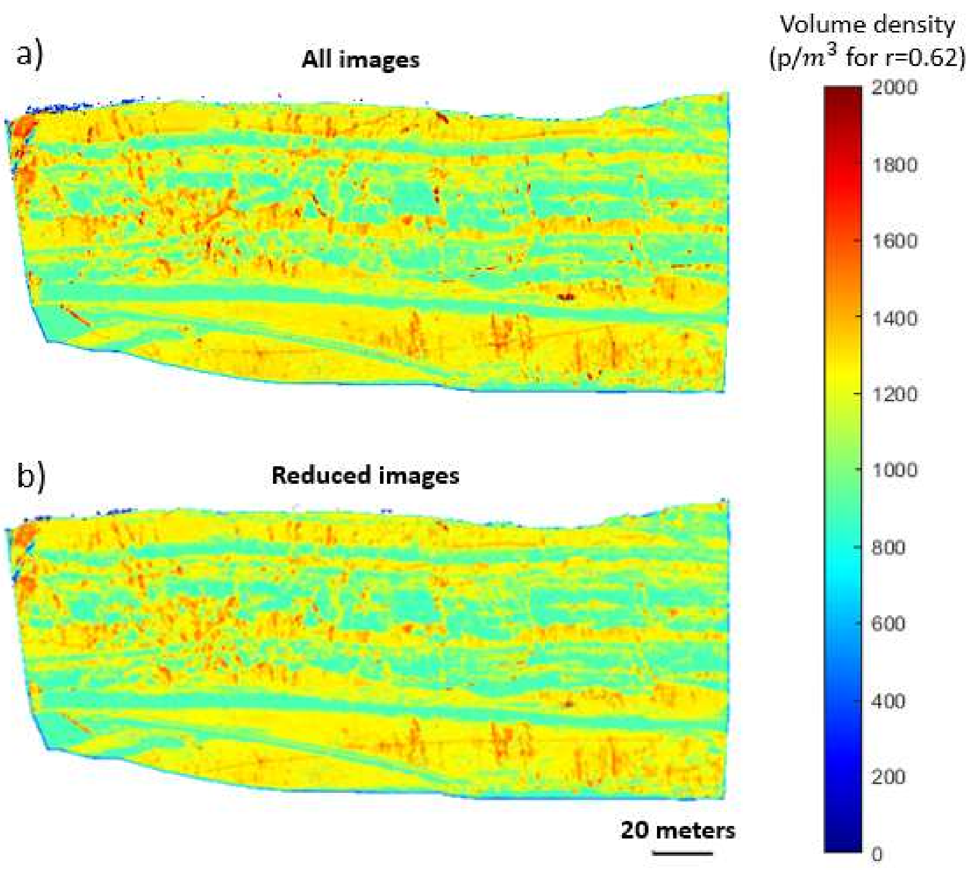


Figure 23. CloudCompare volumetric density; a) all images; b) reduced images

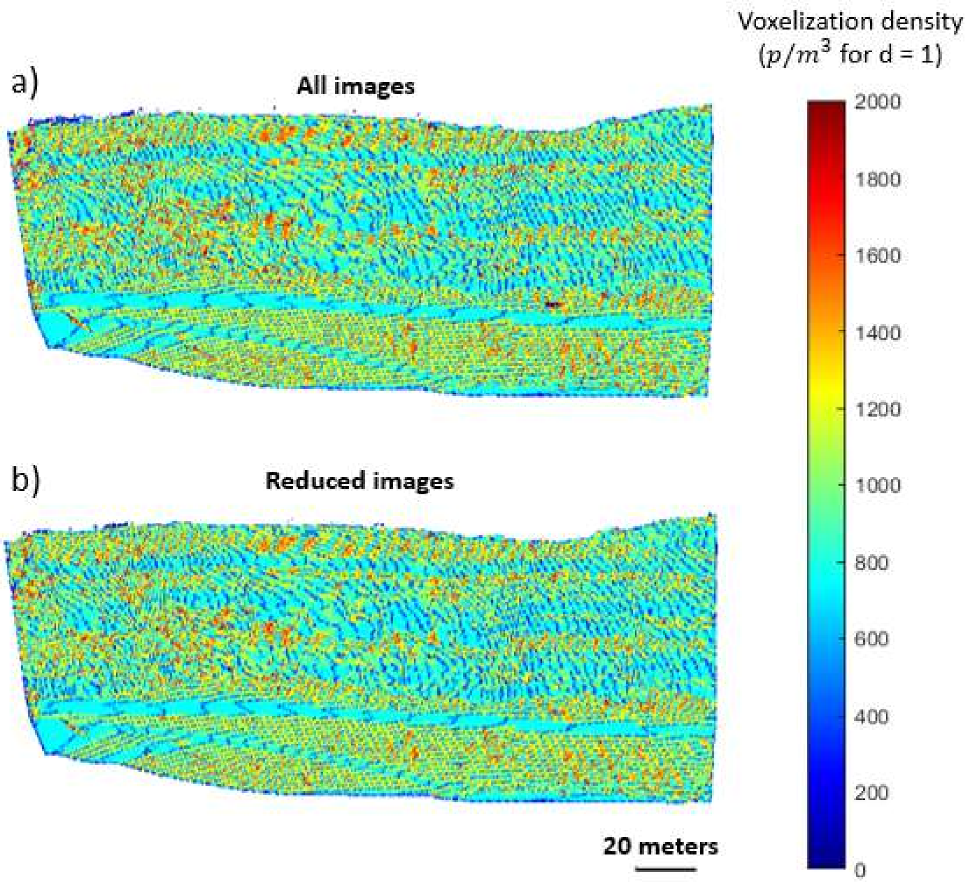


Figure 24. Voxelization density per m3 of the dense cloud generated by the Phantom 4 Pro data with: a) all images; b) reduced images

4.5.2 Analyzing point cloud density

With the optimal processing parameters, the 3D reconstruction of the cliff was performed for Ebee and Phantom datasets resulting in dense clouds of 3544055 and 20221285 points, respectively. In CC, the methodology for evaluating the point cloud density consisted in calculating the neighbors per m^3. Since the neighborhood surface is a sphere and we wanted it to have a volume of 1 m^3, the radius of this sphere was equal to 0.620 m. In this section, a reduced image set from Phantom 4 Pro was used (see section 3.6.1.).

For Ebee Sensefly data, the results were substantially different from the Phantom 4 Pro data (values between 1000 and 1500 points per m^3 - Figure 22). The density of the resulting dense cloud was comprised between 150 and 250 points per m^3 (Figure 25).

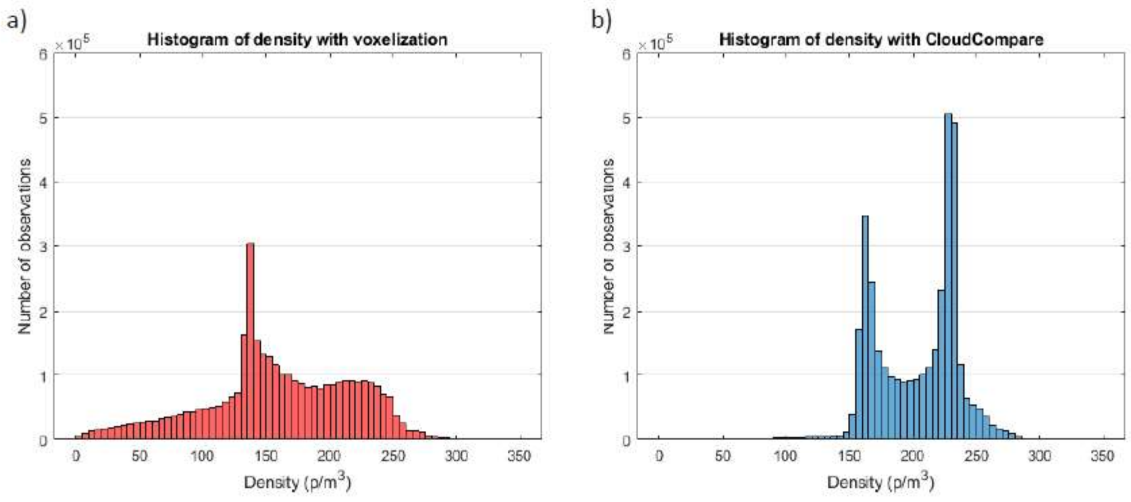


Figure 25. Histogram count for the volumetric density of the dense cloud with equal bins spacing using Ebee Sensefly data a) for voxelization density; b) for CloudCompare volumetric density

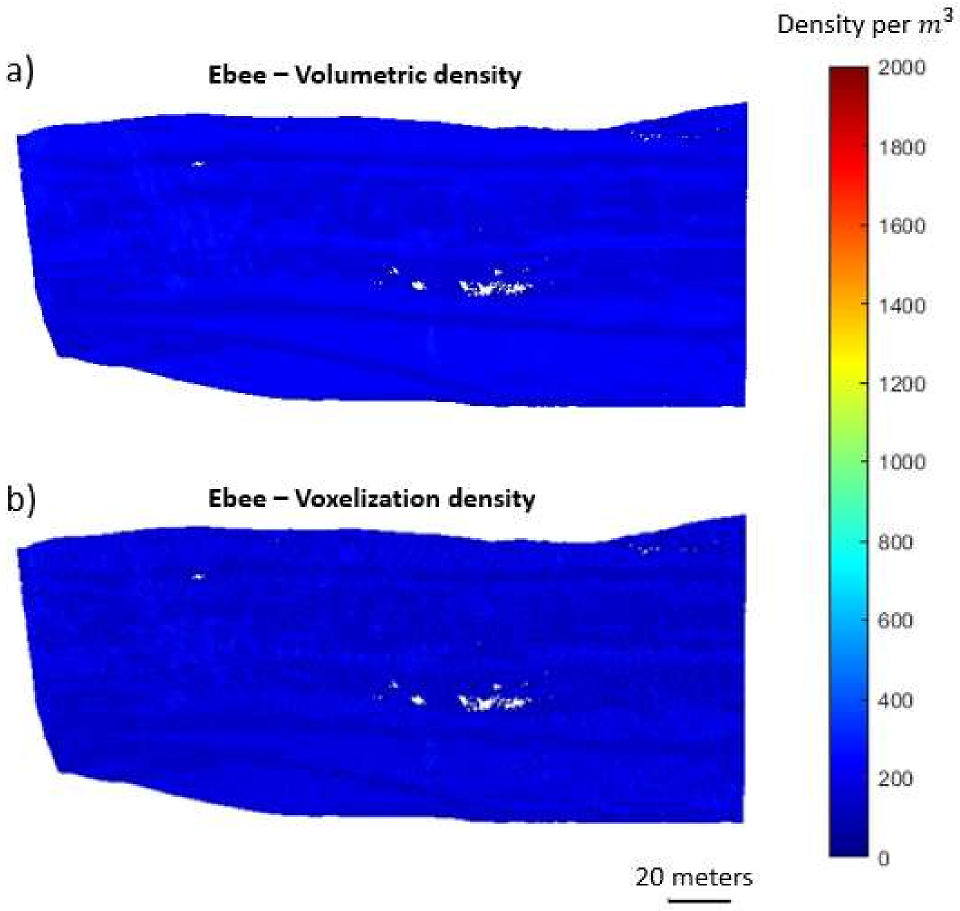


Figure 26. The density of the dense cloud generated with Ebee Sensefly data using CloudCompare volumetric density

The volumetric density, applied to dense clouds showed different results. Analyzing the data distribution, the interquartile range concentrates 50 % of the values which leads to a greater variation in the voxelization method (Figure 27). In fact, at Figure 27a and Figure 25, the histograms have values distributed across all bins. In the opposite direction, the CC density histograms are bimodal and 50% of values are more concentrated leading to a high number of outliers (Figure 27b).

In general, the CC volumetric density is always higher than those produced by voxelization method. This fact is justified by the calculation strategy. As explained in Section 3.6.7, the CC strategy results in the search for neighbors inside a sphere. For this principle, there are as many spheres as the number of points, while the number of voxels depends only on the value of the edges and their position (see APPENDIX C.1). In other words, the way of counting neighbors is the same as the voxelization (the point in sphere center is also counted) but the placement of the neighborhood shape is different. Thus, two points are neighbors if and only if their neighborhood spheres intersect, which does not happen in voxelization where two points are considered neighbors if there is a voxel that contains them (two voxels are adjacent).

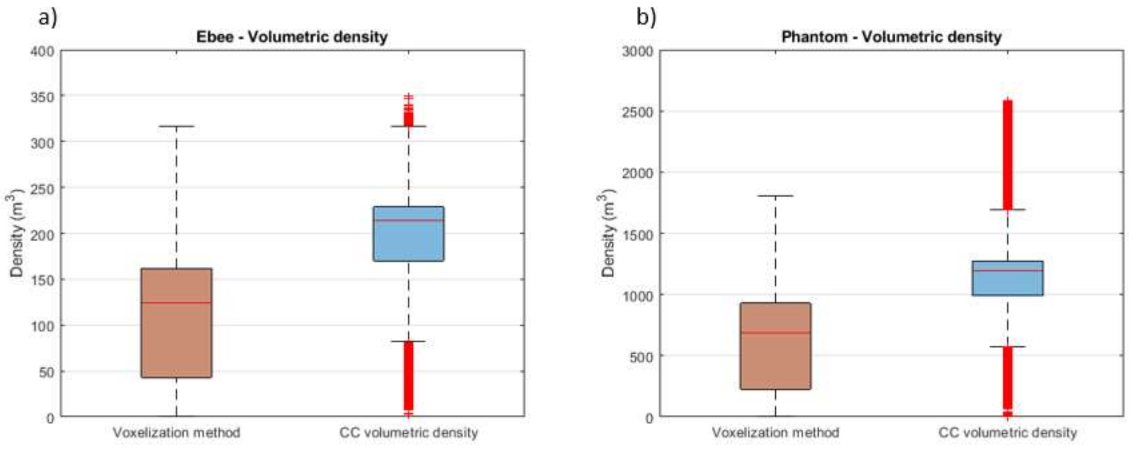


Figure 27. Boxplots represent the density distribution along with the points: a) Ebee Sensefly data; b) Phantom 4 Pro data (reduced images)

4.6 Gap detection and quantification

To identify gap zones, three voxel sizes were created (see section 3.6.7). Groups of voxels with less than 8 and 64 elements were removed in voxels with 0.125 m^3 and 0.0156 m^3, respectively, that represented groups with less than 1 m^3. Starting with the Ebee point cloud, three zones with gaps were identified. In these zones, the number of gap voxels increased when the spatial resolution was reduced (Figure 28). In fact, $VE_{0.25}$[1] had zones that represents noise. In contrast, the Phantom 4 Pro point cloud only presented gap voxels for spatial resolutions less or equal than 0.5 m (Figure 29).

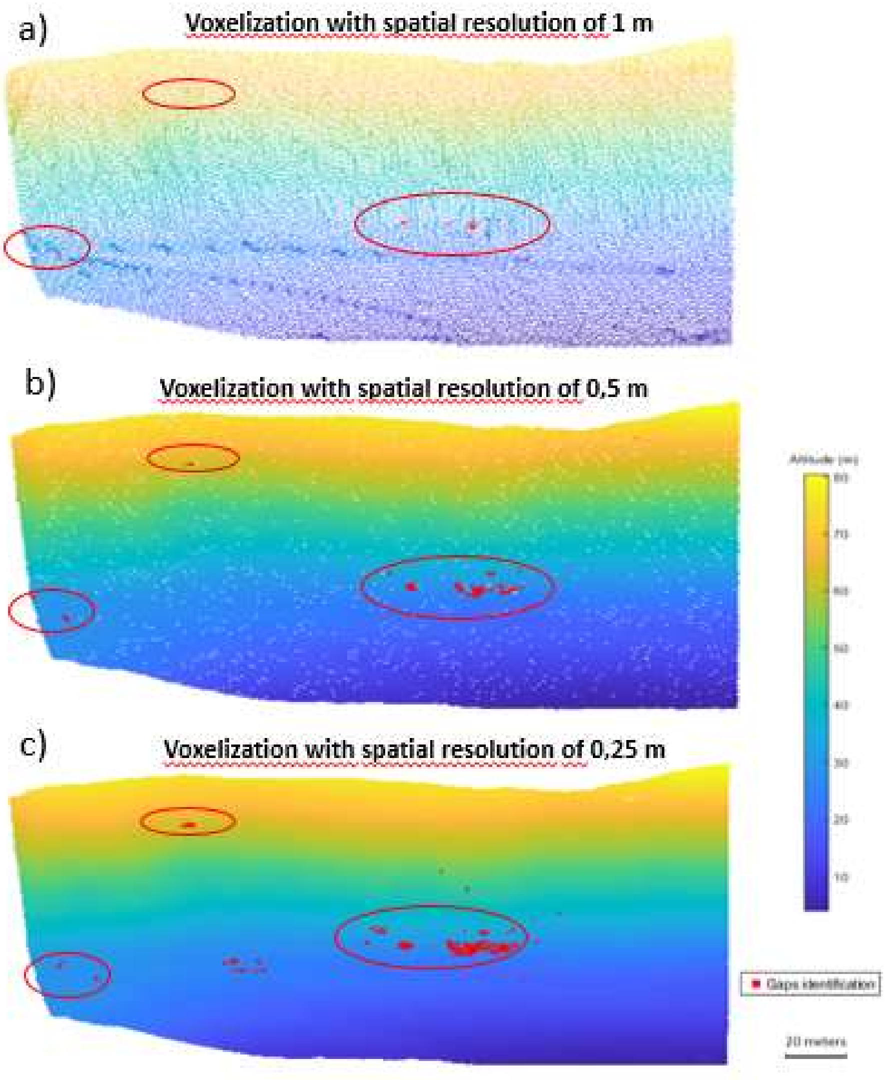

a)
Voxelization with spatial resolution of 1 m
b)
Voxelization with spatial resolution of 0,5 m
c)
Voxelization with spatial resolution of 0,25 m
Altitude (m)
80
70
60
50
40
30
20
10
Gaps identification
20 meters

Figure 28. Gap detection on the Ebee Sensefly dense cloud considering a spatial resolution: a) 1 m; b) 0.5 m; c) 0.25 m

Table 8 summarizes the statistics of the gap voxels estimations. The total of gaps were higher in Ebee data recording values between 50-61 m^3. In the opposite direction, the Phantom data presented volumes between 0-2 m^3.

[1] Notation to a gap voxels set :
VE_i volume of gap voxels for Ebee dense cloud using a spatial resolution of i meters
VPi volume of gap voxels for Phantom dense cloud using a spatial resolution of i meters

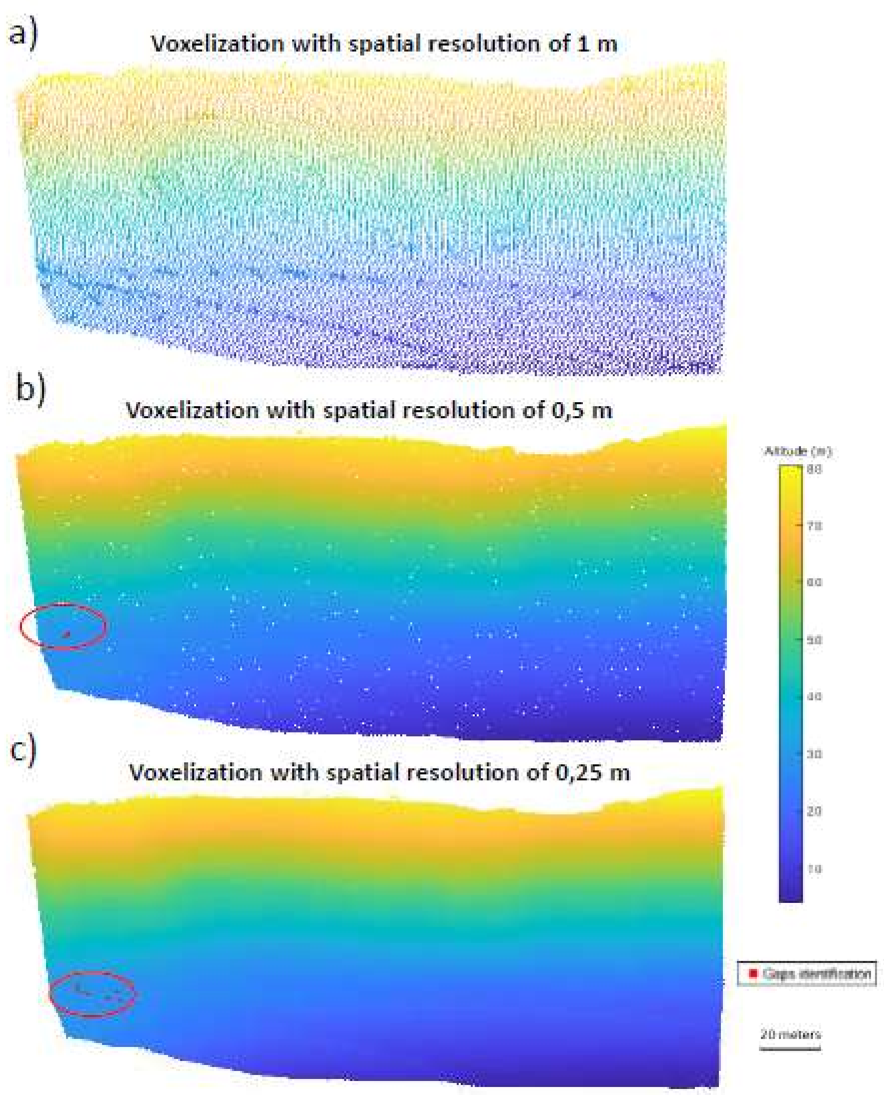

a)
Voxelization with spatial resolution of 1 m
b)
Voxelization with spatial resolution of 0,5 m
c)
Voxelization with spatial resolution of 0,25 m
Altitude (m)
80
70
60
50
40
30
20
10
Gaps identification
20 meters

Figure 29. Gap detection on the Phantom 4 Pro dense cloud considering a spatial resolution: a) 1 m; b) 0.5 m; c) 0.25 m

Table 8. Statistics of the gaps detections with three voxels size (1m, 0.5 m and 0.25 m)

UAS	*Volume per voxel (m^3)*	*Total of volume gaps (m^3)*
Ebee Sensefly	1	50
	0.125	56.75
	0.0156	60.74
Phantom 4 Pro	1	0
	0.125	1.75
	0.0156	1.94

Quantifying zones without data represent an important tool to measure the quality of a point cloud. No method to detect and quantify gaps zones were reported in the literature that supported this study. The algorithm developed estimated voxels that represent gap zones. Experimental tests performed on point clouds allowed the identification of gaps. As expected, the total of gaps were higher in the dense cloud of Ebee Sensefly data. Comparing Figure 26 and Figure 28, most of gap zones were identified and filled with gap voxels. The proportion of gap voxels is clearly linked to the voxel dimension (spatial resolution) as well as the dense cloud density. To assess the effect of dense cloud density on the gap detection, the nearest neighbour distance of all dense cloud points was calculated (Figure 30). As expected, the points are much closer in Phantom data than in Ebee data because of the high difference between the dense cloud points number (about 17 millions). The gap detection depends on the density of points, the number of images and the image geometry (Figures Figure 2 and Figure 13). Therefore, the images dataset obtained Phantom 4 Pro will provide a better 3D reconstruction of the cliff surface given the absence of gaps.

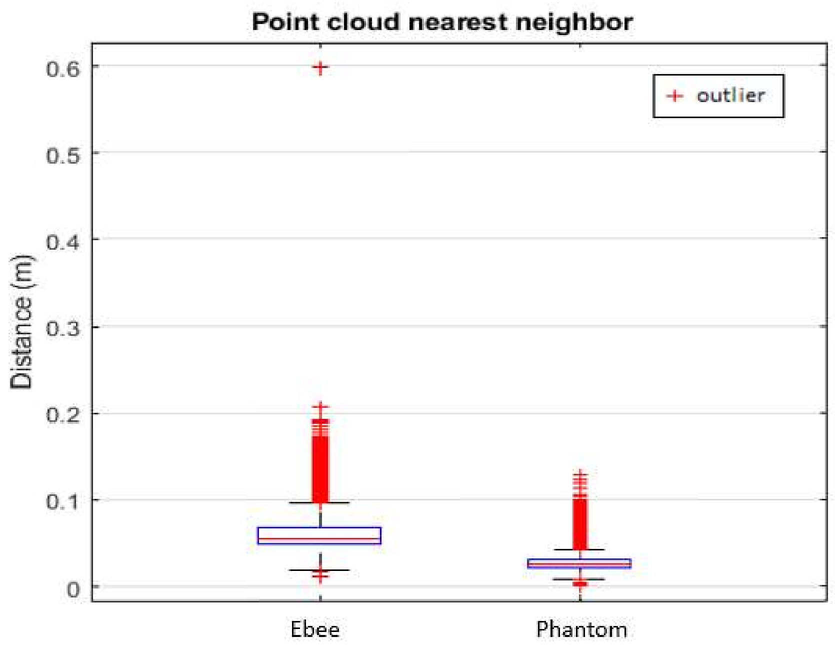


Figure 30. The distance of the point nearest neighbor for the Ebee Sensefly and Phantom 4 Pro dense clouds

4.7 Limitations of the density metrics and Voxelization method

The application of the two density algorithms provide limitations due to the neighbors shapes (section 4.5.2). In CC density, the neighbor shape is dependent on the points number and location. In the voxelization method, the position of voxels grid (neighbors shape) started from the lower limit of all points (x_{min}, y_{min} and z_{min}) to the upper limit (x_{max}, y_{max} and z_{max}). Of course, the choice of the grid position is arbitrary and independent of the points number and location. In the synthetic example, a variation on the density was observed considering two different grid position (see APPENDIX C.1). Optimizing the grid position may minimize the density calculation decreasing the number of empty voxels. This optimization could be done by projecting the points into the planes $X = 0$, $Y = 0$ and $Z = 0$ and calculate contours of points set in each projection. Therefore, a buffer is created from the initial boundary box (parallelepiped) minimizing the search field of voxelization method.

5 Conclusions

The present thesis aimed to make a comprehensive analysis of the processing parameters for the 3D reconstruction of a coastal cliff as well as the automatic identification of areas without data. The data obtained from two UASs had different characteristics, mainly due to the imaging geometry and image numbers collected by the two aircraft, a fixed-wing (Ebee Sensefly) and a multirotor (Phantom 4 Pro). Before the application of an optimized Metashape workflow, a reduction in the number of images of the multirotor was made in order to approximate the number of images of both sets and to evaluate the functionality of "Reduce Overlap" tool. The results showed expressive advantages in terms of processing time. The non-occurrence of more gap zones and the non-variability of RE were also advantages of this platform.

The SfM-MVS processing was done exclusively in Agisoft Metashape in which various combinations of processing parameters included in the BBA were studied. For the Ebee image set, the optimal values to be used were to set a limit on key points at 30000 and tie points at 5000. Concerning the Phantom 4 Pro, the default parameters seem to be suitable, that is, key points limit at 40000 and tie points limit as 4000. The difference between the enable and disable "Adaptive camera model fitting" is not straightforward. However, the impact on the GCPs error is not significant. The impact of the weights associated to the GCPs and tie points only showed to be significant when the weights of the GCPs in the images are overrated (confidence with which the operator identifies the GCPs in the images). After filtering the scattered clouds, it was found that most of the tie points on Ebee are visible on 3 or 2 images (71.5%) while on Phantom they are only visible in 38.4%. In both cases, the reprojection error of these points is around 0.1 pixels.

Derived from occlusions and limitations of the fixed-wing, the point cloud showed zones without data. Through morphological analysis, a method based on voxelization was developed. Three zones without data were identified volumetrically with a total of 50 m^3, 56.75 m^3 and 60.74 m^3 considering the voxel resolution of 1 m, 0.5 m and 0.25 m, respectively.

In general, the point cloud generated from Phantom 4 Pro dataset showed a higher density, which was somewhat expected due to the high number of images compared to the Ebee image dataset.

Future studies will explore the use of triangulations, like 3D Delaunay triangulation or Alpha Shapes, to estimate the gap zones. The voxelization simplify data while a triangulation works directly with the points. The impact of the grid position in voxelization will also be explored (APPENDIX C). Experimental tests showed limitations in voxel visualization. The output was made by points representing the voxel centroids which maybe not intuitive. In addition, the MATLAB point clouds visualization seems not to be optimized when dealing with a large number of points. The methodology implemented used two programming languages (Python and MATLAB) that slows processes and not very practical (e.g. constant change between language interpreters). A recent Python package named Open 3D (Q.-Y. Zhou et al., 2018) has implemented a large number of methods for 3D point clouds. The transition from MATLAB code to Python code will compact workflow and optimize the time in processes.

Finally, to study the impact of reducing the image overlap on the 3D reconstruction of coastal cliffs, a through comparison analysis of the generated point clouds will be made.

References

Acharya, R. (2014). Understanding Satellite Navigation. In *Understanding Satellite Navigation*. https://doi.org/10.1016/C2013-0-06964-2

Agisoft. (2020). *Agisoft PhotoScan User Manual - Professional Edition, Version 1.6*. https://www.agisoft.com/pdf/metashape-pro_1_6_en.pdf, consulted on January 24, 2020

Ahmadabadian, A. H., Robson, S., Boehm, J., Shortis, M., Wenzel, K., & Fritsch, D. (2013). A comparison of dense matching algorithms for scaled surface reconstruction using stereo camera rigs. *ISPRS Journal of Photogrammetry and Remote Sensing*, *78*, 157–167. https://doi.org/10.1016/j.isprsjprs.2013.01.015

Ai, M., Hu, Q., Li, J., Wang, M., Yuan, H., & Wang, S. (2015). A robust photogrammetric processing method of low-altitude UAV images. *Remote Sensing*, *7*(3), 2302–2333. https://doi.org/10.3390/rs70302302

Alsadik, B., Gerke, M., & Vosselman, G. (2014). Visibility analysis of point cloud in close range photogrammetry. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *2*(5), 9–16. https://doi.org/10.5194/isprsannals-II-5-9-2014

Casella, V., Chiabrando, F., Franzini, M., & Manzino, A. M. (2020). Accuracy assessment of a UAV block by different software packages, processing schemes and validation strategies. *ISPRS International Journal of Geo-Information*, *9*(3). https://doi.org/10.3390/ijgi9030164

Esposito, G., Salvini, R., Matano, F., Sacchi, M., Danzi, M., Somma, R., & Troise, C. (2017). Multitemporal monitoring of a coastal landslide through SfM-derived point cloud comparison. *Photogrammetric Record*, *32*(160), 459–479. https://doi.org/10.1111/phor.12218

Fraser, C. S. (1997). Digital camera self-calibration. *ISPRS Journal of Photogrammetry and Remote Sensing*, *52*(4), 149–159. https://doi.org/10.1016/S0924-2716(97)00005-1

Gómez-Gutiérrez, Á., & Gonçalves, G. R. (2020). Surveying coastal cliffs using two UAV platforms (multi-rotor and fixed- wing) and three different approaches for the estimation of volumetric changes. *International Journal of Remote Sensing*, *41*(21), 8143–8175. https://doi.org/10.1080/01431161.2020.1752950

Gonçalves, J. A., & Henriques, R. (2015). UAV photogrammetry for topographic monitoring of coastal areas. *ISPRS Journal of Photogrammetry and Remote Sensing*, *104*, 101–111. https://doi.org/10.1016/j.isprsjprs.2015.02.009

Griffiths, D., & Burningham, H. (2019). Comparison of pre- and self-calibrated camera calibration models for UAS-derived nadir imagery for a SfM application. *Progress in Physical Geography*, *43*(2), 215–235. https://doi.org/10.1177/0309133318788964

Jaud, M., Letortu, P., Théry, C., Grandjean, P., Costa, S., Maquaire, O., Davidson, R., & Le Dantec, N. (2019). UAV survey of a coastal cliff face – Selection of the best imaging angle. *Measurement: Journal of the International Measurement Confederation*, *139*, 10–20. https://doi.org/10.1016/j.measurement.2019.02.024

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, *60*(2), 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94

Manfreda, S., Dvorak, P., Mullerova, J., Herban, S., Vuono, P., Arranz Justel, J., & Perks, M. (2019). Assessing the Accuracy of Digital Surface Models Derived from Optical Imagery Acquired with Unmanned Aerial Systems. *Drones*, *3*(15). https://doi.org/10.3390/drones3010015

Maurer, C. R., Qi, R., & Raghavan, V. (2003). A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *25*(2), 265–270. https://doi.org/10.1109/TPAMI.2003.1177156

Mayer, C., Kersten, T. P., & Pereira, L. G. (2018). A Comprehensive Workflow to Process UAV Images for the Efficient Production of Accurate Geo-information. *IX Conferência Nacional de Cartografia e Geodesia*.

Nguyen, H. M., Wünsche, B., Delmas, P., & Lutteroth, C. (2012). 3D Models from the black box: Investigating the current state of image-based modeling. *20th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, WSCG 2012 - Conference Proceedings*, *PART 2*, 249–258.

Rangel, J. M. G., Gonçalves, G. R., & Pérez, J. A. (2018). The impact of number and spatial distribution of GCPs on the positional accuracy of geospatial products derived from low-cost UASs. *International Journal of Remote Sensing*, *39*(21), 7154–7171. https://doi.org/10.1080/01431161.2018.1515508

Remondino, F., Spera, M. G., Nocerino, E., & Nex, F. (2013). Dense image matching: comparisons and analyses. *2013 Digital Heritage International Congress (DigitalHeritage)*, *1*, 47–54.

Rossi, P., Mancini, F., Dubbini, M., Mazzone, F., & Capra, A. (2017). Combining nadir and oblique uav imagery to reconstruct quarry topography: Methodology and feasibility analysis. *European Journal of Remote Sensing*, *50*(1), 211–221. https://doi.org/10.1080/22797254.2017.1313097

Rupnik, E., Nex, F., & Remondino, F. (2014). Oblique multi-camera systems-orientation and dense matching issues. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, *40*(3W1), 107–114. https://doi.org/10.5194/isprsarchives-XL-3-W1-107-2014

Ružić, I., Marović, I., Benac, Č., & Ilić, S. (2014). Coastal cliff geometry derived from structure-from-motion photogrammetry at Stara Baška, Krk Island, Croatia. *Geo-Marine Letters*, *34*(6). https://doi.org/10.1007/s00367-014-0380-4

Smith, M. W., Carrivick, J. L., & Quincey, D. J. (2015). Structure from motion photogrammetry in physical geography. *Progress in Physical Geography*, *40*(2), 247–275. https://doi.org/10.1177/0309133315615805

Soille, P. (2003). Morphological Image Analysis: Principles and Applications. In *Morphological Image Analysis: Principles and Applications* (pp. 233–234). Springer.

Taddia, Y., Stecchi, F., & Pellegrinelli, A. (2020). Coastal Mapping Using DJI Phantom

4 RTK in Post-Processing Kinematic Mode. *Drones*, *4*(2), 9. https://doi.org/10.3390/drones4020009

Tmuši, G., Manfreda, S., Aasen, H., James, M. R., Gonçalves, G. R., Ben-dor, E., Brook, A., Polinova, M., Arranz, J. J., Mészáros, J., Zhuang, R., Johansen, K., Malbeteau, Y., Pedroso de Lima, I., Davids, C., Herban, S., & McCabe, M. F. (2020). Current Practices in UAS-based Environmental Monitoring. *Remote Sensing*, *12(6)*(1001).

Tonini, M., & Abellan, A. (2014). Rockfall detection from terrestrial lidar point clouds: A clustering approach using R. *Journal of Spatial Information Science*, *8*(1), 95–110. https://doi.org/10.5311/JOSIS.2014.8.123

Trajkovski, K. K., Grigillo, D., & Petrovič, D. (2020). Optimization of UAV flight missions in steep terrain. *Remote Sensing*, *12*(8), 1–20. https://doi.org/10.3390/RS12081293

Warrick, J. A., Ritchie, A. C., Adelman, G., Adelman, K., & Limber, P. W. (2017). New Techniques to Measure Cliff Change from Historical Oblique Aerial Photographs and Structure-from-Motion Photogrammetry. *Journal of Coastal Research*, *33*(1), 39. https://doi.org/10.2112/jcoastres-d-16-00095.1

Westoby, M. J., Brasington, J., Glasser, N. F., Hambrey, M. J., & Reynolds, J. M. (2012). "Structure-from-Motion" photogrammetry: A low-cost, effective tool for geoscience applications. *Geomorphology*, *179*, 300–314. https://doi.org/10.1016/j.geomorph.2012.08.021

Westoby, M. J., Lim, M., Hogg, M., Pound, M. J., Dunlop, L., & Woodward, J. (2018). Cost-effective erosion monitoring of coastal cliffs. *Coastal Engineering*, *138*, 152–164. https://doi.org/10.1016/j.coastaleng.2018.04.008

Xiao, J., Gerke, M., & Vosselman, G. (2012). Building extraction from oblique airborne imagery based on robust façade detection. *ISPRS Journal of Photogrammetry and Remote Sensing*, *68*(1), 56–68. https://doi.org/10.1016/j.isprsjprs.2011.12.006

Young, A. P., & Carilli, J. E. (2019). Global distribution of coastal cliffs. *Earth Surface Processes and Landforms*, *44*(6), 1309–1316. https://doi.org/10.1002/esp.4574

Zhou, F., Cui, Y., Peng, B., & Wang, Y. (2012). A novel optimization method of camera parameters used for vision measurement. *Optics and Laser Technology*, *44*(6), 1840–1849. https://doi.org/10.1016/j.optlastec.2012.01.023

Zhou, Q.-Y., Park, J., & Koltun, V. (2018). Open3D: A Modern Library for 3D Data Processing. *ArXiv:1801.09847*. http://arxiv.org/abs/1801.09847

A. MATLAB codes

Code in MATLAB R2020a to implement the voxelization method

```matlab
function [vd,density,centroids,fillc,loc] = voxelization(pc,sr)
    % voxelization create a voxelization from a point cloud pc with
    % spatial resolution sr. The point cloud have N points.
    %
    % vd = voxelization(pc,sr) return a Nx1 vector with the density
    % of each point.
    %
    % [vd,density,centroids,fillc,loc] = voxelization(pc,sr) return
    % also a MxKxD matrix that each matrix element corresponds to a
    % voxel element, the centroids of all voxels and the centroids
    % of the fill centroids and the voxel that the points belong
    %
    %Functions:
    % histcn n-d histogram count [1]
    %
    %References:
    %[1] Bruno Luong (2020). N-dimensional histogram
    %   (https://www.mathworks.com/matlabcentral/fileexchange/23897-
n-dimensional-histogram),
    %   MATLAB Central File Exchange. Retrieved June 6, 2020.

    %Create a 3D grid
    xmin = double(pc.XLimits(1));xmax = double(pc.XLimits(2));
    ymin = double(pc.YLimits(1));ymax = double(pc.YLimits(2));
    zmin = double(pc.ZLimits(1));zmax = double(pc.ZLimits(2));
    xe=xmin:sr:xmax+sr;
    ye=ymin:sr:ymax+sr;
    ze=zmin:sr:zmax+sr;
    %2. Count points in each voxel
    %histcn calculate a n Dimensional histogram.
    [Voxels,~,mid,loc] = histcn(pc.Location,xe, ye,ze);
    vol_per_voxel = sr^3;
    density = Voxels./vol_per_voxel;

    %Linear index of the points. It can be looked as the id of the
    %voxels
    L = sub2ind(size(Voxels),loc(:,1),loc(:,2),loc(:,3));

    %Assigning the density of points (point i receives the density
    % of the voxel where it is inserted)
    vd = zeros(length(L),1);
    for i=1:length(L)
        vd(i) = density(L(i));
    end
    xm = mid{1,1};ym=mid{1,2};zm=mid{1,3};
    centroids = combvec(xm,ym,zm)';
    cB = reshape(Voxels,[],1);
    fillc = centroids(cB>0,1:3);
end
```

Code for implementing the fallDetection method

```matlab
function fall = fallDetection(mask,buffer)
% fallDetection estime the missing values of a 3D binary image.
%
% fall = fallDetection(mask,buffer) apply the estimation excluding
% the x,y and z in along the boundaries. Note that buffer is a 1x3
% integer array representing the voxels across each direction.
%
% See also BWDIST, IMREGIONALMIN

    s = size(mask);
    fall = false(s(1),s(2),s(3));
    bx = buffer(1);by = buffer(2);bz = buffer(3);
    for k=1+bz:s(3)-bz
        fall(1+bx:end-bx,1+by:end-by,k) = ...
             mergePath(mask(1+bx:end-bx,1+by:end-by,k)) - ...
             mask(1+bx:end-bx,1+by:end-by,k);
    end
end

%Auxiliary functions
function out = mergePath(mask)
% mergePath create pixels to join all the groups in a 2D binary
% image.
%
% out = mergePath(mask) return a 2D binary image with only a group
% considering a 8-connectivity.

    out = mask;
    cond = true;
    while cond
        L = bwlabel(out,8);
        globalMax = max(L,[],'all');
        %Test if there is one path
        if globalMax ==1
            cond = false;
        else
            L1 = L==1;L2 = L == 2;
            D1 = bwdist(L1, 'quasi-euclidean');
            D2 = bwdist(L2, 'quasi-euclidean');
            %Cost surface calculation
            D_sum = D1 + D2;
            path_pixels = imregionalmin(D_sum,8);
            out = out | path_pixels;
        end
    end
end
```

B. Python codes

Set of Python (version 3.7.4) functions to infer analyzes to a workflow in Metashape API (version 1.6.1).

```
#GLOBAL VARIABLE
epsg_code = "4326"
geoid_path = r"D:\Diogo\GeodPT08.tif"
reduce_path = r"D:\Diogo\UArribaS\PCalada2018\Voo"
coordsys='''
    COMPD_CS["ETRS89 / Portugal TM06 / GeodPT08",
    PROJCS["ETRS89 / Portugal TM06",GEOGCS["ETRS89",
    DATUM["European Terrestrial Reference System 1989",
    SPHEROID["GRS 1980",6378137,298.257222101,
    AUTHORITY["EPSG","7019"]],
    TOWGS84[0,0,0,0,0,0,0],
    AUTHORITY["EPSG","6258"]],
    PRIMEM["Greenwich",0,
    AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.01745329251994328,
    AUTHORITY["EPSG","9102"]],AUTHORITY["EPSG","4258"]],
    PROJECTION["Transverse_Mercator",AUTHORITY["EPSG","9807"]],
    PARAMETER["latitude_of_origin",39.66825833333333],
    PARAMETER["central_meridian",-8.133108333333331],
    PARAMETER["scale_factor",1],
    PARAMETER["false_easting",0],PARAMETER["false_northing",0],
    UNIT["metre",1,AUTHORITY["EPSG","9001"]],
    AUTHORITY["EPSG","3763"]],VERT_CS["GeodPT08",
    VERT_DATUM["GeodPT08",2005],UNIT["metre",1,AUTHORITY["EPSG","9001"]]]]
    '''
#Matching accuracy
accuracy = {
    "highest": 0,
    "high": 1,
    "medium": 2,
    "low": 4,
    "lowest": 8
}
#---------------------------------------------------------------------------
def files_list(base_path,extension='.psx'):
    """
    Input:
        - base_path(string) -> base path for the project(s)
        - extension(string) -> format for the files to group
    Output:
        - files(list) -> list of string with the pahts of the files found
    """
    import os

    files = []
    for r, d, f in os.walk(base_path):
        for file in f:
            if extension in file:
                files.append(os.path.join(r, file))
    return files

def reference_settings(chunk,settings):
    """
    Input:
        - chunk (Metashape Obj)
        - settings(dict) -> change the refence settings with a specific key
    Output:
        - name (type) -> explanation
    Note: keys: 'ca_m','ca_p' camera accuracy in meters and pixels;
                'ma_m','ma_p' marker accuracy in meters and pixels;
                'tpa' tie points accuracy
    """
    from Metashape import Vector

    chunk.camera_location_accuracy = Vector(
        (settings['ca_m'],settings['ca_m'],settings['ca_m'])) #vector meter
    chunk.camera_rotation_accuracy = Vector(
        (settings['ca_p'],settings['ca_p'],settings['ca_p']))#vector degree
    chunk.marker_location_accuracy = Vector(
        (settings['ma_m'],settings['ma_m'],settings['ma_m']))#vector meter
    chunk.marker_projection_accuracy = settings['ma_p']#float pixels
    chunk.tiepoint_accuracy = settings['tpa']#float pixels

def add_coordsys_geoid(chunk):
    """
    Add a geoid to the Metashape database
    Input:
        - chunk (Metashape Obj)
    """
    import Metashape

    chunk.crs.addGeoid(geoid_path)
    chunk.crs = Metashape.CoordinateSystem(coordsys)
    chunk.updateTransform()

def change_pixel_size(chunk,psize,index=0):
    """
    Input:
        - chunk (Metashape Obj)
        - psize (float) -> pixel size of a given sensor
        - index (int) -> index of the sensor
    """
    import Metashape
    chunk.sensors[index].pixel_size = Metashape.Vector([psize,psize])

def save_project_plus_images(doc,f_path,img_path):
    """
    Input:
        - doc(Metashape document Obj)
        - f_path(string) -> path for the file location
        - img_path (string) -> path for the images location
    Output:
        - doc(Metashape document Obj)
    Note:
    """
    import Metashape

    #1. Create project
    doc.save(f_path)
    doc.addChunk()
    chunk = doc.chunk
    #2. Read images
    files = files_list(img_path,extension='.JPG')
    chunk.addPhotos(
        files,
        load_xmp_accuracy=True
    )
    #3. Change camera coordenate system
    cameras_crs = Metashape.CoordinateSystem("EPSG::"+epsg_code)
    chunk.camera_crs = cameras_crs
    doc.save()
    return doc

def read_markers_xml(path,PATH_REDUCE=None):
    """
    Input:
        - path (string): path for xml file created by Metashape
        - PATH_REDUCE(string) -> path for the reduced images set
    Output:
        - CAMERAS (dict) -> {id of camera: (name of camera, id of actual
camera)}
        - M_img_coords (dict) -> {id of marker:{id of camera: image
coordenate
        system}}
    Note: a marker can have more than one camera
    """
    from xml.dom import minidom
    import os

    tree = minidom.parse(path)
    if PATH_REDUCE is not None:
        all_cameras = []
        for r, d, f in os.walk(reduce_path):
            for file in f:
                if '.JPG' in file:
                    all_cameras.append(file.split('.')[0][-4:])
        ids = list(range(len(all_cameras)))
        reduce_images = []
        for r, d, f in os.walk(PATH_REDUCE):
            for file in f:
                if '.JPG' in file:
                    reduce_images.append(file.split('.')[0][-4:])
        CAMERAS = {}
        actual_id = 0
        for l in ids:
            if all_cameras[l] in reduce_images:
                CAMERAS[str(l)]=(all_cameras[l],str(actual_id))
                actual_id+=1
    #1. Get id and name from all cameras
    else:
        images = tree.getElementsByTagName('camera')
        print('Nice')
        CAMERAS = {}
        for img in images:
            key = img.attributes['id'].value
            temp = img.attributes['label'].value
            im_aux = temp.split('.')[0][4:]
            CAMERAS[key] = (im_aux,key)
    #2. Get image coordinates from markers in all images
    itens = tree.getElementsByTagName('markers')
    markers = itens[1].getElementsByTagName('marker')
    M_img_coords = {}
    for marker in markers:
        key = marker.attributes['marker_id'].value
        cameras = marker.getElementsByTagName('location')
        if len(cameras)>0:
            temp = {}
            for camera in cameras:
                cam_key = camera.attributes['camera_id'].value
                if cam_key in list(CAMERAS.keys()):
                    cam_coord = (
                        float(camera.attributes['x'].value),
                        float(camera.attributes['y'].value))
                    cam_actual_key = CAMERAS[cam_key][1]
                    temp[cam_actual_key] = cam_coord
            M_img_coords[key] = temp
    return CAMERAS, M_img_coords

def upload_img_coord(chunk,M):
    """
    Input:
        - chunk (Metashape Obj)
        - M(dict) -> dictionary parameterized like the output
(M_img_coords) of
        function read_markers_xml
    """
    import Metashape

    for marker in chunk.markers:
        k = str(marker.key)
        if k in M.keys():
            for camera in chunk.cameras:
                c = str(camera.key)
                if c in M[k].keys():
                    vector = Metashape.Vector(M[k][c])
                    marker.projections[camera]=Metashape.Marker.Projection(
                        vector,True)

def change_marker_function(chunk,marker_list,function):
    """
    Input:
        - chunk(Metashape chunk Obj)
        - marker_list(list) -> markers for change function
        - function(bool) -> True(control point), False(check point)
    """
    for marker in chunk.markers:
        #print("key: {}".format(marker.key))
        #print("label: "+marker.label)
        if marker.key in marker_list:
            marker.reference.enabled = function
    chunk.updateTransform()
    print("Points moved to check points: "+str(marker_list))

def remove_marker(chunk,marker_list):
    """
    Input:
        - chunk (Metashape chunk Obj)
        - marker_list (list) -> markers for change function
    """
    for marker in chunk.markers:
        if marker.key in marker_list:
            chunk.remove(marker)
    chunk.updateTransform()
    print("Marker(s) removed: "+str(marker_list))

def add_altitude_to_image(chunk,alt=100):
    """
    Input:
        - chunk(Metashape chunk Obj)
        - alt -> altitude to add to cameras
    Source: adapted from https://github.com/agisoft-llc/metashape-scripts
    """
    import Metashape
    for camera in chunk.cameras:
        if camera.reference.location:
            coord = camera.reference.location
            camera.reference.location = Metashape.Vector(
                [coord.x, coord.y, coord.z + alt]
            )
    chunk.updateTransform()
    Metashape.app.update()

def rmse_calculation_per_marker_pix(chunk,flag=True):
    """
    RMSE in pixels
    Input:
        - chunk(Metashape chunk Obj)
        - flag(bool) -> True if it is GCPs; False if it is CHPs
    Output:
        - name (type) -> indformation displayed id_marker  RMSE
        and estimated coordinates
    """
    from numpy import sqrt
    markers_rmse = {}
    for marker in chunk.markers:
        if marker.reference.enabled == flag:
            key = str(marker.key)
            sm = 0
            n = 0
            nproj = marker.projections.keys()
            if len(nproj)>0:
                for camera in marker.projections.keys():
                    v_proj = marker.projections[camera].coord
                    v_reproj = camera.project(marker.position)
                    diff = (v_proj - v_reproj).norm2()
                    sm += diff
                    n +=1
                markers_rmse[key] = (
                    marker.label, len(nproj), round(sqrt(sm/n),3))
    return markers_rmse

def rmse_calculation_per_marker_meter(chunk,flag=True):
    """
    RMSE in meters
    Input:
        - chunk(Metashape chunk Obj)
        - flag(bool) -> True if it is GCPs; False if it is CHPs
    Output:
        - name (type) -> indformation displayed id_marker  RMSE
        and estimated coordinates
    """
    markers_rmse = {}
    for marker in chunk.markers:
        if marker.reference.enabled == flag:
            key = str(marker.key)
            source = marker.reference.location
            estim = chunk.crs.project(
                chunk.transform.matrix.mulp(marker.position))
            error = (estim - source).norm()
            markers_rmse[key] = (
                marker.label,
                1,
                round(error,4),
                (
                    round(estim.x,4),
                    round(estim.y,4),
                    round(estim.z,4)
                )
            )
    return markers_rmse

def rmse_aux(dic):
    """
    Input:
        - name (type) -> explanation
    Output:
        - name (type) -> explanation
    Note:
    """
    from numpy import sqrt
    if not dic:
        return None
    else:
        weights = 0
        aux_error = 0
        ks = list(dic.keys())
        for el in ks:
            aux_error += dic[el][1]*dic[el][2]**2
            weights += dic[el][1]
        return sqrt(aux_error/weights)

def rmse_calculation(chunk):
    """
    Input:
        - chunk (Metashape chunk Obj)
    Output:
        - dic(dict) -> information about RMSE on the GCPs and CHPs
    Note:
    """
    #Error from control points (ctp)
    d_p = rmse_calculation_per_marker_pix(chunk)
    er_pixel_ctp = rmse_aux(d_p)
    if er_pixel_ctp == None:
        d_ctp = "There are no check points!"
    else:
        d_m = rmse_calculation_per_marker_meter(chunk)
        er_meter_ctp = rmse_aux(d_m)
        d_ctp = {}
        for k in d_p.keys():
            d_ctp[k] = (
                d_p[k][0],
                {
                    'pixel': (d_p[k][1],d_p[k][2]),
                    'meter': (d_m[k][1],d_m[k][2])
                },
                d_m[k][3]
            )
        d_ctp['all'] = {
            'pixel': round(er_pixel_ctp,3),
            'meter': round(er_meter_ctp,4)
        }
    #Error from check points (ckp)
    d_p = rmse_calculation_per_marker_pix(chunk,flag=False)
    er_pixel_ckp = rmse_aux(d_p)
    if er_pixel_ckp == None:
        d_ckp = "There are no check points!"
    else:
        d_m = rmse_calculation_per_marker_meter(chunk,flag=False)
        er_meter_ckp = rmse_aux(d_m)
        d_ckp = {}
        for k in d_p.keys():
            d_ckp[k] = (
                d_p[k][0],
                {
                    'pixel': (d_p[k][1],d_p[k][2]),
                    'meter': (d_m[k][1],d_m[k][2])
                },
                d_m[k][3]
            )
        d_ckp['all'] = {
            'pixel': round(er_pixel_ckp,3),
            'meter':round(er_meter_ckp,4)
        }
    dic = {}
    dic['ctp'] = d_ctp
    dic['ckp'] = d_ckp
    return dic

def export_stats(file_path, data,rms=None):
    """
    Create a txt file with the project statistics like RMSE
    Input:
        - name (type) -> explanation
        - data(dict) -> dictionary organized like the output of
rmse_calculation
    """
    f_in = open(file_path,'w')
    for ks in data.keys():
        if isinstance(data[ks],dict):
            if ks == 'ctp':
                f_in.write("# CONTROL POINTS\n")
                f_in.write("#   |       EstimatedCoordinates       |\n")
                f_in.write("# ID| %-13s  %-13s  %-13s| %-7s | %-7s\n" % (
                    'X(m)',
                    'Y(m)',
                    'Z(m)',
                    'ERROR(m)',
                    'ERROR(p)')
                )
            else:
                f_in.write("\n# CHECK POINTS")
                f_in.write("#   |         Estimated Coordinates        |\n")
                f_in.write("# ID| %-13s  %-13s  %-13s| %-7s | %-7s\n" % (
                    'X(m)',
                    'Y(m)',
                    'Z(m)',
                    'ERROR(m)',
                    'ERROR(p)')
                )
            e = data[ks].pop('all')
            for k in data[ks].keys():
                f_in.write("%4s| %-13.4f  %-13.4f  %-13.4f| %-7.4f | %-
7.4f\n" % (
                    data[ks][k][0],
                    data[ks][k][2][0],
                    data[ks][k][2][1],
                    data[ks][k][2][2],
                    data[ks][k][1]['meter'][1],
                    data[ks][k][1]['pixel'][1])
                )
            f_in.write("# Total error: {} m  {} p\n".format(
                 e['meter'],e['pixel']))
        else:
            f_in.write("# {}\n".format(data[ks]))
    f_in.write("# RMS Reprojection Error: {} pix".format(rms))
    f_in.close()
    print("File created and saved on "+file_path)

def calc_reprojection(chunk):
    """
    Input:
        - chunk (Metashape chunk Obj)
    Output:
        - rms (type) -> explanation
        - point_errors (dict) -> grouped by tie point and each tie point id
        are the error of that point in the correspond image
        - maxe
    Note:
    Source: https://www.agisoft.com/forum/index.php?topic=11548
    """
    from numpy import sqrt

    point_cloud = chunk.point_cloud
    points = point_cloud.points
    npoints = len(points)
    projections = chunk.point_cloud.projections
    err_sum = 0
    num = 0
    maxe = 0
    point_ids = [-1] * len(point_cloud.tracks)
    point_errors = dict()
    for point_id in range(0, npoints):
        point_ids[points[point_id].track_id] = point_id
    #track id of each point id
    for camera in chunk.cameras:
        if not camera.transform:
            continue
        for proj in projections[camera]:
            track_id = proj.track_id
            point_id = point_ids[track_id]
            if point_id < 0:
                continue
            point = points[point_id]
            if not point.valid:
                continue
            error = camera.error(point.coord, proj.coord).norm2()
            err_sum += error
            num += 1
            if point_id not in point_errors.keys():
                point_errors[point_id] = [camera.key,error]
                point_errors[point_id] = [error]
            else:
                point_errors[point_id].append(camera.key)
                point_errors[point_id].append(error)
            if error > maxe: maxe = error
    rms = sqrt(err_sum / num)
    return rms, point_errors, maxe

def sparse_cloud_filtering(chunk,threshold,c = 'RE'):
    """
    Input:
        - chunk (Metashape chunk Obj)
        - threshold(multiple) -> threshold for the criteria selection
        - c(string) -> criterion for "Gradual Selection
    """
    import Metashape

    criterion = {
        'RE': Metashape.PointCloud.Filter.ReprojectionError,
        'RU': Metashape.PointCloud.Filter.ReconstructionUncertainty,
        'IC': Metashape.PointCloud.Filter.ImageCount,
        'PA': Metashape.PointCloud.Filter.ProjectionAccuracy
    }
    sparse = chunk.point_cloud
    print(len(sparse.points))
    fltr_cld = Metashape.PointCloud.Filter()
    fltr_cld.init(sparse,criterion[c])
    fltr_cld.selectPoints(threshold)
    nselected = len([p for p in chunk.point_cloud.points if p.selected])
    sparse.removeSelectedPoints()
    print("*", nselected, " points removed from filtering")
    Metashape.app.update()
    print(len(sparse.points))
    chunk.optimizeCameras(fit_f=True, fit_cx=True, fit_cy=True,fit_b1=True,
        fit_b2=True, fit_k1=True, fit_k2=True, fit_k3=True,
        fit_k4=False, fit_p1=True, fit_p2=True, fit_corrections=False,
        adaptive_fitting=False, tiepoint_covariance=False)

def create_project(paths,s,reduce=False,loop=False):
    """
    Input:
        - paths(dict) -> paths for the images, markers
coordinates(Metashape
        format and where the project will be saved)
        - s(dict) -> dictionary with the tie points and markers accuracies
        - reduce (bool) -> if we have a case where the original image
        - loop(bool) -> if create_project is invoked externally inside a
          loop
        varying s values
    """
    import Metashape

    base_path = paths['base']
    markers_path = paths['markers']
    xml_path = paths['xml']
    project_name = paths['name']
    if loop:
        project_path =  + "{}\\{}tpa{}ma{}.psx".format(
            base_path,project_name,str(s['tpa']),str(s['ma_p']))
    else:
        project_path = base_path + "\\" + project_name+".psx"
    images_path = paths['images']
    Metashape.app.cpu_enable = False
    mask = 2 ** len(Metashape.app.enumGPUDevices()) - 1
    Metashape.app.gpu_mask = mask
    print("GPU(s) is(are) enable(s).")
    doc = Metashape.Document()
    doc.read_only = False
    print("Creating project ...")
    #Save project and upload images with a given epsg code
    doc = save_project_plus_images(doc,project_path,images_path)
    chunk = doc.chunk
    #Import markers
    add_altitude_to_image(chunk)
    if reduce:
        C,M = read_markers_xml(xml_path,images_path)
    else:
        C,M = read_markers_xml(xml_path)
    chunk.importReference(
        markers_path,
        Metashape.ReferenceFormatCSV,
        delimiter = ",",
        columns = "nxyzXYZ",
        create_markers = True,
        skip_rows = 1
    )
    add_coordsys_geoid(chunk)
    upload_img_coord(chunk,M)
    add_altitude_to_image(chunk)
    reference_settings(chunk,s)
    chunk.importMasks(path=paths['masks']+'\\{filename}_mask.png',
                      source=Metashape.MaskSourceFile,
                      operation=Metashape.MaskOperationReplacement)
    doc.save()
    #FREE PROJECT FROM MEMORY
    doc.clear()
    print("Project created. Turning off !!!")

def open_project(file_path):
    """
    Input:
        - file_path(string) -> path where the project is located
    Output:
        - doc(Metashape document Obj)
    """
    import Metashape

    try:
        Metashape.app.cpu_enable = False
        mask = 2 ** len(Metashape.app.enumGPUDevices()) - 1
        Metashape.app.gpu_mask = mask
        print("GPU(s) is(are) enable(s).")
        doc = Metashape.Document()
        doc.read_only = False
        doc.open(file_path, read_only=False, ignore_lock=True)
        print("Project is open!")
        return doc
    except:
        print("Something rouge! Please check the path or if the project
exists!")

def align(paths, ac="high", tiepoints=5000, keypoints=30000,
adapt_fit=False):
    """
    Input:
        - paths (dict) -> path where is located the project(s)
        - tiepoints (int) -> tie points limit
        - keypoints (int) -> key points limit
        - adapt_fit (bool) -> if it will be used adaptive model fitting
    Output:
        - name (type) -> explanation
    Note: it is a more generic function to align camera than
build_sparseCloud
    """
    import Metashape

    files = files_list(paths['base'])
    for f in files:
        doc = open_project(f)
        print("Aligning images ...")
        chunk = doc.chunk
        add_coordsys_geoid(chunk)
        chunk.matchPhotos(
            downscale = accuracy[ac],
            generic_preselection = True,
            reference_preselection = True,
            filter_mask=True,
            mask_tiepoints=False,
            keypoint_limit = keypoints,
            tiepoint_limit = tiepoints
        )
        chunk.alignCameras(
            adaptive_fitting = adapt_fit
        )
        chunk.updateTransform()
        Metashape.app.update()
        doc.save()
        doc.clear()
        print("Images aligned")

def build_sparseCloud(paths, tiepoints=4000, keypoints=70000,
adapt_fit=True,
    change_pixel=False, reduceoverlap=False):
    """
    Input:
        - paths (dict) -> paths for the images, markers coordinates
          (Metashape format and where the project will be saved)
        - tiepoints (int) -> tie points limit
        - keypoints (int) -> key points limit
        - adapt_fit (bool) -> if it will be used adaptive model fitting
        - change_pixel (bool) -> cases where the pixel size does not
          uploaded
        - reduceoverlap (bool) -> if we have a case where the original
          image
        set is reduced with the "Reduceoverlap Metashape" tool
    Output:
        - name (type) -> explanation
    Note: it is also provided overall statistics and camera calibrations
    files
    """
    import Metashape
    import time

    base_path = paths['base']
    markers_path = paths['markers']
    xml_path = paths['xml']
    project_name = "Ebee_Calada_t"+str(tiepoints)+"_k"+str(keypoints)
    project_path = base_path + "\\" + project_name+".psx"
    images_path = paths['images']
    file_name = "stats_t"+str(tiepoints)+"_k"+str(keypoints)+".txt"
    file_path = base_path + "\\" + file_name
    calibration_name = project_name+"_calibrationP"
    calibration_path = base_path+"\\"+calibration_name+".xml"
    Metashape.app.cpu_enable = False
    mask = 2 ** len(Metashape.app.enumGPUDevices()) - 1
    Metashape.app.gpu_mask = mask
    print("GPU(s) is(are) enable(s).")
    doc = Metashape.Document()
    doc.read_only = False
    try:
        doc.open(project_path, read_only=False, ignore_lock=True)
        print("The project is open!")
        chunk = doc.chunk
        add_coordsys_geoid(chunk)
    except:
        print("Project doesn' t exist. Creating project ...")
        t0 = time.perf_counter()
        #Save project and upload images with a given epsg code
        doc = save_project_plus_images(doc,project_path,images_path)
        chunk = doc.chunk
        #Import markers
        add_altitude_to_image(chunk)
        if reduceoverlap:
            C,M = read_markers_xml(xml_path,images_path)
        else:
            C,M = read_markers_xml(xml_path)
        chunk.importReference(
            markers_path,
            Metashape.ReferenceFormatCSV,
            delimiter = ",",
            columns = "nxyzXYZ",
            create_markers = True,
            skip_rows = 1
        )
        add_coordsys_geoid(chunk)
        upload_img_coord(chunk,M)
        if change_pixel:
            psize = 0.00125 #in milimeters, for a Sony Cyber-shot DSC-WX220
            change_pixel_size(chunk,psize)
        print("Project created and data imported in {} s.".format(
            round(time.perf_counter()-t0,2))
        )
        print("Ready to go!!")
        #ALIGN PHOTOS
        print("Aligning photos ...")
        t0 = time.perf_counter()
        chunk.matchPhotos(
            downscale = accuracy["high"],
            generic_preselection = True,
            reference_preselection = True,
            keypoint_limit = keypoints,
            tiepoint_limit = tiepoints
        )
        chunk.alignCameras(
            adaptive_fitting = adapt_fit
        )
        print("Time spend in aligning: {} s.".format(
            round(time.perf_counter()-t0,2))
        )
        chunk.updateTransform()
        Metashape.app.update()
        doc.save()
    #CALCULATE STATISTICS AND EXPORT INFORMATION
    di1 = rmse_calculation(chunk)
    rms,a1,a2 = calc_reprojection(chunk)
    export_stats(file_path,di1,rms)
    chunk.sensors[0].calibration.save(calibration_path)
    sparse = chunk.point_cloud
    len_sparse=len(sparse.points)
    doc.save()
    #FREE PROJECT FROM MEMORY
    doc.clear()
    print("All good. Turning off !!!")
    return rms, len_sparse

def sparseCloud_assess(paths):
    """
    Assess a sparse cloud focus on the Metashape Image Count and
    Reprojection
    errors.
    Input:
        - paths(dict) -> base path for the project and data
    Note: it is also provide boxplots and barplots
    """
    import Metashape

    base_path = paths['base']
    project_name = paths['name']
    project_path = base_path + "\\" + project_name +".psx"
    doc = open_project(project_path)
    chunk = doc.chunk
    criterion = {
        'RE': Metashape.PointCloud.Filter.ReprojectionError,
        'RU': Metashape.PointCloud.Filter.ReconstructionUncertainty,
        'IC': Metashape.PointCloud.Filter.ImageCount,
        'PA': Metashape.PointCloud.Filter.ProjectionAccuracy
    }
    sparse = chunk.point_cloud
    print("Total of points: {}".format(len(sparse.points)))
    fltr_re = Metashape.PointCloud.Filter()
    fltr_ic = Metashape.PointCloud.Filter()
    fltr_re.init(sparse,criterion['RE'])
    fltr_ic.init(sparse,criterion['IC'])

    import numpy as np
    import matplotlib.pyplot as plt

    RE = np.array(fltr_re.values)
    IC = np.array(fltr_ic.values)
    print("Max Image Count: {}".format(np.amax(IC)))
    print("Min Image Count: {}".format(np.amin(IC)))
    #boolean with the index where the condition is verified
    index_re = RE>0.4
    index_ic = IC<=3
    index_re.astype(np.int)
    index_ic.astype(np.int)
    ic_3 = IC[IC<=3]
    re_3 = RE[IC<=3]
    print("Mean RE in IC <= 3: {}".format(np.mean(re_3)))
    print("Std RE in IC <= 3: {}".format(np.std(re_3)))
    print("Max RE in IC <= 3: {}".format(np.amax(re_3)))
    print("Min RE in IC <= 3: {}".format(np.amin(re_3)))
    print("Median RE in IC <= 3: {}".format(np.median(re_3)))
    print(np.quantile(re_3,[0,0.25,0.5,0.75,1]))
    pointT = len(ic_3)
    #Plot histogram
    bins = list(range(np.amin(IC),np.amax(IC)+1))
    hist, bins = np.histogram(IC, bins=bins)
    width = np.diff(bins)
    center = (bins[:-1] + bins[1:]) / 2
    fig1, ax1 = plt.subplots()
    ax1.set_title("Histogram of Image Count")
    ax1.bar(center, hist, align='center', width=width)
    v = list(range(np.amin(IC),np.amax(IC)+15,15))
    ax1.set_xticks(v)
    ax1.set_xlabel('Tie points image count observation')
    ax1.set_ylabel('Number of observations')
    ax1.yaxis.grid(True)
    #Plot boxplot
    fig1, ax1 = plt.subplots()
    ax1.set_title("BoxPlot of Image Count")
    ax1.boxplot([re_3, RE])
    ax1.set_ylabel('RMS Reprojection error (pix)')
    ax1.set_xticklabels(['3 or less image count', 'Global'])
    ax1.yaxis.grid(True)
    print("number of points with 3 or less images projections: {}".format(
        pointT))
    print("number of points with re greater than 0.4: {}".format(
        len(RE[index_re])))
    #Images projection of the points that satisfy the condition
    ic_ = IC[RE>0.4]
    ic__ = ic_[ic_<=3]
    pointN = len(ic__)
    print("number of points with 3 or less images projections and more than
0.4 re: {}".format(pointN))
    porp = float(pointN)/pointT
    print("The porposion is {}".format(round(porp,4)))
    rms,_,__ = calc_reprojection(chunk)
    print("RMS Reprojection error: {}".format(rms))
    #FREE PROJECT FROM MEMORY
    doc.clear()

def errors(paths):
    """
    Input:
        - paths(dict) -> base path for the existent(s) project(s)
    Output:
        - stats(dict) -> dictionary with information about the GCPs RMSE,
        CHPs
        RMSE and RMS reprojection error
    """
    import os

    files = files_list(paths['base'])
    stats = {}
    for f in files:
        doc = open_project(f)
        print("Project {}".format(os.path.basename(f)))
        chunk = doc.chunk
        add_coordsys_geoid(chunk)
        #CALCULATE STATISTICS AND EXPORT INFORMATION
        rmse = rmse_calculation(chunk)
        rms,a1,a2 = calc_reprojection(chunk)
        stats[os.path.basename(f)] = {
            'ctp': rmse['ctp']['all']['meter'],
            'ckp': rmse['ckp']['all']['meter'],
            'rms': rms
        }
        doc.save()
        doc.clear()
    return stats

def play_with_cp(paths, tiepoints=4000, keypoints=70000,chps=[]):
    """
    Input:
        - paths (dict) -> base path for the project and data
        - tiepoints (int) -> tie points number
        - keypoints (int) -> key points number
        - chps (list) -> identifier to the points that will be change side
    Output:
        - ctp_error (float) -> RMSE on the Ground Control Points
        - ckp_error (float) -> RMSE on the Check Points
    """
    import Metashape

    base_path = paths['base']
    project_name = "Ebee_Calada_t"+str(tiepoints)+"_k"+str(keypoints)
    project_path = base_path + "\\" + project_name+".psx"
    file_name = "stats_{}GCPs_t{}_k{}.txt".format(
            len(chps),tiepoints,keypoints
    )
    file_path = base_path + "\\" + file_name
    doc = open_project(project_path)
    chunk = doc.chunks[0]
    add_coordsys_geoid(chunk)
    new_chunk = chunk.copy()
    LABEL = "Tie Points with {} CHPs({})".format(
        len(chps),str([x+1 for x in chps])
    )
    new_chunk.label = LABEL
    Metashape.app.update()
    doc.save()
    print("Chunk duplicate and project saved.")
    #MOVE MARKERS TO CHECK POINTS
    change_marker_function(new_chunk,chps,False)
    print("Change done.")
    #CALCULATE STATISTICS AND EXPORT INFORMATION
    di1 = rmse_calculation(new_chunk)
    ctp_error = di1['ctp']['all']['meter']
    ckp_error = di1['ckp']['all']['meter']
    rms,a1,a2 = calc_reprojection(new_chunk)
    export_stats(file_path,di1,rms)
    Metashape.app.update()
    doc.save()
    #FREE PROJECT FROM MEMORY
    doc.clear()
    return ctp_error,ckp_error

def filterSC(paths):
    """
    Input:
        - paths (dict) -> base path for the existent(s) project(s)
    Output:
        - data (list) -> list with information of the filtering organized
        [project_name  sparse_cloud_point_number  RMSreprojection_error]
    """
    import Metashape
    from os.path import basename

    files = files_list(paths['base'])
    data = []
    for p in files:
        a = basename(p)
        doc = open_project(p)
        chunk = doc.chunk
        add_coordsys_geoid(chunk)
        sparse_cloud_filtering(chunk,0.4)
        chunk.updateTransform()
        Metashape.app.update()
        rms,a1,a2 = calc_reprojection(chunk)
        sparse = chunk.point_cloud
        len_sparse=len(sparse.points)
        data.append([a,len_sparse,rms])
        doc.save()
        #FREE PROJECT FROM MEMORY
        doc.clear()
    return data

def get_RMS(paths):
    """
    Input:
        - paths (string) -> base path for the existent(s) project(s)
    Output:
        - data (list) -> list with information organized [
        project_name  sparse_cloud_point_number  RMSreprojection_error]
    """
    from os.path import basename

    files = files_list(paths)
    data = []
    print('Process started!')
    for p in files:
        a = basename(p)
        doc = open_project(p)
        chunk = doc.chunk
        add_coordsys_geoid(chunk)
        #Statistics
        rms,a1,a2 = calc_reprojection(chunk)
        sparse = chunk.point_cloud
        len_sparse=len(sparse.points)
        data.append([a,len_sparse,rms])
        doc.clear()
    return data
```

C. Volumetric density

C.1 Two-dimensional unmatched illustration

Counting points within a range changes the final density changing only the number of bins on the grid. Figure C1 shows different densities derived from the grid position. If the number of points within the square containing the red point is calculated, the density changes from 2 to 3 points per m^2. When the square is fixed, the number of neighbors of the green points is different from the square count number (three points with 3 neighbors and one with just one neighbor against one square with 4 points in Figure C1c and three squares with 3, 1 and 1 points in Figure C1d respectively). In summary, the voxelization density is dependent on the grid position.

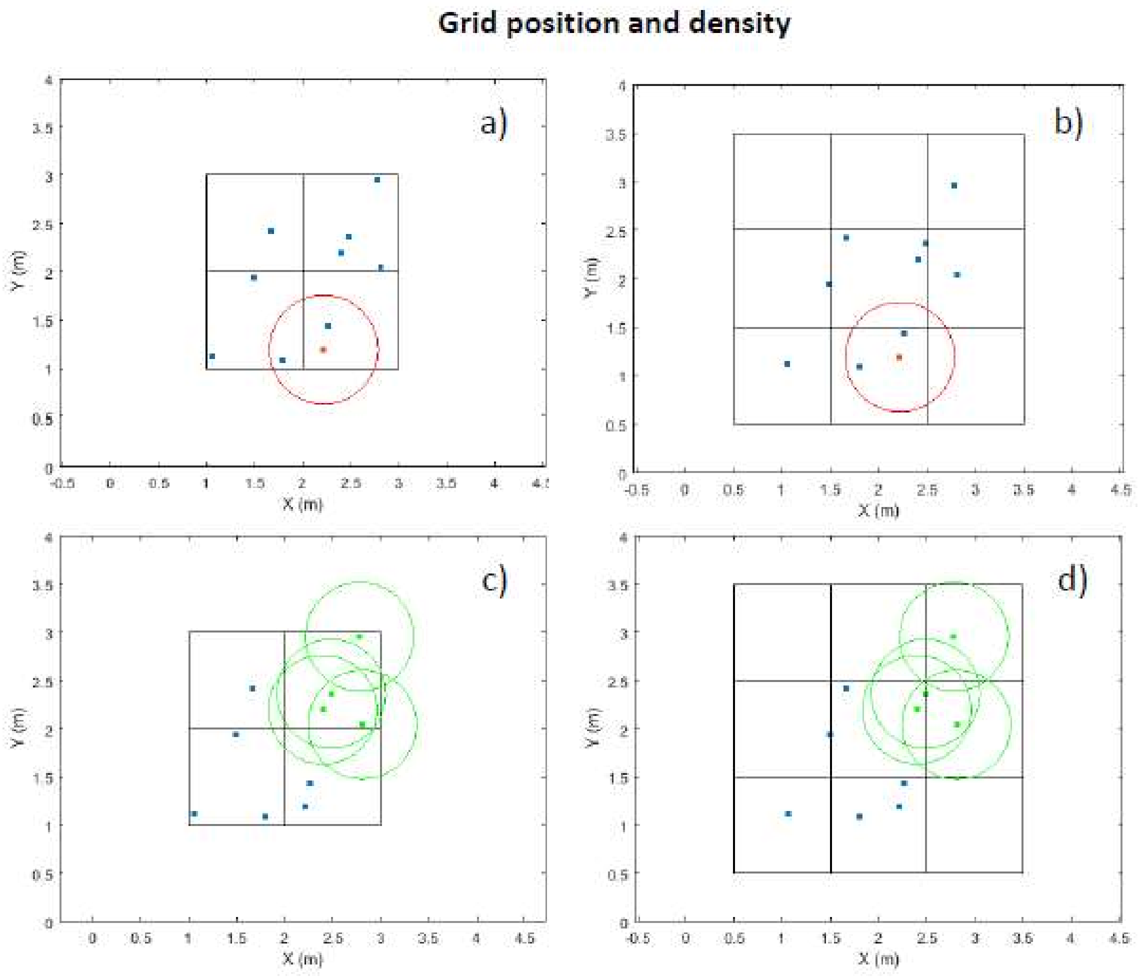


Figure C1.Two grid position with the same resolution for same CC neighbours circle

C.2 Replicate CloudCompare volumetric density

To understand the CloudCompare algorithm for volumetric density calculation, a replication was made in MATLAB R2020a using “Live Script” mode. The idea was to generate a synthetic point cloud and compute three volumetric densities (Figure C2): 1. volumetric density from voxelization method (APPENDIX A); 2. volumetric density from CC replication method; 3. real volumetric density from CC. Therefore, a qualification with RMSE will be made to evaluate the reliability of reconstructed method. Lastly, a comparison between densities of 1. and 2. will be assess the possibility of differences.

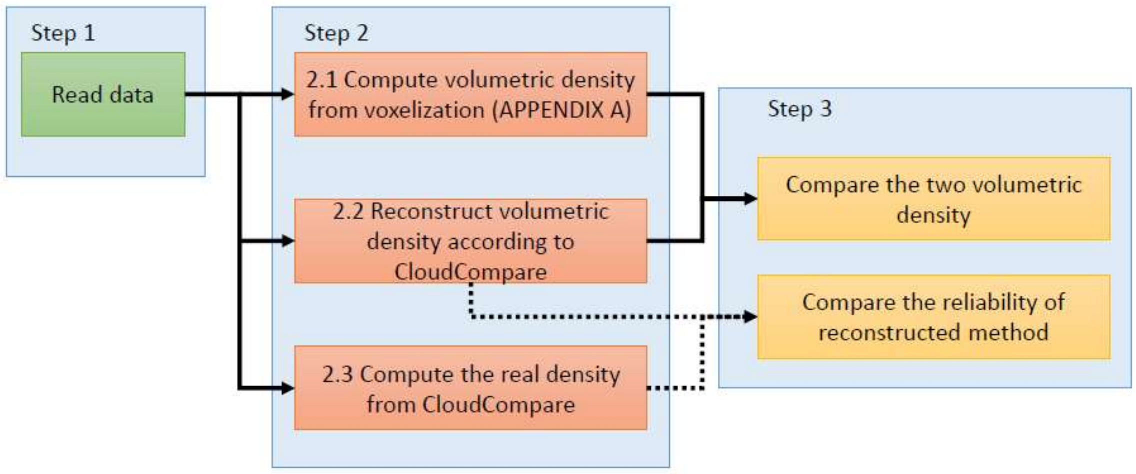

Step 1
Read data
Step 2
2.1 Compute volumetric density from voxelization (APPENDIX A)
2.2 Reconstruct volumetric density according to CloudCompare
2.3 Compute the real density from CloudCompare
Step 3
Compare the two volumetric density
Compare the reliability of reconstructed method

Figure C2. Flowchart to compare CloudCompare volumetric density and volumetric density

Step 1: Read data

Before reading data, a random point cloud was created with 50 points and saved with the pcd (Point Cloud Data) format. Therefore, the point cloud has been read and duplicate to calculate the three densities proposed in Step 2.

```
pc = pcread("SyntheticPC.pcd");
pc_CCreplicate = pc;
pc_voxelization = pc;
```

Step 2: Compute densities from voxelization method and CloudCompare

In this step, the aims is to compute two different volumetric densities (voxelization method and CC). Besides that the volumetric density of CC will be replicated to understand better the algorithm.

2.1 Compute volumetric density from voxelization

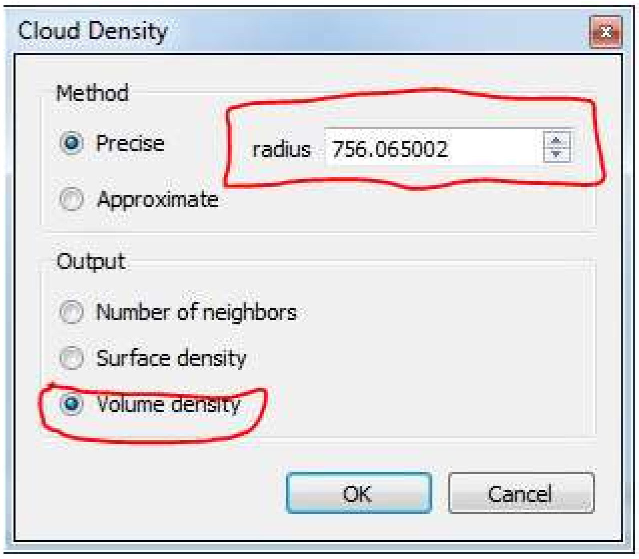
The computation of voxelization is straightforward and is given by voxelization method (APPENDIX A). The density are calculated dividing the number of points each voxel by the single voxel volume. *vd* is a 50x1 vector with density of each points.

```
srV = 1;
[vd,~] = voxelization(pc_voxelization,srV);
pc_voxelization.Intensity = vd;%Only for visualization
```

2.2 Reconstruct volumetric density according to CloudCompare

The input "radius" is the radius of the neighbor sphere, let us say, N_i is the number of neighbors of the point i (Figure C3). To search neighbors per m^3, we need to calculate the correspondent radius of a sphere with 1 m^3. So,

$$V = \frac{4}{3}\pi r^3 \Leftrightarrow r = \sqrt[3]{\frac{3V}{4\pi}}$$



Method
[✓ Precise] Approximate
radius 756.065002

Output
Number of neighbors
Surface density
[✓ Volume density]

Figure C3. CloudCompare interface to calculate the "Volume density". Note that red boxes represent the method and variables in study

```
np = pc_CCreplicate.Count;
r = ((3*1)/(4*pi))^(1/3);
N = zeros(np,1);
for i=1:np
    [index,~] = findNeighborsInRadius(...
        pc_CCreplicate,pc_CCreplicate.Location(i,:),r);
    N(i) = numel(index);
end
```

The neighbors per volume, from a point i, are given by the formula

$$S_i = \frac{N_i}{V}$$

```
V = (4/3)*pi*r^3;
S = N./V;
pc_CCreplicate.Intensity = single(S);
```

2.3 Compute the real density from CloudCompare

In CC, the density was calculated with the same radius of 2.2. As the point cloud was exported as an ASCII file, the last 3 columns represent the "Volume density", "Surface density" and "Number of neighbors", respectively. In this case, it will be used only the “Volume density” column.

```
data = readmatrix("CCDensity.txt");
CCData = data(:,end-2);
diff = CCData-S;
rmse = sqrt(mean(diff.^2));
```

The result of RMSE between the replicated and real CC volumetric density was 0 m.

Step 3: Comparison between methods of Step 2

The output point clouds from Step 2 was parameterized to be visualized with *pcshow* function. After that, the comparison between densities methods will be presented

Visualize results from Step 2

The results of Step 2 were associated to each point cloud and visualize as an intensity field (

Figure C4).

```
figure;pcshow(pc_CCreplicate,"MarkerSize",200)%Figure C4 a
colormap('parula')
cb = colorbar()
cb.Label.String = 'Volumetric density (pts/m3)';
set(gcf,'color','w');set(gca,'color','w');
xlabel('M (meters)');ylabel('P (meters)');zlabel('Z (meters)');

pc.Intensity = single(CCData); %Figure C4 b
figure;pcshow(pc,"MarkerSize",200)
colormap('parula')
cb = colorbar();
cb.Label.String = 'Volumetric density (pts/m3)';
set(gcf,'color','w');set(gca,'color','w');
xlabel('M (meters)');ylabel('P (meters)');zlabel('Z (meters)')

figure;pcshow(pc_voxelization,"MarkerSize",200)%Figure C4 c
colormap('parula')
cb = colorbar();
cb.Label.String = 'Volumetric density (pts/m3)';
set(gcf,'color','w');set(gca,'color','w');
xlabel('M (meters)');ylabel('P (meters)');zlabel('Z (meters)');
```

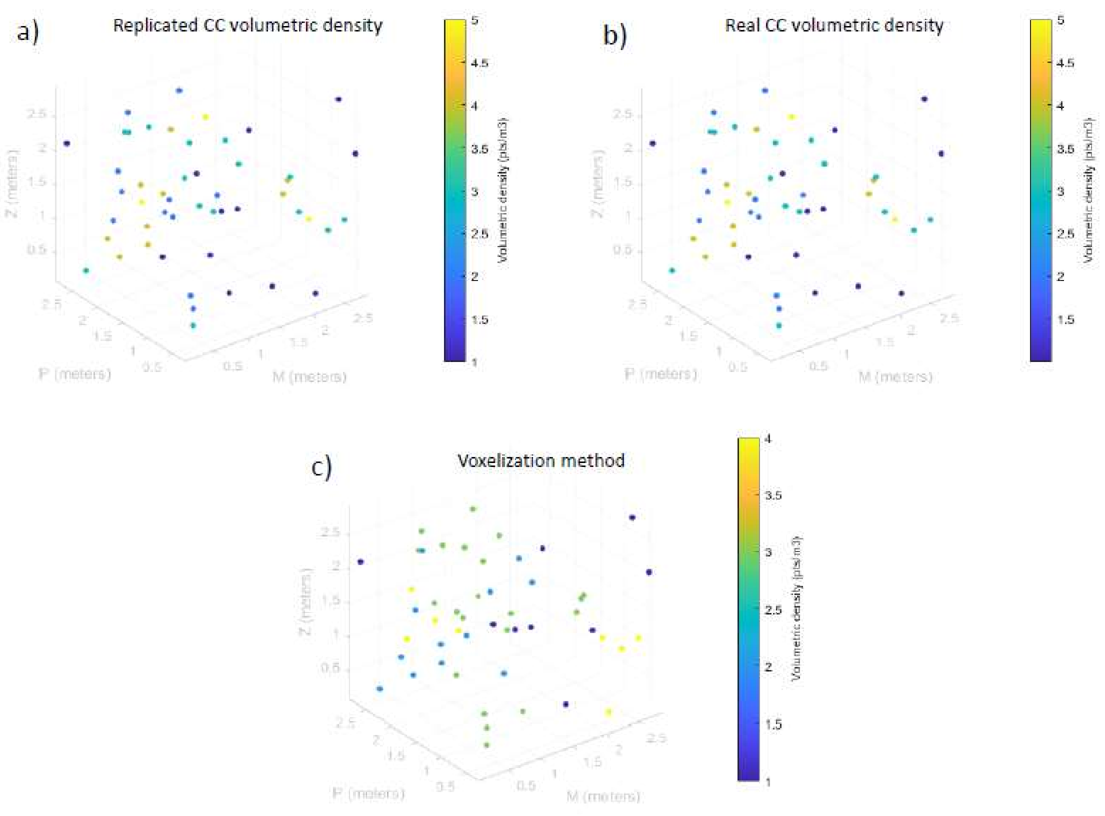


Figure C4. Volumetric density calculated: a) from the reconstructed CC volumetric density; b) from the real CC volumetric density; c) from voxelization method

Visually, Figure C4a-b are equals which justify the RMSE being 0 and the reliability of the CC replicated method. Looking at Figure C4c, the density is now the same as in Figure C4a or b which leads to a lower density values in voxelization method.

Comparison between methods 2.1 and 2.2

As said before, the density results from CC and voxelization methods was considerable different. To highlight these differences a histogram was created coupling the densities of steps 2.1 and 2.2 (Figure C5). Comparing bins values, there is same differences resulting from the chosen algorithm. Points inside a voxel have the same density while points inside a sphere may have different density (see Figure C1c)

```
figure;histogram(S,1:1:6)
hold on
histogram(vd,1:1:6)
ax = gca;
ax.YGrid = 'on';
```

```
xlabel('Density values')
ylabel('Number of observations')
legend('CloudCompare','Voxelization')
```

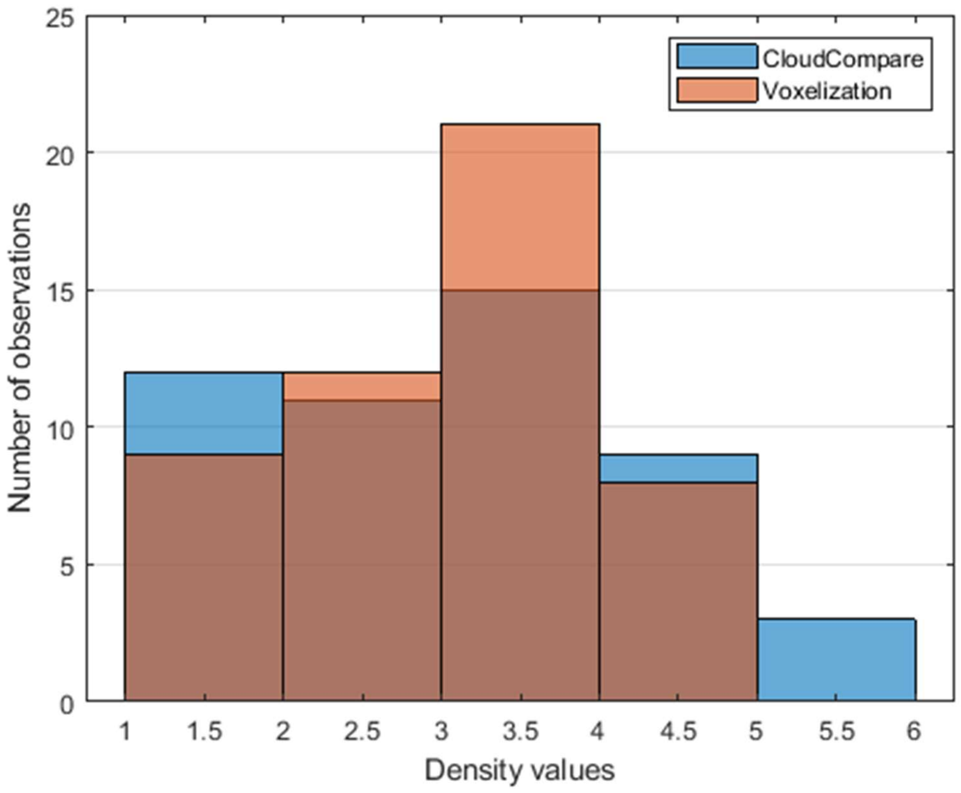


Figure C5. Histogram count for the volumetric density with equal bins spacing for CloudCompare and voxelization method