Improving Open-Source Photogrammetric Workflows for Processing Big Datasets

Report: Tie-points reduction tools in MicMac. Description and experiments

O. Martinez-Rubi
Netherlands eScience Center,
Science Park 140 (Matrix 1), 1098 XG Amsterdam, the Netherlands

October 18, 2016

Contents

Abstract

Photogrammetric techniques are used for the generation of point cloud data. Multiple images acquired with photo cameras are processed into dense point clouds using photogrammetric workflows. These are normally based on a basic photogrammetric workflow with three steps: tie-points detection; estimation of camera positions and orientations and of calibration parameters; and dense-matching point cloud generation. When processing large image sets, the execution of the workflows is time-consuming and can become limiting due to the hardware requirements of the processing. This report presents two algorithms that decrease the memory requirements and the processing time of the step that performs the estimation of camera positions and orientations and of calibration parameters. The algorithms are implemented as Free and Open-Source Software (FOSS) tools within the MicMac photogrammetry suite. They are tested with three different datasets to assess the effect on the memory usage as well as on the quality of the generated point clouds. The algorithms presented can be combined with other existing solutions that target at speeding up the processing. Moreover, they allow running photogrammetric workflows in hardware systems that could not be used before. This work has been done as part of the eScience project "Improving Open-Source Photogrammetric Workflows for Processing Big Datasets".

1 Introduction

In the geo-information domain, a point cloud is a three-dimensional (3D) representation of a surface. To obtain point clouds, several techniques are available, the most popular of which are Lidar techniques. These are based on measuring distances using laser lights, but Lidar scanners tend to be rather expensive. A cheaper alternative to Lidar are photogrammetric techniques, since the only mandatory device is a photo camera. These techniques are sometimes referred to as Structure from Motion (SfM) techniques. They became popular through their use in small areas, objects or structures ([26]), and they have recently come to be used for larger areas, especially thanks to the increased usage of Unmanned Aerial Vehicles (UAVs) ([25, 22]). Many studies are available in which Lidar and photogrammetric techniques are compared ([3, 6, 12, 5]).

1.1 Photogrammetric workflow

A photogrammetric workflow is a process where multiple images acquired from different viewing angles are combined and processed into a dense point cloud. Photogrammetric workflows have been used by many researchers ([7, 13, 20]) and some Free and Open Source Software (FOSS) implementations are available ([2, 11, 29, 8, 24]). Each implementation is different, and workflows can become complicated with additional steps to improve the results or to speed up the process. However, most of them are built on the same basic workflow with three steps as depicted in Figure 1. Next, we will provide a brief description of the steps:

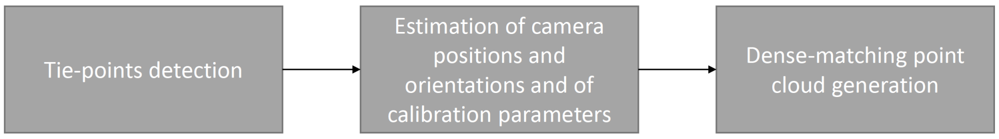

Tie-points detection
Estimation of camera positions and orientations and of calibration parameters
Dense-matching point cloud generation

Figure 1: Steps of the basic photogrammetry workflow.

- **Tie-points detection**. First, key features in the images are extracted. This is done for example with the SIFT algorithm ([19]). Second, the key features are cross-matched between different images to detect points in the images that represent the same physical locations and that are visible in different images. The detected points are called tie-points (they are also referred to as homologous points in related literature).

- **Estimation of camera positions and orientations and of calibration parameters**. Several parameters are estimated: parameters related to the position and orientation of the camera from where the pictures were acquired; and calibration parameters, i.e. properties of the image sensor of the camera used such as lens distortion. The estimation of the parameters may contain some preparation or initialization steps, but the main part of the processing consists of the so-called bundle adjustment: all the tie-points from all the images are loaded in the memory of the hardware system, and the parameters are refined in an iterative process.

- **Dense-matching point cloud generation**. The pixels of the images are projected and intersected in 3D to produce the dense point cloud.

1.2 Motivation and goal

The general trend in the last years is to increase the number of images as well as to increase their resolution. The increase in resolution also causes that for each image thousands of key features can be detected.

Executing photogrammetric workflows is time consuming and can become limiting when there is a high number of images. In some cases, additional data such as Geotagging (GPS) data or Ground Control Points (GCPs) are collected to aid the processing. Nevertheless, this increases the cost of the acquisition campaign, and sometimes it is unfeasible to collect the additional data. For example, it is impossible to use GCPs in areas which are inaccessible.

From a computational perspective, parallelization techniques can be applied to speed up the execution of the workflow when dealing with large image sets. On the one hand, the steps performing the tie-points detection and the dense-matching point cloud generation are relatively easy to parallelize, i.e. the processing can be split in chunks that are independent and can run in parallel (the processing of a chunk does not require data from another chunk). For example, the detection of key features with the SIFT algorithm for an image is independent from the rest of images. Therefore, these steps can be easily speed up by using hardware systems with multiple nodes and cores. On the other hand, the bundle adjustment, which performs the estimation of the camera positions and orientations and of the calibration parameters, is not easily parallelizable. The parameters estimation requires all the tie-points from all the images. This demands a high available memory, and also makes it difficult to split the processing in independent chunks that could benefit from hardware systems with multiple nodes and cores. Therefore, other solutions must be explored.

There are some elegant, efficient and complete solutions specially tailored to deal with large image sets ([9, 1, 17]). Other studies target at speeding up the bundle adjustment ([23, 27, 18, 15, 4, 30, 16, 21]). We recommend to the reader the work by [16] which includes an inventory of existing techniques and algorithms for speeding up the bundle adjustment.

Even though distributed computing solutions for the parallelization of the tie-points detection and the dense-matching point cloud generation have also been explored during the project, this report targets at the limitations in the estimation of the camera positions and orientations and of the calibration parameters. We are preparing a paper where the focus is on the distributed solutions. This reports focuses on the bundle adjustment limitations, and it tackles them from a different perspective. We do not aim at modifying the bundle adjustment itself. Instead, we propose a tie-points reduction step that runs before the bundle adjustment. The execution of the tie-points reduction step decreases the memory requirements of the bundle adjustment and also speeds it up. Our solution can be combined with other solutions previously developed.

Our work is motivated by the hypothesis that when orienting images, which is done in the bundle adjustment, a high number of tie-points is not really needed. A small number of tie-points is required as long as the selected tie-points have certain properties: they are properly distributed in the pixel-space of the images, and are present in many images.

The tie-points reduction step proposed smartly decreases the number of tie-points. We propose two different algorithms to perform the task. The algorithms are implemented as stand-alone tools within the MicMac photogrammetry suite ([24]). The idea of reducing tie-points has already been introduced in a study conducted by [21]. However, in that case the tie-points reduction was part of the bundle adjustment. Our solution runs before the bundle adjustment, after the tie-points detection. In this way, the memory requirements of the bundle adjustment are decreased in the beginning of its execution.

By using the tie-points reduction step proposed, we also enable the execution of photogrammetric workflows for large image sets in hardware systems that could not be used before. This is specially useful in situations when there is not access to hardware systems with high memory and/or with multiple nodes and cores.

1.3 Report outline

The remainder of the report is organized as follows: Section 2 gives an overview on MicMac. Section 3 describes the tie-points reduction algorithms. Section 4 contains information regarding a set of experiments that assess the quality and the effect of the developed algorithms. Finally, Section 5 contains the conclusions and the future work.

2 MicMac overview

MicMac is an Free and Open-Source Software (FOSS) suite that executes photogrammetric workflows. It contains highly configurable tools that can be combined to create different photogrammetric workflows tailored for different situations. When creating a photogrammetric workflow, several characteristics must be taken into account: the acquisition method used (UAV, aerial, terrestrial, etc.), the availability of auxiliary data (Geotagging, GCPs, etc.), the size of the image set, etc..

The execution of various workflows with MicMac is described in greater detail and with the help of examples given by [10]. For the complete set of tools and options of MicMac we refer the reader to the MicMac user guide [1]. Next, we will provide a brief description on the tools that are normally used in photogrammetric workflows with MicMac. The tools are identified within the basic photogrammetric workflow as described in Subsection 1.1. After the images have been acquired, the steps to generate a dense point cloud with MicMac usually include:

[1]To access the user guide, download MicMac using *Mercurial* from *https://geoportail.forge.ign.fr/hg/culture3d*; the guide is in the Documentation folder.

- **Tie-points detection** with the *Tapioca* tool. Note that by definition, a tie-point is a point that represents the same physical location in two or more images. Hence, a tie-point can be linked to image pairs, image triplets or higher order image sets. However, even when a tie-point is present in more than two images, *Tapioca* only provides the tie-point by image pairs. Obviously, not all the possible image pairs share tie-points, the image pairs that share tie-points are called valid image pairs. For each valid image pair, a file is created containing the list of tie-points shared by the image pair. For each tie-point *Tapioca* stores four values, the tie-point X,Y positions in the two images. In some cases, information about which are the valid image pairs can be provided to *Tapioca* in order to speed up the processing. For example, this information can be derived from Geotagging data (if this was collected with the images).
- **Estimation of camera positions and orientations and of calibration parameters**. Before the bundle adjustment, various initialization steps may be required depending on the situation. For large image sets, an initial estimation of the calibration parameters may be provided using only a small subset of the images. Additionally, an initialization of the position and orientation parameters is recommended and can be performed with the *Martini* tool. The bundle adjustment is run with the *Tapas* tool.
- **Dense-matching point cloud generation** with the *Malt*, *Tawny* and *Nuage2Ply* tools.

2.1 pymicmac

Pymicmac (https://github.com/ImproPhoto/pymicmac) is a python interface to MicMac. It allows to define through an XML file photogrammetric workflows to be executed with MicMac. A sequence of MicMac commands is specified in the XML file. During the execution of each command pymicmac also monitors the usage of CPU, memory and disk. Pymicmac also contains tools for the distributed execution of some MicMac commands, namely of the tie-point detection tool *Tapioca* and of the dense-matching tools *Malt*, *Tawny* and *Nuage2Ply*.

3 Tie-points reduction

We propose to run a tie-points reduction step before the bundle adjustment. We have added the stand-alone tools *RedTieP* and *OriRedTieP* in the MicMac suite. Each of these tools implement a different algorithm for the tie-points reduction. By running *RedTieP* or *OriRedTieP* before *Tapas*, the memory requirements and the processing time of the latter are decreased. In both cases, the reduced set of tie-points are stored in the same format as provided by *Tapioca*, i.e. by valid image pairs. Thus, adding the tie-points reduction step proposed to an existing MicMac photogrammetric workflow is straight-forward.

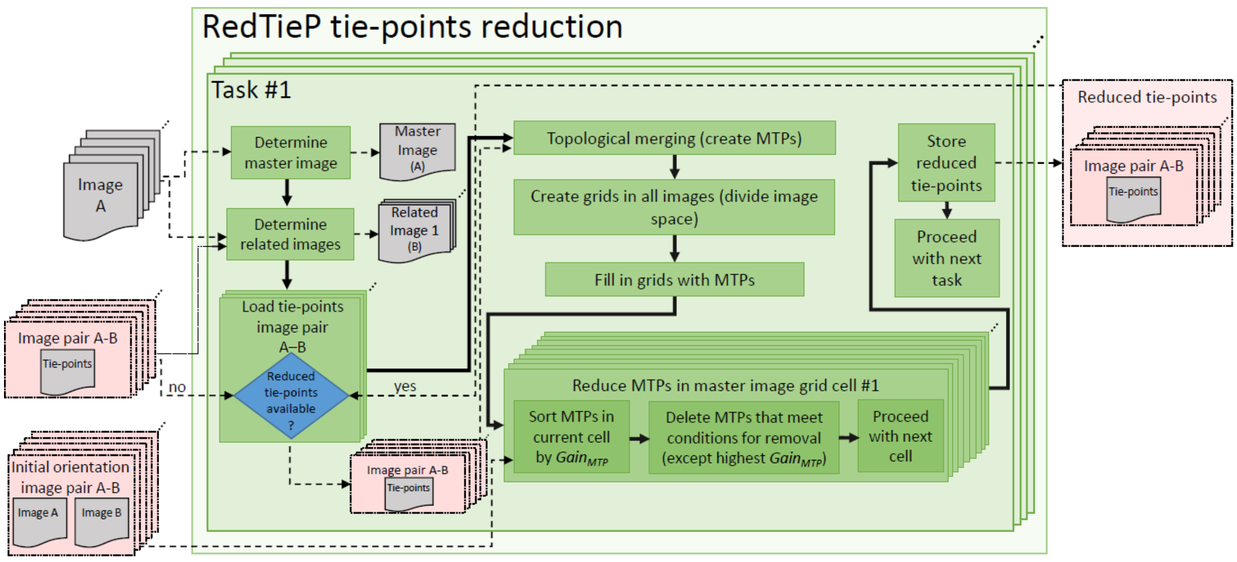
In the next subsections, we provide detailed information regarding the implemented algorithms in both tools. In Section 4 these are tested and compared.

3.1 *RedTieP*

The first option for the tie-points reduction step is to use *RedTieP*. Before its execution, this tool requires an estimation of the relative orientation between the valid image pairs. This can be done with an auxiliary tool called *NO_AllOri2Im*.

3.1.1 Steps

Figure 2 depicts an overview of the steps performed by *RedTieP*.



RedTieP tie-points reduction
Task #1
Image A
Determine master image
Master Image (A)
Determine related images
Related Image 1 (B)
Load tie-points image pair A–B
Reduced tie-points available ?
no
yes
Image pair A-B
Tie-points
Initial orientation image pair A-B
Image A
Image B
Topological merging (create MTPs)
Create grids in all images (divide image space)
Fill in grids with MTPs
Reduce MTPs in master image grid cell #1
Sort MTPs in current cell by $Gain_{MTP}$
Delete MTPs that meet conditions for removal (except highest $Gain_{MTP}$)
Proceed with next cell
Store reduced tie-points
Proceed with next task
Reduced tie-points

Figure 2: Overview of the steps performed by *RedTieP*.

The tie-points reduction is divided into tasks that are sequentially executed. There are as many tasks as images, and in each task various steps are performed. As an illustrative example we use the first task with Figures 3 to 6 explaining the steps in a graphical manner. Next, we will describe the steps performed by all the tasks. In each task, the algorithm performs:

1. **Determines the master image**. This is the image driving the tie-points reduction in this task. The *ith* task uses as master image the *ith* image from the list of images.

2. **Determines the related images**. These are the images that share tie-points with the master image. In other words, there is a valid image pair between the master image and each of the related images. To find these image the algorithm looks at the files generated by *Tapioca*, which only outputs files for valid image pairs.

3. **Loads (only) the tie-points of valid image pairs composed by the master image and each of the related images**. Note that the tie-points of valid image pairs composed only by related images are not loaded. When loading the tie-points of a valid image pair, if the related image in this valid image pair was a master image in a previously executed task, the algorithm uses the list of reduced tie-points produced in the earlier executed task instead of the original list of tie-points (given by *Tapioca*). For the first task, since no other tasks have been executed before, all the tie-points are loaded from the original list. Figure 3 shows a representation of the files loaded in our illustrative example. The figure depicts the positions of the tie-points in the master image and in the related images.

4. Performs the **topological merging** of tie-points into multi-tie-points (MTPs). Because of the format in which the tie-points are stored by *Tapioca*, i.e. by valid image pairs, the algorithm requires a topological merging; this links all the tie-points, which represent the same physical location but are split in different files, into individual entities. A MTP stores the number of valid image pairs in which a related tie-point is present (multiplicity)

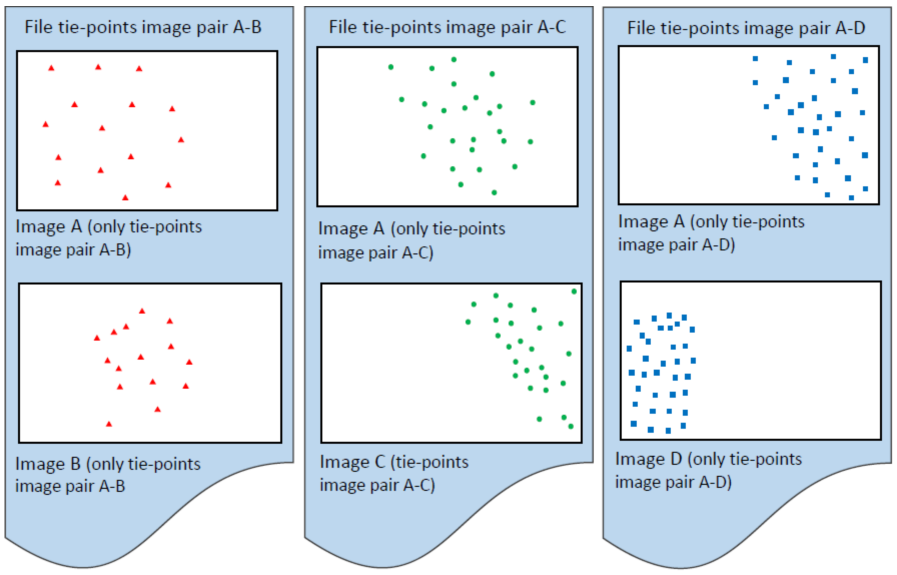

File tie-points image pair A-B
Image A (only tie-points image pair A-B)
Image B (only tie-points image pair A-B
File tie-points image pair A-C
Image A (only tie-points image pair A-C)
Image C (tie-points image pair A-C)
File tie-points image pair A-D
Image A (only tie-points image pair A-D)
Image D (only tie-points image pair A-D)

Figure 3: Loaded tie-points from the valid image pairs composed by the master image and the related images for the first task. In this example, we assume that the related images of image A are images B, C and D. The tie-points are loaded from three different files. Note that for the master image (image A) it may happen that a tie-point from a file "overlaps" with a tie-point from another file; this is the case when the same physical location is visible in more than two images.

as well as the positions of the related tie-points in those images. Figure 4 depicts the MTPs of the first task in our illustrative example. Note that only valid image pairs which include the master image are considered when computing the multiplicity because other valid image pairs were not loaded. This creates the following effect: because of the way *Tapioca* computes the tie-points, it may happen that a tie-point is detected between two image pairs which have one image in common but it is not detected in the image pair of the other two images. For example, a tie-point detected in image pairs A-B and A-C may not be detected in the image pair B-C. This situation happens when the cross-matching process in *Tapioca* for that image pair did not find the correspondence for the tie-point. The MTPs where this situation happens should have a lower multiplicity when compared to MTPs where this does not happen. However, the algorithm does not load image-pairs between related images in an on-going task. Thus, the information on missing tie-points in valid image pairs is partially occluded. Implementing a feature to address the issue would make the algorithm slower because the number of images to handle in each task would be considerably higher. In order to avoid increasing the complexity of the algorithm and the cost of its execution, we decided not to implement it.

5. **Creates grids that divide the image pixel-space for all the images** related to the current task, i.e. including the master image and the related images. The size of the grid determines how many tie-points are desired for the master image. The size is configurable by the user.

6. **Fills in the grids with the MTPs**. When filling in the grid of a certain image, we use the positions of the related tie-points in that image. Note that since a MTP is shared between two or more images, i.e. it has related tie-points in two or more images, its

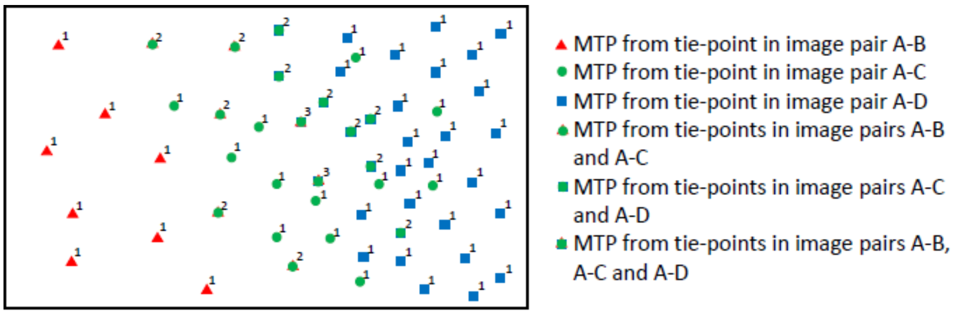

MTP from tie-point in image pair A-B
MTP from tie-point in image pair A-C
MTP from tie-point in image pair A-D
MTP from tie-points in image pairs A-B and A-C
MTP from tie-points in image pairs A-C and A-D
MTP from tie-points in image pairs A-B, A-C and A-D

Figure 4: Multi-tie-points (MTPs) represented by their positions in the master image for the first task. The number next to each MTP indicates the multiplicity, i.e. the number of valid image pairs in which a related tie-point is present.

coordinates in the image spaces of the various images are different. In Figure 5 we depict the filled in grids for the first task of our illustrative example.

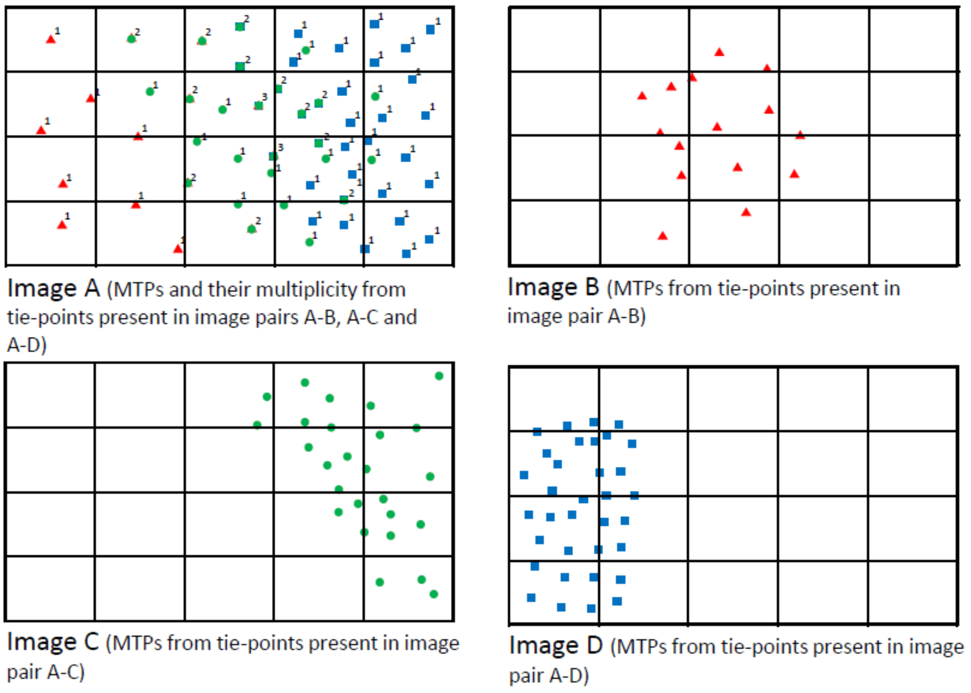


Image A (MTPs and their multiplicity from tie-points present in image pairs A-B, A-C and A-D)
Image B (MTPs from tie-points present in image pair A-B)
Image C (MTPs from tie-points present in image pair A-C)
Image D (MTPs from tie-points present in image pair A-D)

Figure 5: Filled-in grids for the master image and the related images for the first task. In this example we use a grid size of 20 cells.

7. **Attempts to reduce tie-points in the cells of the master image grid**. Next, we will describe the steps performed. For each cell of the master image grid, the algorithm:

   - **Sorts the MTPs according to their** $Gain_{MTP}$. The formula to compute the $Gain_{MTP}$ for a MTP is

$$Gain_{MTP} = \frac{M_{MTP}}{1 + (K \cdot \frac{A_{MTP}}{A_{median}})^2}$$

where M_{MTP} is the multiplicity of the MTP, i.e. the number of valid image pairs where the MTP has related tie-points; K is a threshold value configurable by the user; A_{MTP} is the accuracy value of the MTP, and it is computed as the worst (highest) estimated accuracy of the tie-points related to the MTP (the estimated accuracy of a tie-point is computed from the estimated orientation between the valid image pair where the tie-point is present); and A_{median} is the median of accuracy values of all the MTPs. Note that if K is zero, the $Gain_{MTP}$ is directly the multiplicity. This is an interesting case since no estimated orientation is required. Thus, the tie-points reduction can run right after the tie-point detection.

- **Deletes the MTPs in the current grid cell that meet the conditions for removal except the one with highest** $Gain_{MTP}$. A MTP can be deleted if it meets the following conditions:
  - (Condition 1) It is not present in a related image that was master in an earlier executed task. If the tie-points related to this MTP were deleteable, they would have been deleted before.
  - For each related image where the MTP has a tie-point present: (Condition 2) there is at least another tie-point in the current master grid cell that is also shared with the related image, and (Condition 3) there is at least another tie-point in the grid cell of the related image.

  Figure 6 shows an example of the attempt of removing MTPs in a cell of the master image grid for the first task.
- **Proceeds with the reduction of MTPs in the next grid cell** of the master image.

8. **Stores the reduced tie-points** (related to the MTPs that are not deleted) after all the cells of the master image grid have been processed.

9. **Proceeds with the execution of the next task** (with a new master image) until all the tasks have been executed.

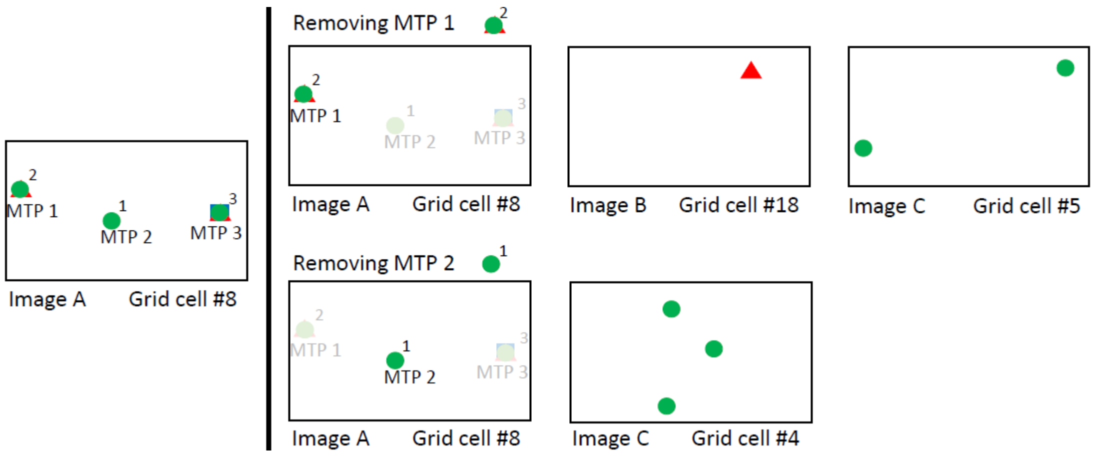


Figure 6: Example of removing MTPs in the cell #8 of the master image grid for the first task in our illustrative example. Left: Cell #8 of the master image grid in the first task. The numbering of the cells starts at #1 in the top-left position of the grid and the order is row-wise. For the sake of simplicity in this example, we have used $K = 0$ for the computation of the $Gain_{MTP}$ values. Thus, $Gain_{MTP}$ is the multiplicity. In this cell, the algorithm will attempt to remove the MTPs with lower $Gain_{MTP}$, the MTPs #1 and #2 respectively. Right: Attempt to remove the MTPs. Regarding MTP #1: Condition 1 is met, it is always met in the first task because no MTPs have been reduced yet; Condition 2 is met because in the master image grid cell #8 there is another MTP shared with image B (red triangle) and two shared with image C (green circle); Condition 3 is not met because by removing this MTP, the grid cell #18 in image B, which is the cell in the image #2 grid where the related tie-point is present, would become empty. Therefore, this MTP cannot be removed. Regarding MTP #2: Condition 1 is met; Condition 2 is met because in the master image grid cell #8 there are two shared MTPs with image C (green circle); Condition 3 is met because in the grid cell #4 in image C, which is the cell in the image C grid where the related tie-point is present, there are two more tie-points. Therefore, this MTP can be deleted.

3.1.2 Parallel algorithm

In the algorithm performed in *RedTieP*, there is cross-dependency of valid image pairs in the several tasks of the tie-points reduction step. The *ith* task reduces tie-points from all the valid image pairs between the *ith* image (master) and its related images. This characteristic causes that the tasks parallelization is not trivial. Concretely, inconsistencies are created if tasks executed in parallel attempt to read, reduce and write tie-points related to the same valid image pairs.

We have developed a parallel execution mode for the presented algorithm. To overcome the described cross-dependency of valid image pairs, the parallel algorithm guarantees that all the tasks running in parallel, at any time, process different valid image pairs. This characteristic defines an execution workflow with tasks that are mutually exclusive for parallel execution depending on their involved valid image pairs. We use *Noodles* ([14]; http://nlesc.github.io/noodles/) for the execution of the tasks in parallel. *Noodles* is a programmable workflow engine implemented in Python. First, the various tasks are defined and their mutual execution restrictions are specified; then, *Noodles* finds a coherent parallelization strategy and executes it using as many cores as indicated by the user. *Noodles* does not guarantee any specific task execution order. Any ordering specified by the user is ignored.

3.1.3 User options

RedTieP offers a set of options to configure its tie-point removal process. The options that can be configured are:

- The **image sorting** affects the order in which the images are processed, and four options are available: by name in ascending order (this usually matches the acquisition order), by name in descending order, by number of tie-points in ascending order, and by number of tie-points in descending order.
- The **grid size** determines the number of tie-points of the reduced set. Additionally, there is an adaptive mode that, if enabled, enlarge the grid size in images that have lower number of tie-points compared to others.
- The value of $\boldsymbol{K}$ in the $Gain_{MTP}$ formula. This determines the weight of the accuracy of a MTP with respect to its multiplicity. A low value of K gives more weight to the the multiplicity, and, if $K=0$, then the $Gain_{MTP}$ is directly the multiplicity.
- The use **parallel execution** with *Noodles*. In this case the image sorting is not determined by the user choice, but by the best parallelization strategy that *Noodles* can find.

3.2 *OriRedTieP*

The second option for the tie-points reduction step is to use *OriRedTieP*. Before its execution, this tool requires the initialization of the positions and orientation parameters of the whole image set. This can be done with the *Martini* tool. Initializing the parameters for the whole image set is a slightly more complicated process compared to the processing done by *NO_AllOri2Im* (the tool executed before *RedTieP*). In that case, we “only” estimated the relative orientation between valid image pairs, while *Martini* deals with the whole image set as a block.

3.2.1 Steps

Figure 7 depicts an overview of the steps performed by *OriRedTieP*. Next, we will describe its steps. The tool:

1. **Projects the images in the 3D space**. These projections define 2D footprints in the 3D space and can be computed from the estimation of camera positions and orientations. Figure 8 is a graphical explanation of the 3D projections of the images.
2. **Determines a 2D bounding box** (XY) in the 3D space of the projected footprints of all the images, **and defines the tiles** that divide the space of the bounding box. For each tile, the algorithm stores the list of images whose 2D footprint (in the 3D space) overlaps the area defined by the tile. In Figure 8 depicts an example of a bounding box and the tiles that are defined.
3. Performs the **tie-points reduction**. The various tiles are processed. This processing can be done in parallel, and for each tile various steps are performed. As an illustrative example, Figures 9 to 11 explain the steps in a graphical manner. These figures follow the

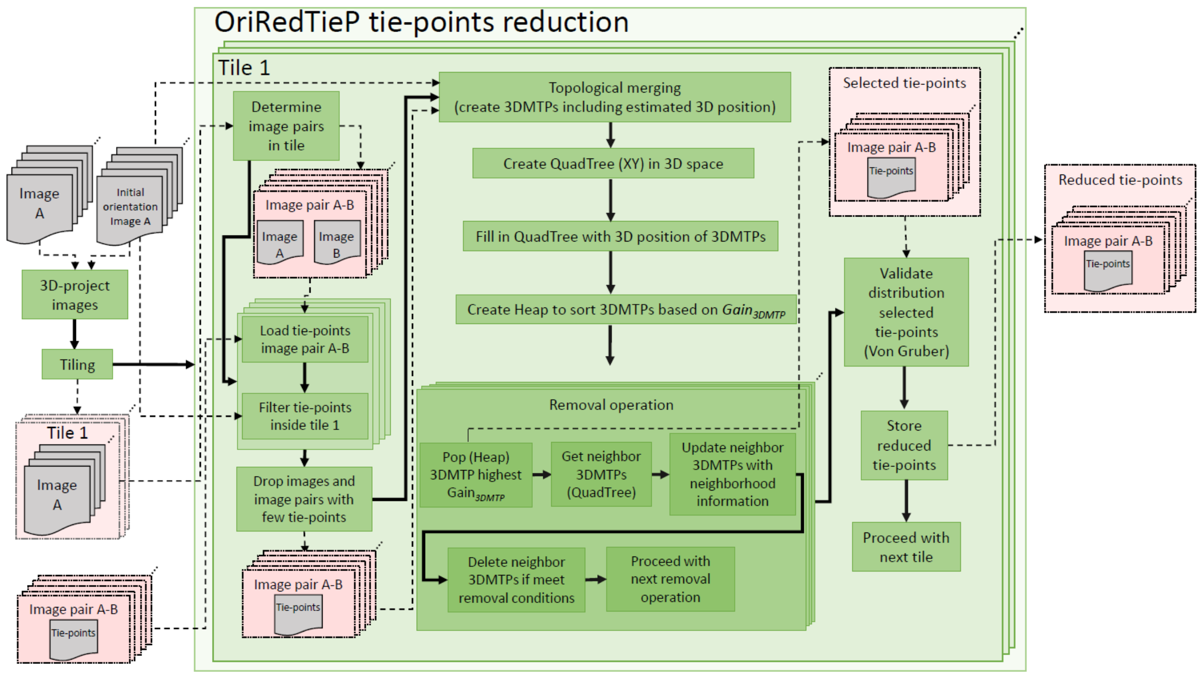

OriRedTieP tie-points reduction
Tile 1
Image A
Initial orientation Image A
3D-project images
Tiling
Tile 1
Image A
Image pair A-B
Tie-points
Determine image pairs in tile
Image pair A-B
Image A
Image B
Load tie-points image pair A-B
Filter tie-points inside tile 1
Drop images and image pairs with few tie-points
Image pair A-B
Tie-points
Topological merging (create 3DMTPs including estimated 3D position)
Create QuadTree (XY) in 3D space
Fill in QuadTree with 3D position of 3DMTPs
Create Heap to sort 3DMTPs based on Gain3DMTP
Removal operation
Pop (Heap) 3DMTP highest Gain3DMTP
Get neighbor 3DMTPs (QuadTree)
Update neighbor 3DMTPs with neighborhood information
Delete neighbor 3DMTPs if meet removal conditions
Proceed with next removal operation
Selected tie-points
Image pair A-B
Tie-points
Validate distribution selected tie-points (Von Gruber)
Store reduced tie-points
Proceed with next tile
Reduced tie-points
Image pair A-B
Tie-points

Figure 7: Overview of the steps performed by *OriRedTieP*.

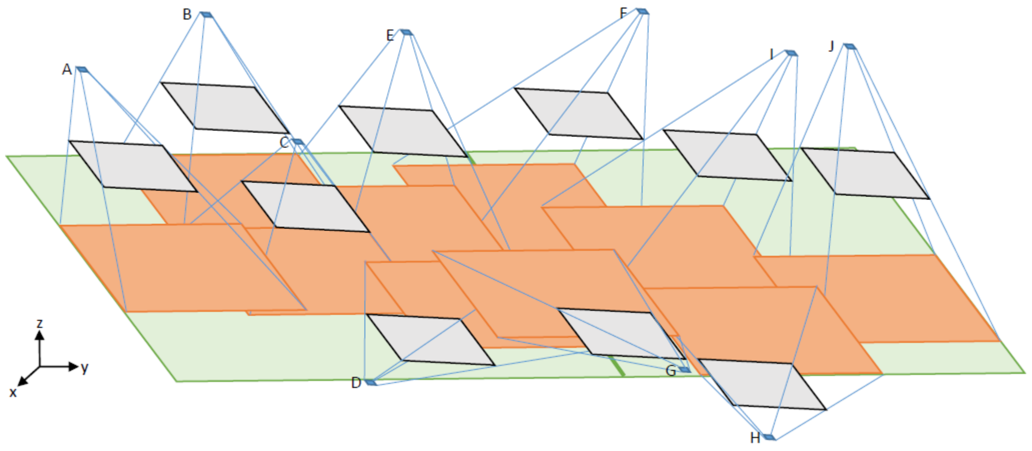


Figure 8: Example of projecting images in the 3D space, determining the bounding box and defining the tiles. In this example, two tiles are defined: the first tile overlaps images A, B, C, D, E, F and G; and the second tile overlaps images D, F, G, H, I and J.

example used in Figure 8. Next, we will describe the steps performed in the processing of each tile. In each tile, the algorithm:

(a) **Loads the tie-points from all the valid image pairs between the images whose 2D footprint overlaps the area defined by the current tile**. When loading the tie-points of a valid image pair, the estimated positions of the tie-points in the 3D space are computed. This is done using the intersection of the tie-points positions in the images of the image pair and the initial estimation of the camera positions and orientations. This is illustrated in the example shown in Figure 9.

(b) **Filter out the tie-points that lay outside the tile**. Since the footprint of an image may be only partially overlapping a tile, it may happen that tie-points loaded from a valid image pair have some tie-points that are actually outside the tile. The estimated 3D positions of the tie-points are used to decide which tie-points must be processed in the current tile.

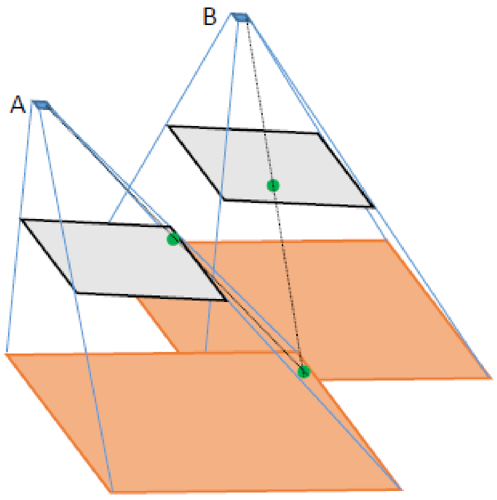


Figure 9: Example of computing the position of a tie-point in the 3D space. This is computed from the tie-point positions in the images of the image pair, and from the estimation of the positions and orientations of the camera.

(c) **Drops valid image pairs and images with a low number of tie-points** (the thresholds are configurable by the user) overlapping the current tile. This may happen if almost all the related tie-points are outside the tile. The images and valid image pairs that have very few tie-points are dropped in the processing of the current tile. This means their related tie-points are also dropped.

(d) Performs the **topological merging** of tie-points into 3D multi-tie-points (3DMTPs). These are similar to the MTPs defined in the *RedTieP* algorithm (see Subsection 3.1.1, step 4). A 3DMTP stores the number of valid image pairs in which a related tie-point is present (multiplicity) as well as the positions of the related tie-points in those images. However, a 3DMTP also stores a 3D position. The 3D position is computed by (i) using the initial estimation of camera positions and orientations, and (ii) by projecting in 3D and intersecting the tie-points positions in the two or more images where the 3DMTP has related tie-points. Note that it is possible that more than one valid image pair is used for the computation of the 3D position (contrary to the 3D position estimated in the step 4.a of this algorithm). See Figure 10 for an illustrative example. Also note that in the 3DMTPs all the valid image pairs are considered, this is contrary to the MTPs used in *RedTieP* which only considered valid image pairs that included the master image. However, not all the valid image pairs have necessarily tie-points related to the 3DMTP. The reason is the same as the one already described in the *RedTieP* algorithm (see Subsection 3.1.1, step 3.d).

(e) **Creates a QuadTree structure with the XY extent in the 3D space** of the current tile being processed.

(f) **Fills in the QuadTree with the 3DMTPs**. The estimated 3D positions are used.

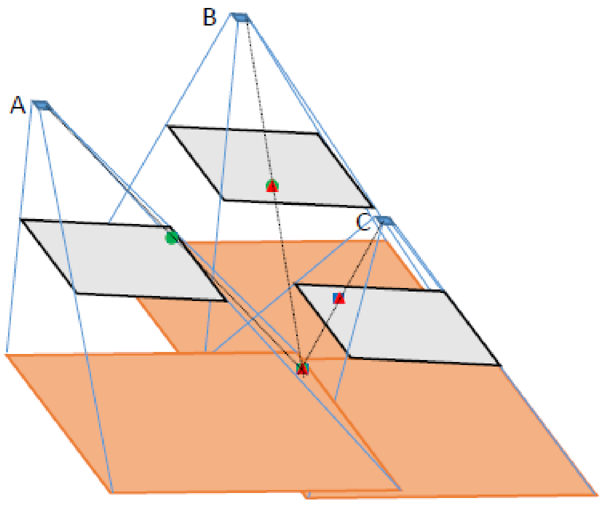


Figure 10: Example of computing the 3D position of a 3DMTP from the related tie-points in three images. This computation is part of the topological merging where the 3DMTPs are created. In this example we assume that related tie-points were detected in all the possible image pairs between the three images, i.e. image pairs A-B, A-C and B-C. Thus, this 3DMTP has a multiplicity of 3.

(g) **Creates a heap data structure** that contains the 3DMTPs sorted according to their $Gain_{3DMTP}$. The formula to compute $Gain_{3DMTP}$ is similar to the one defined in *RedTieP* for computing $Gain_{MTP}$ (see Subsection 3.1.1, step 7). However, in this case it is dynamic. The formula is

$$Gain_{3DMTP} = \frac{M_{3DMTP}}{1 + (K \cdot \frac{E_{3DMTP}}{E_{median}})^2} \cdot (0.5 + \frac{n_{3DMTP}}{N_{3DMTP}})$$

where M_{3DMTP} is the multiplicity of the 3DMTP, i.e. the number of valid image pairs where the 3DMTP has related tie-points; K is a threshold value configurable by the user; E_{3DMTP} is the estimated error when computing the 3D position of the 3DMTP; E_{median} is the median of estimated errors of all the 3DMTPs; N_{3DMTP} is the number of images where a related tie-point is present (note that because of possible missing related tie-points in valid image pairs, M_{3DMTP} can not be directly computed from N_{3DMTP}); and n_{3DMTP} is the number of images, from the ones where a related tie-point is present, that meet the following condition: there is not any 3DMTP already selected for keeping and nearby which have related tie-points in the same image. Thus, the value of n_{3DMTP} changes when a nearby 3DMTP is selected for keeping.

(h) Performs a set of **removal operations**. In each removal operation, it attempts to remove tie-points using the QuadTree and the heap data structure. Figure 11 depicts an example of the first four removal operations. In each removal operation, the algorithm:

- **Pops the 3DMTP with the highest** $Gain_{3DMTP}$ from the heap and selects it for keeping.
- **Gets the neighbor 3DMTPs using the QuadTree**. These are within a given distance (configurable by the user) to the 3DMTP extracted from the

heap.

- **Updates the n_{3DMTP} and the $Gain_{3DMTP}$ values of the neighbor 3DMTPs.** This takes into account which images have tie-points related to the 3DMTP selected for keeping in this removal operation.
- **Deletes the neighbor 3DMTPs whose n_{3DMTP} value is zero**. The n_{3DMTP} value of a 3DMTP is zero if 3DMTPs already selected for keeping and near to this 3DMTPs have related tie-points in all the images where this 3DTMP also has related tie-points.
- **Proceeds with the next removal operation** where a new 3DMTP with the highest $Gain_{3DMTP}$ is subtracted from the heap. Note that the 3DMTPs in the heap data structure are resorted every time a 3DMTP is subtracted or removed from the heap and also when the $Gain_{3DMTP}$ of a 3DMTP is updated.

Note that this removal process only considers for the 3DMTPs the images in which related tie-points are present, but it does not consider the image pairs. Because of possible missing related tie-points in valid image pairs, an additional step to guarantee a good distribution in the image pairs is required.

(i) **Validates the proper distribution of the related tie-points in the selected 3DMTPs in the image pairs**. This is done based on the von Gruber point locations ([28], pp. 574-577). The algorithm performs for all the valid image pairs in the current tile the following checking: for each deleted tie-point, the algorithm checks which is the closest tie-point which has been selected for keeping. If that tie-point is too far, then the deleted tie-point is “un-deleted” and added for keeping. The distance in which a tie-point is considered too far can be configured by the user.

(j) **Stores the reduced set of tie-points** after a proper tie-point distribution is guaranteed in the images and image pairs.

(k) **Proceeds with the processing of the next tile** until all the tiles have been processed.

3.2.2 User options

OriRedTieP offers a set of options to configure its tie-point removal process. The options that can be configured are:

- The **tile size** determines the number of tiles in which the 3D space will be divided. Smaller tile sizes means more tiles. By default, the algorithm parallelizes the processing for the different tiles. However, the user can choose to deactivate the parallel processing.
- The **precision threshold** determines the maximum error allowed in the computation of

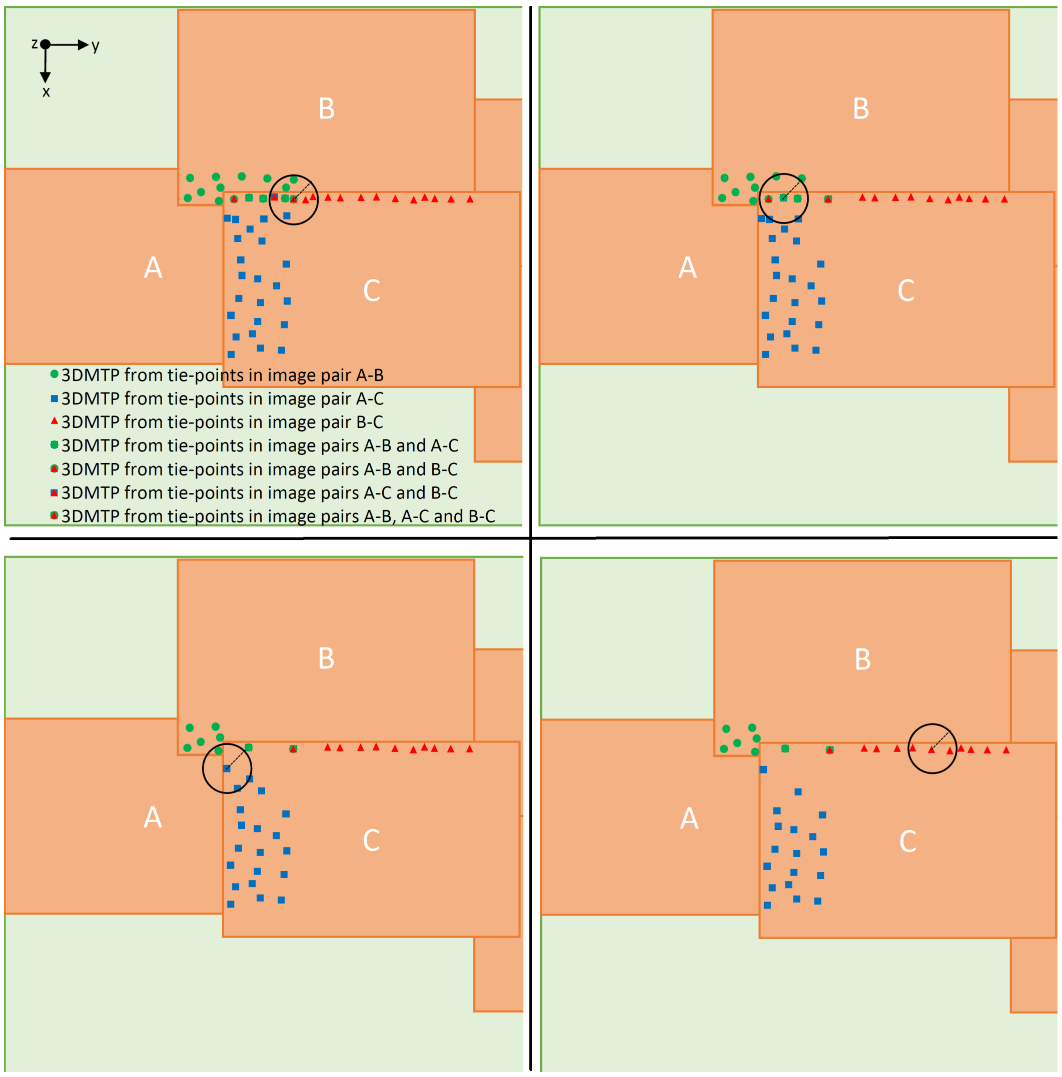

z
y
x
A
B
C
● 3DMTP from tie-points in image pair A-B
■ 3DMTP from tie-points in image pair A-C
▲ 3DMTP from tie-points in image pair B-C
■ 3DMTP from tie-points in image pairs A-B and A-C
● 3DMTP from tie-points in image pairs A-B and B-C
■ 3DMTP from tie-points in image pairs A-C and B-C
● 3DMTP from tie-points in image pairs A-B, A-C and B-C

Figure 11: Example of the first four removal operations of the tie-points reduction for the first tile. For the sake of simplicity, only the tie-points from images A, B and C are considered. In the first removal operation (top-left image) the 3DMTP with the highest $Gain_{3DMTP}$ is subtracted and selected for keeping. The neighbor 3DMTPs are searched and their n_{3DMTP} values are updated. In this case, the 3DMTP selected for keeping has tie-points in all images A, B and C. Thus, all the neighbor 3DMTPs can be deleted. After subtracting the 3DMTP for keeping and deleting the neighbor 3DMTPs the heap data structure resorts the 3DMTPs.

Figure 11: (Previous page.) In the second removal operation (top-right image) another 3DMTP is subtracted and selected for keeping. The neighbor 3DMTPs to this 3DMTP are searched and their n_{3DMTP} values are updated. In this case the 3DMTP selected for keeping (in this iteration) has related tie-points in image pairs A-B and A-C but not in image pair B-C. Since the 3DMTP has related tie-points in all the images (even with the missing related tie-point in image pair B-C), all the neighbor 3DMTPs are deleted. Note that in this case one of the neighbors that gets deleted had a related tie-point in image pair B-C. Situations like this one, if not solved, could damage the distribution of tie-points in images and image-pairs. Therefore, an extra step after the removal operations must be performed to guarantee good tie-point distribution. In the third removal operation (bottom-left image) another 3DMTP is subtracted and selected for keeping. In this case this 3DMTP only has related tie-points in images A and C. Thus, from its neighbor 3DMTPs two can be deleted (blue rectangles) but one can not be deleted (green circle). In the forth removal operation (bottom-right image) another 3DMTP is subtracted and selected for keeping, and all its neighbor 3DMTPs can be deleted because they are all related to the same images.

the 3D position of a tie-point from the image positions in an image pair. If the computation of the 3D position of a tie-point has an error larger than the threshold, the tie-point is deleted.

- The **neighbor distance factor** affects the radius at which neighbor 3DMTPs are searched in the tie-point removal process. For larger factors more neighbor 3DMTPs are found and deleted.
- The **Von Gruber multiplier** is applied to the neighbor distance factor. The result is used in the check for proper tie-point distribution in image pairs using the Von Gruber locations. It determines how far a deleted tie-point can be from its closest tie-point selected for keeping.
- The value of $\boldsymbol{K}$ in the $Gain_{3DMTP}$ formula. This determines the weight of the error in 3D position estimation of a 3DMTP with respect to its multiplicity. A low value of K gives more weight to the the multiplicity.

4 Experiments

4.1 Definition

We defined a set of three experiments with various image sets to assess the effect of using the tie-points reduction tools in photogrammetric workflows implemented with MicMac. Particularly, we checked the effect on the memory requirements and processing time of the bundle adjustment (*Tapas* in MicMac) and in the quality of the generated dense point cloud. For all the image sets we have collected (or plan to collect) GCPs. A GCP is a point from which we know its 3D position in the real world and its position in the images. GCPs are used to check the correctness of the estimation of camera positions and orientations performed by the bundle adjustment. Since the quality of the generated dense point cloud depends on the proper estimation of the camera positions and orientations, we can use the GCPs to quantify the effect on the quality of the dense point cloud.

Experiment	**Image set**	**#Images**	**Image size [pixels]**	**Total size [MB]**	**Description**
small	Aquileia	18	4,592x3,056	146	Archaeological site in Aquileia (Italy)
medium	Gronau	570	5,184x3,456	5,264	Square in city center of Gronau (Netherlands)
big	Zwolle	1,196	17,310x11,310	218,168	Zwolle city and surroundings (3% Netherlands)

Table 1: Experiments and datasets

Table 1 shows the different experiments and image sets. The main idea of having three different experiments is as follows: in the *small* experiment we use an image set with a low number of images, and we test a large number of combinations of options of the tie-points reduction tools. This experiment is used to find the most promising combinations of options, i.e. the ones that offer good results. These combinations are further tested in the *medium* experiment with a larger image set. We find the best combinations, and we test them in the *big* experiment with an even larger image set. In all the experiments we follow the same procedure. Next, we will describe the steps. In each experiment, we:

1. Run *Tapioca* to detect the tie-points.
2. Run *Tapas* with all the tie-points. In the *medium* and *big* experiments an initialization step is also run. A small subset of the images is used to obtain a estimation of the calibration parameters.
3. Run *GCPBascule* with the available GCPs to assess the quality of the estimated parameters in *Tapas* when using all the tie-points. *GCPBascule* is a tool included in MicMac. First, *GCPBascule* transforms the estimated camera positions and orientations to the correct reference system by fitting a 3D transformation with some of the GCPs - normally 4-5 GCPs equally distributed in the 3D space. For each GCP, a 3D transformation fitting error is computed. Second, *GCPBascule* computes the errors produced in the rest of GCPs, which we call Check Points (CPs), when using the fitted camera positions and orientations.
4. Define which combinations of options we want to test for the *RedTieP* tool.
5. For each combination of options:
   (a) Run *RedTieP*. Note that before running *RedTieP*, we first have to run *No_AllOri2Im*.
   (b) Run *Tapas* with the reduced set of tie-points.
   (c) Run *GCPBascule* to obtain the errors in the GCPs and CPs.
   (d) Compare the errors obtained by *GCPBascule* in the previous step (5.c) with the ones obtained in the step 3 (no tie-points reduction).
6. Re-do steps 4 and 5, but with the *OriRedTieP* tool. Note that in this case, we have to run first *Martini* (instead of *OriRedTieP*).

4.2 Execution

During the project, the *small* and *medium* experiments have been executed. However, the *big* experiment could not be executed and will be done in the future.

For the *small* and *medium* experiments, the hardware system used was a Virtual Machine (VM) with Ubuntu 14.04, 4 Intel i7-4720HQ cores at 2.6 GHz, 13.4 GB RAM and a virtual hard disk of 200 GB created from a 7,200 rpm SATA disk of 1 TB. For the *big* experiment, we plan to use a server running Ubuntu 16.04 with 24 Intel Xeon E5-2643 cores at 3.4 GHz, 128 GB RAM and an external 5,400 rpm SATA disk of 2 TB.

4.2.1 *Small* experiment

In this experiment we used the *Aquileia* image set. For both tools we tested several combinations of their options. Even though the image set was too small to derive meaningful conclusions, it was useful to extract some facts.

RedTieP

We tested various image sorting options, grid sizes with and without adaptive modes, and values for K. We also tested the use of *Noodles* to parallelize the processing. In general, we observed that the errors obtained by *GCPBascule* when using *RedTieP* were in the same order of magnitude as when no tie-points reduction was applied, and in some cases the errors were even better. However, the difference was small, and we believe it was caused essentially by noise of the bundle adjustment process. Regarding the processing time in *Tapas*, we observed that it decreases with the same factor as the tie-points reduction factor. Thus, when tie-points are reduced to 10%, the processing time is also decreased to 10%. Regarding memory consumption, we observed a decrease factor proportional to the tie-points reduction factor.

We observed that even though larger grids means a larger set of reduced tie-points, this does not necessarily mean better results. We also found that using the adaptive mode is not worthy, it is better to simply use a larger grid. We observed that a K value of 0 gives results similar to when K is not 0. Using *Noodles* made *RedTieP* faster, the decrease factor of the processing time in *RedTieP* was roughly the same as the number of cores used.

OriRedTieP

We tested various precision thresholds, neighbor distance factors, Von Gruber multipliers and K values. Similarly to *RedTieP*, the errors obtained by *GCPBascule* are were in the same order of magnitude as when no tie-points reduction was applied. As expected, the same behavior was observed regarding the processing time in *Tapas* and its memory consumption.

We observed that the number of tie-points in the reduced set increases proportionally with the neighbor distance factor and the Von Gruber Multiplier. However, as in *RedTieP*, a larger set of reduced tie-points does not necessarily means better results. We observed that the best value for K was 0.5.

4.2.2 *Medium* experiment

Even though the image set was small and more testing is required, the *small* experiment showed that using smart tie-points reduction with the implemented tools is promising. The reason is that we did not observe significant damage in the estimation of the camera positions and orientations. The experiment was also useful to identify some optimal values for the configuration parameters of both tools. In *RedTieP* it is recommended to use image sorting by name in ascending order, and to avoid using adaptive grids. Setting K to 0 did not show significant degrade of the results. This an interesting case because no estimated orientation is required if K is 0, and this means that in principle the tie-points reduction could run just after the tie-points reduction without needing *NO_AllOri2Im*. In *OriRedTieP* it is recommended to use the value of 0.5 for K. In this case, even if K is set to 0, the estimated orientation of the whole block is still required to estimate the 3D positions of the 3DMTPs.

In the *medium* experiment we used the *Gronau* image set, and we tested some combination of options of the tie-points reduction tools. We targeted at obtaining, for each tool, two different sizes for the set of reduced tie-points: one with around 5% of the original number of tie-points and the other with around 10%. For the *RedTieP* this was done by using grid sizes of 12x12 and 20x20, while for the *OriRedTieP* this was done with using neighbor distance factors of 65 and 40. In *RedTieP* we tested each grid size with and without *Noodles* and with different values of K. In *OriRedTieP* we tested each neighbor distance factor with different threshold precisions and Von Gruber multipliers.

Table 2 shows the results of the *medium* experiment. The processing time for *Tapioca* (tie-point detection) is not included in the table because it is common in all the photogrammetric workflows tested. In the hardware system used *Tapioca* took 292,318.26 seconds. Note that *Tapioca* is actually more time consuming than *Tapas* even if no tie-points reduction is done. Therefore, speeding up the tie-point detection is also an important task. We have also work on this but it was not the focus of this report. For more information on this topic see our work-in-progress paper.

Regarding the results of the photogrammetric workflows using *RedTiep* (upper rows in Table 2) we can extract that:

- If using grid sizes of 12x12, we get tie-points reduction factor of around 6-7%, if using 20x20 they are of around 10-11%.
- The *RedTieP* processing time when using *Noodles* (with four cores) is around three times faster when compared to using the same set of options but without *Noodles*.
- We do not observe dependence of the processing time in *RedTieP* with the grid size (tie-points reduction factor).
- If using *RedTieP* in the workflows, the processing times and memory usages of *Tapas* decrease. The decrement depends on the tie-points reduction factor. For the smallest tie-points reduction factor (0.0656), the processing time decreases 96.82% and the memory usage decreases 90.26%.
- There is an interesting effect when using *Noodles*. Tapas is around 30-40% faster compared to using the same set of options in *RedTieP* but without *Noodles*. It is also using less memory. The number of iterations of *Tapas* is reduced 30-40% which is probably the cause of the decrements in the processing times.
- The *Tapas* residuals in all the workflows where *RedTieP* was used are similar, and they are all worse that when no tie-points reduction is used (first row in the table), and in the worst case the residual is increased 14%.
- The mean in GCPs errors computed by *GCPBascule* when no tie-points reduction is used is 0.0814. When using *RedTieP*, this value in in five situations even better (smaller).
- The mean in CPs errors computed by *GCPBascule* when not using *RedTieP* is 0.0902. When using *RedTieP*, this value is normally slightly worse except three cases where using *RedTieP* is actually better.

					Tapas				**GCPBascule**			
									GCPs		**CPs**	
					Time[s]	**Mem.**	**#It.**	**Res.**	**Mean**	**Std.**	**Mean**	**Std.**
No tie-points reduction					68,948.50	63.16	160	0.6423	0.0814	0.0337	0.0902	0.0489
RedTieP												
Grid	***K***	**Noodles**	**Size Red.**	**Time[s]**								
12x12	0.0	N	0.0678	1,512.03 + 3,216.88	3,039.09	6.59	132	0.7287	0.0934	0.0239	0.1045	0.0599
12x12	0.0	Y	0.0656	1,512.03 + 1,168.30	2,190.68	6.15	89	0.7327	0.0815	0.0258	0.0968	0.0517
12x12	0.5	N	0.0721	1,512.03 + 3,193.48	3,969.40	7.15	157	0.7284	0.0732	0.0278	0.0876	0.0484
12x12	0.5	Y	0.0699	1,512.03 + 1,104.91	2,299.47	6.52	91	0.7336	0.0837	0.0211	0.0979	0.0527
12x12	0.25	N	0.0727	1,512.03 + 3,457.58	4,143.76	7.11	157	0.7257	0.0704	0.0237	0.0884	0.0473
12x12	0.25	Y	0.0703	1,512.03 + 1,063.20	2,205.58	6.49	88	0.7274	0.1145	0.0289	0.1221	0.0711
12x12	0.75	N	0.0711	1,512.03 + 3,212.84	3,996.07	7.13	159	0.7295	0.0784	0.0292	0.0927	0.0500
12x12	0.75	Y	0.0684	1,512.03 + 1,067.74	2,263.47	6.49	92	0.7340	0.0770	0.0266	0.0884	0.0511
20x20	0.0	N	0.1023	1,512.03 + 3,199.66	4,393.28	8.81	126	0.7351	0.0893	0.0201	0.1029	0.0567
20x20	0.0	Y	0.0997	1,512.03 + 1,133.32	3,367.35	8.39	90	0.7358	0.0970	0.0219	0.1090	0.0608
20x20	0.5	N	0.1102	1,512.03 + 3,198.78	6,196.00	9.67	162	0.7320	0.0812	0.0189	0.0953	0.0525
20x20	0.5	Y	0.1056	1,512.03 + 1,114.75	3,312.86	8.89	87	0.7365	0.1032	0.0379	0.1083	0.0662
OriRedTieP												
Thres.	**NDF**	**VGM**	**Size Red.**	**Time[s]**								
5.0	65	1.5	0.0654	9,151.24 + 1,814.34	1,543.83	5.53	50	0.6845	0.1034	0.0289	0.1104	0.0646
20.0	65	1.5	0.0660	9,151.24 + 1,895.16	1,619.79	5.57	50	0.6790	0.1130	0.0293	0.1194	0.0705
5.0	65	1.75	0.0617	9,151.24 + 1,777.20	1,476.58	5.25	51	0.6915	0.1125	0.0298	0.1178	0.0708
20.0	65	1.75	0.0621	9,151.24 + 1,785.26	1,402.70	5.34	49	0.6858	0.1080	0.0287	0.1144	0.0676
5.0	40	1.5	0.1199	9,151.24 + 1,228.23	2,845.47	8.88	50	0.6808	0.1127	0.0258	0.1209	0.0711
20.0	40	1.5	0.1199	9,151.24 + 1,123.83	2,849.77	8.91	50	0.6756	0.1141	0.0286	0.1220	0.0728
5.0	40	1.75	0.1106	9,151.24 + 1,133.63	2,745.68	8.25	52	0.6906	0.1152	0.0266	0.1237	0.0728
20.0	40	1.75	0.1105	9,151.24 + 1,127.70	2,588.80	8.23	49	0.6852	0.1191	0.0300	0.1275	0.0759

Table 2: Results of the *medium* experiment with the *Gronau* image set. A photogrammetric workflow without tie-points reduction (*Tapioca* - *Tapas* - *GCPBascule*) is compared with photogrammetric workflows when tie-points reductions is done with *RedTiep* (*Tapioca* - *NO_AllOri2IM* - *RedTieP* - *Tapas* - *GCPBascule*) and when it is done with *OriRedTieP* (*Tapioca* - *Martini* - *OriRedTieP* - *Tapas* - *GCPBascule*). Several user options are tested for both tie-points reduction tools. **Thres.** is the precision threshold; **NDF** is the neighbor distance factor; **VGM** is the Von Gruber multiplier; **Size Red.** is the tie-points reduction factor, i.e. the size of the set of reduced tie-points compared to the original one; **#It.** is the number of iteration *Tapas* performed; and **Res.** is the average image orientation residual of the bundle adjustment. Note that **Time[s]** in the *RedTieP* includes 1,512.03 seconds and in the *OriRedTieP* it includes 9,151.24 seconds. These are the times of *NO_AllOri2Im* and *Martini* respectively. The units of the residuals in *Tapas* are in image pixels (re-projection errors), while the units in which the GCPs/CPs are specified (meters in this experiment) are used for the errors provided by *GCPBascule*.

- Using *Noodles* in *RedTiep* does not seem to have a noticeable effect on observed the GCPs and CPs errors.

- Regarding the value of **K**, setting it to 0 gives slightly worse results. From the K values tested which were different than 0, we could not observe noticeable differences in the GCPs and CPs errors by using one or others.

- As observed in the *small* experiment, the numbers with the bigger gird (20x20) are not better than using 12x12.

Regarding the results of the photogrammetric workflows using *OriRedTiep* (lower rows in Table 2) we can extract that:

- The tie-points reduction factor depends on the neighbor distance factor and the Von Gruber multiplier (NDF and VGM in the table). With a neighbor distance factor of 65 and a Von Gruber multiplier of 1.5, we get a tie-points reduction factor of around 6%. With a neighbor distance factor of 40 and the same Von Gruber multiplier, we get a tie-points reduction factor of around 11%. Increasing the Von Gruber multiplier (increases the allowed maximum distance between a selected tie-point and an unselected one) reduces the size of the reduced tie-points set.

- Contrary to *RedTieP*, the processing time of *OriRedTieP* depends on the size of the reduced set of tie-points. It increases inversely proportional to the tie-points reduction factor. Thus, the more points it has to remove, the more time it takes. Because of the need to run *Martini*, the tie-points reduction with *OriRedTieP* is more time consuming than with *RedTieP*.

- We tested the values of 5.0 and 20.0 for the precision threshold. For these different values, we do not observe any change in the *OriRedTieP* processing time nor in the tie-points reduction factor.

- Similarly to *RedTieP*, using *OriRedTieP* decreases the processing time and the memory usage of *Tapas*. The decrement also depends on the tie-points reduction factor. For the smallest tie-points reduction factor (0.0617), the processing time decreases 97.86 % and the memory usage decreases 91.69%. The decrement is slightly more pronounced when using *OriRedTieP* instead of *RedTieP*.

- Regarding the number of iterations in Tapas, the workflows where *OriRedTieP* was used required much less iterations than when no tie-points reductions is done and also less than if using *RedTieP*.

- The *Tapas* residuals in workflows with *OriRedTieP* are smaller than the ones in workflows with *RedTieP*. However, they are not as good as if no tie-points reduction is done. In the worst case, the residual is increased 7%.

- The means in GCPs and in CPs computed by *GCPBascule* are always worse if using *OriRedTieP* than if no tie-points reduction is done. In general, we can observe that the errors are smaller in the workflows using *RedTieP*.

- Similarly to *RedTieP*, having larger sets of reduced tie-points does not necessarily mean better results in the GCPs and CPs errors.

- For the values tested for the precision threshold, we do not observe any change in *Tapas*, but the *GCPBascule* results are better with the value of 5.0.

- For the values tested for the Von Gruber multiplier, we observe that both *Tapas* and *GCPBascule* results are better with the value of 1.5.

Concerning the best combination of options for *RedTieP*, aiming at a 95% tie-points reduction factor gives better results than aiming at 90%. Thus, for image sets with 15 Megapixels using grids size of 12x12 is advisable. The most optimal value for K is not clear. Thus, we advise to use the default value of 0.5. Using *Noodles* is also advisable.

Concerning the best combination of options for *OriRedTieP*, we also advise aiming at 95% tie-points reduction factor. This can be achieved by using a neighbor distance factor of 65. Using a precision threshold of 5.0 and a Von Gruber multiplier of 1.5 offers the best results.

With the found best combination of options, both tools achieve to decrease the processing time and the memory usage of *Tapas* in around 90% while also decreasing the number of iterations (up to 70% when using *OriRedTieP*, and up to 50% when using *RedTieP* with *Noodles*). The residuals in *Tapas* when using tie-points reductions tools slightly degrade, around 14% if using *RedTieP* and around 7% if using *OriRedTieP*. The mean of GCPs errors increases 3% if using *RedTieP*, and 27% if using *OriRedTieP*.

4.2.3 *Big* experiment

The last phase of the experiments is the *big* experiment where we will use the *Zwolle* image set. First, we will run *Tapas* without tie-points reduction; then, for each tool implemented, we will run tie-points reduction using the best configuration of options for the tool as found in the *medium* experiment.

The images of the *Zwolle* image set have a size of 200 Megapixels. In order to aim at a tie-points reduction factor of 95% with *RedTieP*, we will use a grid of 36x36, a K value of 0.5 and *Noodles*. To aim at the same reduction factor with *OriRedTieP*, we will use a precision threshold of 5.0, a neighbor distance factor of 65 and a Von Gruber multiplier of 1.5.

5 Conclusions

In this report we have proposed a tie-points reduction step that runs after the tie-points detection in photogrammetric workflows. In this way, we achieve a decrement of the memory requirements and the processing time of the bundle adjustment (the main operation in the estimation of camera positions and orientation and of calibration parameters). The effect on the quality of the bundle adjustment results is small. The execution of the tie-points reduction step eases the bundle adjustment processing of large image sets, and it enables the processing in hardware systems that could not be used before.

We have presented two algorithms to perform the tie-points reduction. The algorithms are implemented as stand-alone tools within the MicMac photogrammetry suite. In the first tool, which is called *RedTieP*, the algorithm reduces the tie-points using the position of the tie-points in the images spaces. In the second tool, which is called *OriRedTieP*, the algorithm uses the estimated 3D position of the tie-points in the real world.

We have defined a set of experiments –*small*, *medium* and *big*– with differently sized image sets where both algorithms / tools can be tested. We have executed the first two experiments where we have compared the bundle adjustment results of photogrammetric workflows without tie-points reduction with the results of workflows that include tie-points reduction. The results showed that by using either of the tools to decrease the number of tie-points in a certain factor, we achieve a decrement of roughly the same factor in the memory usage and in the processing time, while only having a decrement of the quality of the bundle adjustment results of a small factor. Concretely, the results of our *medium* experiment yielded that by decreasing the number of tie-points in around 95%, the memory usage and the processing time of the bundle adjustment decreased in around 90%. With that tie-points reduction, the bundle adjustment residuals increased around 14% using *RedTieP* and 7% using *OriRedTieP*, and the GCPs errors increased 3% using *RedTieP* and 27% using *OriRedTieP*.

Regarding future work, we will assess the effect of the tie-points reduction in a larger image set with the *big* experiment. Currently, the effect on the quality of the dense point cloud when using a reduced set of tie-points is done using the bundle adjustment residuals and the GCPs/CPs errors. Other quality assessment metrics will be explored. For example, comparing Digital Elevation Models (DEMs) generated from the point clouds in both situation (with and without tie-points reduction).

In each of the tasks executed in *RedTieP*, we only consider image pairs where the master image is present. This has the effect explained in Subsection 3.1.1, step 4. In future work, we will profile the exact effect of adding all the valid image pairs.

The errors in GCPs and CPs when using *OriRedTieP* are too large compared to the results obtained when using *RedTieP*. This is unexpected since *OriRedTieP* uses, in principle, more information that *RedTieP* to perform the tie-points reduction (it uses the 3D estimated position of the tie-points). In the *Tapas* residuals the expected behavior is observed, and *OriRedTieP* results are better, but this is not the case in the *GCPBascule* results. This has to be further investigated.

This report has focused on dealing with the limitations of the estimation of camera positions and orientations and of calibration parameters. It has done it by presenting the tie-points reduction algorithms. However, for the other steps of the basic photogrammetric workflow – tie-point detection and dense-matching point cloud generation– we have also identified that by using distributed computing solutions it is possible to tackle the processing of large image sets. We have implemented solutions and these are presented in a paper work-in-progress.

6 References

[1] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M. Seitz, and Richard Szeliski. Building rome in a day. *Commun. ACM*, 54(10):105–112, October 2011.

[2] Henri Astre. Sfmtoolkit, 2010.

[3] Emmanuel P. Baltsavias. A comparison between photogrammetry and laser scanning. {*ISPRS*} *Journal of Photogrammetry and Remote Sensing*, 54(23):83 – 94, 1999.

[4] Martin Byröd and Karl Åström. Bundle adjustment using conjugate gradients with multiscale preconditioning, 2009.

[5] Gianpaolo Conte, Piotr Rudol, and Patrick Doherty. Evaluation of a light-weight lidar and a photogrammetric system for unmanned airborne mapping applications. *PFG Photogrammetrie, Fernerkundung, Geoinformation*, 2014(4):287–298, 08 2014.

[6] P. A. Davis, S. N. Mietz, K. A. Kohl, M. R. Rosiek, F. M. Gonzales, M. F. Manone, J. E. Hazel, and M. A. Kaplinski. Evaluation of lidar and photogrammetry for monitoring volume changes in riparian resources within the grand canyon, arizona. In *Integrating Remote Sensing at the Global, Regional and Local Scale. Pecora 15/Land Satellite Information IV Conference, American Society for Photogrammetry and Remote Sensing*, 2002.

[7] Frank Dellaert, Steven M. Seitz, Charles E. Thorpe, and Sebastian Thrun. Structure from Motion without Correspondence. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:2557–564 vol.2, 2000.

[8] Niels Drost, Jurriaan H. Spaaks, and Jason Maassen. structure-from-motion: 1.0.0, February 2016.

[9] Jan-Michael Frahm, Pierre Fite-Georgel, David Gallup, Tim Johnson, Rahul Raguram, Changchang Wu, Yi-Hung Jen, Enrique Dunn, Brian Clipp, Svetlana Lazebnik, and Marc Pollefeys. Building rome on a cloudless day. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 368–381, Berlin, Heidelberg, 2010. Springer-Verlag.

[10] Jean-Michel Friedt. Photogrammetric 3d structure reconstruction using micmac. In *13th International Circumpolar Remote Sensing Symposium*, 2014.

[11] Simon Fuhrmann, Fabian Langguth, and Michael Goesele. Mve-a multiview reconstruction environment. In *Proceedings of the Eurographics Workshop on Graphics and Cultural Heritage (GCH)*, volume 6, page 8, 2014.

[12] Alejandro Lorenzo Gil, Laia Nez-Casillas, Martin Isenburg, Alfonso Alonso Benito, Jos Julio Rodrigo Bello, and Manuel Arbelo. A comparison between lidar and photogrammetry digital terrain models in a forest area on tenerife island. *Canadian Journal of Remote Sensing*, 39(05):396–409, 2013.

[13] Klaus Häming and Gabriele Peters. The structure-from-motion reconstruction pipeline - a survey with focus on short image sequences. *Kybernetika*, 46(5):926–937, 2010.

[14] Johan Hidding and Willem van Hage. Noodles: a parallel programming model for python, 2016.

[15] Steven Holmes, Gabe Sibley, Georg Klein, and David W Murray. A relative frame representation for fixed-time bundle adjustment in sfm. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2264–2269. IEEE, 2009.

[16] Yekeun Jeong, David Nister, Drew Steedly, Richard Szeliski, and In-So Kweon. Pushing the envelope of modern methods for bundle adjustment. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(8):1605–1617, 2012.

[17] Bryan Klingner, David Martin, and James Roseborough. Street view motion-from-structure-from-motion. In *Proceedings of the International Conference on Computer Vision*, 2013.

[18] Manolis IA Lourakis and Antonis A Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)*, 36(1):2, 2009.

[19] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[20] Oscar Martinez-Rubi, Maurice de Kleijn, Stefan Verhoeven, Niels Drost, Jisk Attema, Maarten van Meersbergen, Rob van Nieuwpoort, Rens de Hond, Eduardo Dias, and Pjotr Svetachov. Using modular 3d digital earth applications based on point clouds for the study of complex sites. *International Journal of Digital Earth*, 0(0):1–18, 0.

[21] Helmut Mayer. *Pattern Recognition: 36th German Conference, GCPR 2014, Münster, Germany, September 2-5, 2014, Proceedings*, chapter Efficient Hierarchical Triplet Merging for Camera Pose Estimation, pages 399–409. Springer International Publishing, Cham, 2014.

[22] Francesco Nex and Fabio Remondino. Uav for 3d mapping applications: a review. *Applied Geomatics*, 6(1):1–15, 2013.

[23] Kai Ni, Drew Steedly, and Frank Dellaert. Out-of-core bundle adjustment for large-scale 3d reconstruction. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[24] Marc Pierrot-Deseilligny. Micmac, 2016.

[25] Tomi Rosnell and Eija Honkavaara. Point cloud generation from aerial image data acquired by a quadrocopter type micro unmanned aerial vehicle and a digital still camera. *Sensors*, 12(1):453, 2012.

[26] F. Schaffalitzky and A. Zisserman. *Computer Vision — ECCV 2002: 7th European Conference on Computer Vision Copenhagen, Denmark, May 28–31, 2002 Proceedings, Part I*, chapter Multi-view Matching for Unordered Image Sets, or "How Do I Organize My Holiday Snaps?", pages 414–431. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[27] Dieter Sibley, Christopher Mei, Ian Reid, and Paul Newman. Adaptive relative bundle adjustment. In *Robotics: science and systems*, volume 32, page 33, 2009.

[28] Chester C Slama, Charles Theurer, Soren W Henriksen, et al. *Manual of photogrammetry.* Number Ed. 4. American Society of photogrammetry, 1980.

[29] Chris Sweeney. *Theia Multiview Geometry Library: Tutorial & Reference.* University of California Santa Barbara., 2016.

[30] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3057–3064. IEEE, 2011.