# Photogrammetry / SFM guides for the C-Astral Bramor PPX and other UAVs. [DRAFT]

**Author: Ciaran Robb, Earth Observation lab, DGES, Llandinam building, University of Aberystwyth.**

# Photogrammetry / SFM guides for the Bramor PPX and other drones. [DRAFT]

## Introduction

This document outlines workflows to process imagery from **U**nmanned **A**erial **S**ystem (UAS) to produce both ortho-imagery and Digital Surface Models (DSM) using Structure from Motion (SfM). Key text resources are linked throughout the document, use ctrl + click to navigate directly to the material. If a commercial package is preferred, please see the generic documentation for Agisoft Metashape (formerly Photoscan) for which a license is available. The project originally made regular use of the C-Astral Bramor PPX fixed wing platform and so occasionally refers to it, but the workflows are applicable to all systems.

### Software Material Overview

The material in this document increases in complexity as it goes on and for most users the first sections are most important as SfM processing can be accessed entirely from Graphical User Interfaces for either commercial or open source software. Three software sources are covered here and are:

- Agisoft Metashape (commercial)
- Open Drone Map (open source)
- MicMac (open source)

The recommended commercial software is Agisoft Metashape, of which QinetiQ have gained a license, through the related GEOM project. This provides a complete desktop-based GUI which is available as a floating single-use licence. A comprehensive manual is found here.

Both open source packages are available in GUI format and are integrated in WebODM. WebODM is a web browser-based SfM software that is installed via a dockerized container. The most user-friendly implementation of MicMac has recently become available (1/05/20 at the time of writing) as an integrated node of Open Drone Map as a web browser-based GUI found here. The MicMac QGIS plugin previously written is experimental and currently requires manual copying and pasting of files etc. and is not recommended but is retained for posterity. It also requires that you have compiled/installed MicMic with QT support first (see website). It is recommended you simply use the Open Drone Map MicMac node. Open Drone Map is covered here in its GUI form only with MicMac also covered in command line form thereafter.

For those who wish to look deeper into the underlying processes the later material contains use of BASH (Borne-Again-Shell) and the python programming language if you wish to use some of the open source libraries in fine grained detail**.** For the shell scripts, a unix-based OS is required, ideally IOS, ubuntu or similar. However a docker package is also available which makes this OS independent. The command-line based material is here to provide a background on SfM generally as was used in the earlier stages of the project, as well as affording a more detailed view on processing.

## Hardware Setup

Needless to say - the more processing power, the better! Ultimately, lots of CPU and GPU resource is the ideal scenario, though the importance of the latter is dependent to some degree on the number of the former. Sfm is process hungry but largely favours parallel processing during most stages. SSD disks are recommended for read-write times but HDDs are still sufficient. Processing datasets of 100s images will typically take hours on a modest machine (e.g. 4 CPU cores) and 1000s of photos will run into days.

The recommended use of GPU acceleration is most relevant to Agisoft Metashape which leverages GPU acceleration extensively/effectively. The use of HDDs is still viable of course. The open source libraries are dependent on CPU cores alone and so more of these are meritful if using them! Open Drone Map only uses CPU processing at the time of writing. MicMac has limited support of GPU acceleration but is problematic and not recommended.

# An outline of Structure from Motion processing

Orthorectification is the warping and resampling of raw camera imagery to an approximate representation of the area covered in reality (e.g. on the ground below). Digital surface models are image-based 2.5 dimensional representations of the earth's surface where each pixel value is (usually) an elevation value. Traditionally, DSMs are created through photogrammetry; where a combination of the displacement of common and ground control points (GCPs) are used to produce a DSM. Similar methods may be employed in 'true' 3D to resolve an object from oblique angles.

Structure from motion (SfM) is a semi-automated method that extracts 2.5/3-dimensional data from overlapping imagery using a combination of feature-matching between imagery, camera lens geometry and modelling camera position. This process is further enhanced if the camera position is known/recorded in a geographic projection. The basic premise is to match common features across overlapping images and model their relative orientation or pose in 3D space as illustrated below in Fig1a.

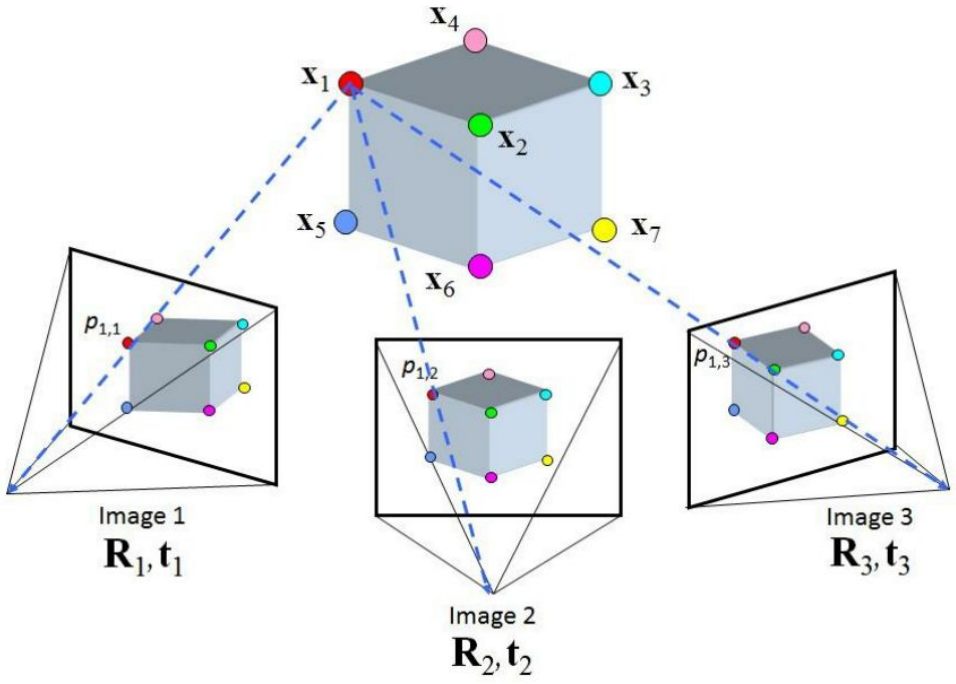



Fig 1a. Triangulation of image features to estimate 3D position. Once the relative orientation of images taken of the subject (the cube) are established, a feature can be triangulated in 3d space as vectors from the relative camera positions relative to the camera pose centres. The more overlapping photos and perspectives; the better the orientation.

This is of course, is a simplified explanation - the reality is a little more complex as are estimating position within certain bounds (essentially minimizing an error) and different geometries are used depending on the reconstruction problem. These vary as can be seen below in the figure taken from Rupnik et al. (2017).

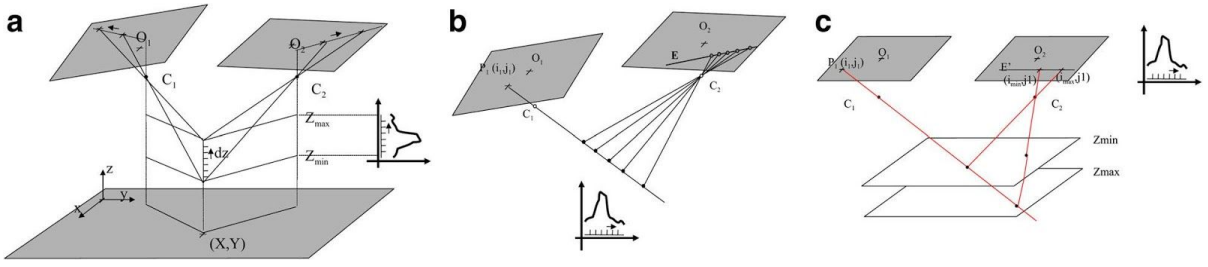



Fig 2. Reconstruction geometries - ground geometry(a), image geometry(b) without resampling and resampling to epipolar geometry(c).

The image pose and hence 3D reconstruction may be modelled in:

- Ground geometry (Fig 2a) , where a plane is assumed (i.e. the ground - the case with most nadir acquisitions).
- Image geometry, where the pose is discretized along the ray.

- Epipolar geometry (essentially image geometry, but resampled).

Ground geometry is usually the most efficient and intuitive approach for nadir/pseudo-nadir acquisitions as the reconstruction takes place in the geographic coordinate system Z-plane and is reliable for relatively smooth landscape structures and ortho-photo generation.

For image/epipolar geometries, a master and set of slave images are chosen and depth calculated in 3D space. This approach is better for complete 3D reconstruction of discrete/specific objects (e.g. a building) or where the surface is more complex (e.g. forest canopies, urban structures), but comes at the cost of longer processing time. The selection of master images can still be automated and the reconstruction can still be projected in geographical space nonetheless. The geometry choices are automated in the software documented and are provided for information purposes.

**Please see the myriad of online resources for in-depth explanations of these approaches.**

# Agisoft Metashape

## Processing chain overview

The breakdown for Metashape is broadly similar, though some aspects are automated. The entire workflow is accessed from the 'Workflow' menu in the figures (4th in from the left).

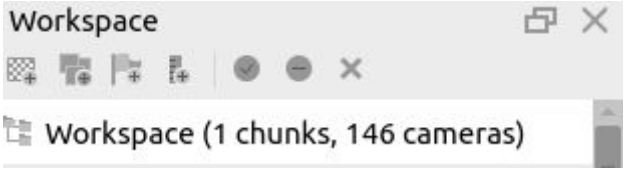
1. Load photos
2. Inspect / remove photos (usually unnecessary)
3. Align photos (equivalent to relative orientation, GPS bundle adjustment)
4. Dense point cloud production
5. Generate Mesh
6. Generate Tiles
7. DSM production
8. Ortho-mosaic production

A brief summary is now given, but it is rare that any more than this is required when using this software. Each step has a couple of 'Advanced options', but these are rarely needed.

## Agisoft Metashape workflow

### Adding the photo dataset

Photos are loaded via Workflow > Add Photos appearing in the workspace pane on the left.




### Align photos

Next, click on Workflow > Align photos, which will orient the photos and upon completion will display the camera poses and sparse points as seen in Fig 5

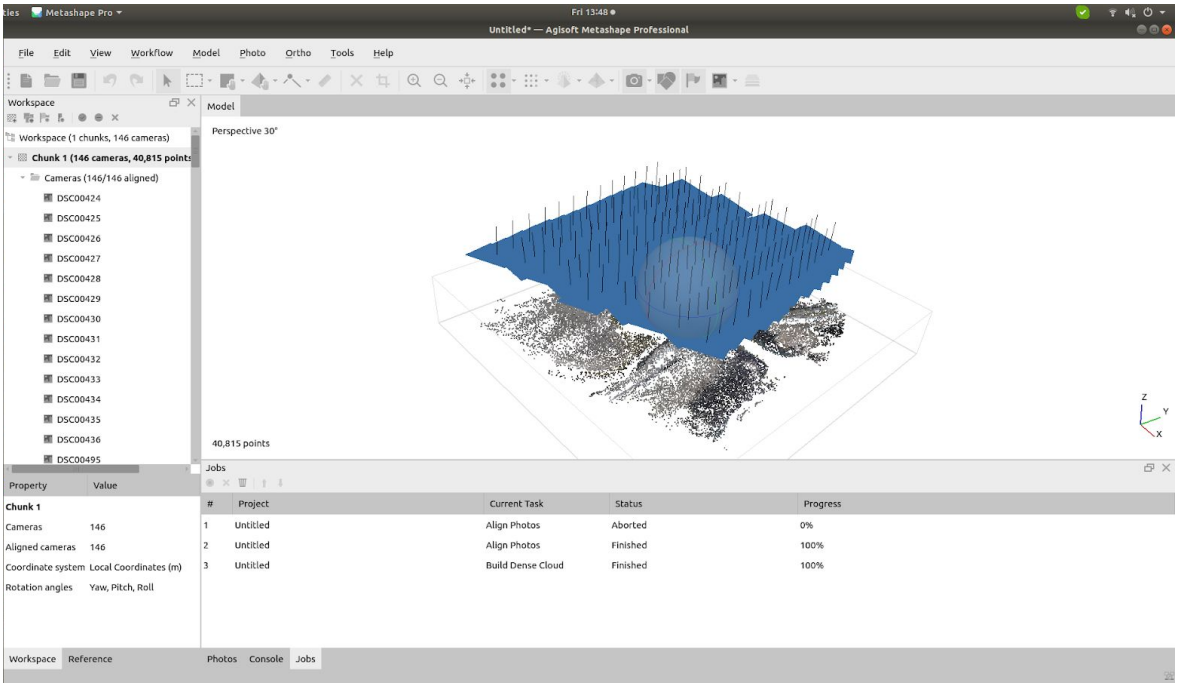



Fig 5: Sparse cloud in Metashape with modelled camera poses above.

## Dense Cloud

Next click Workflow  > Dense Cloud and after processing the dense cloud will appear in the pain. Double click to view.

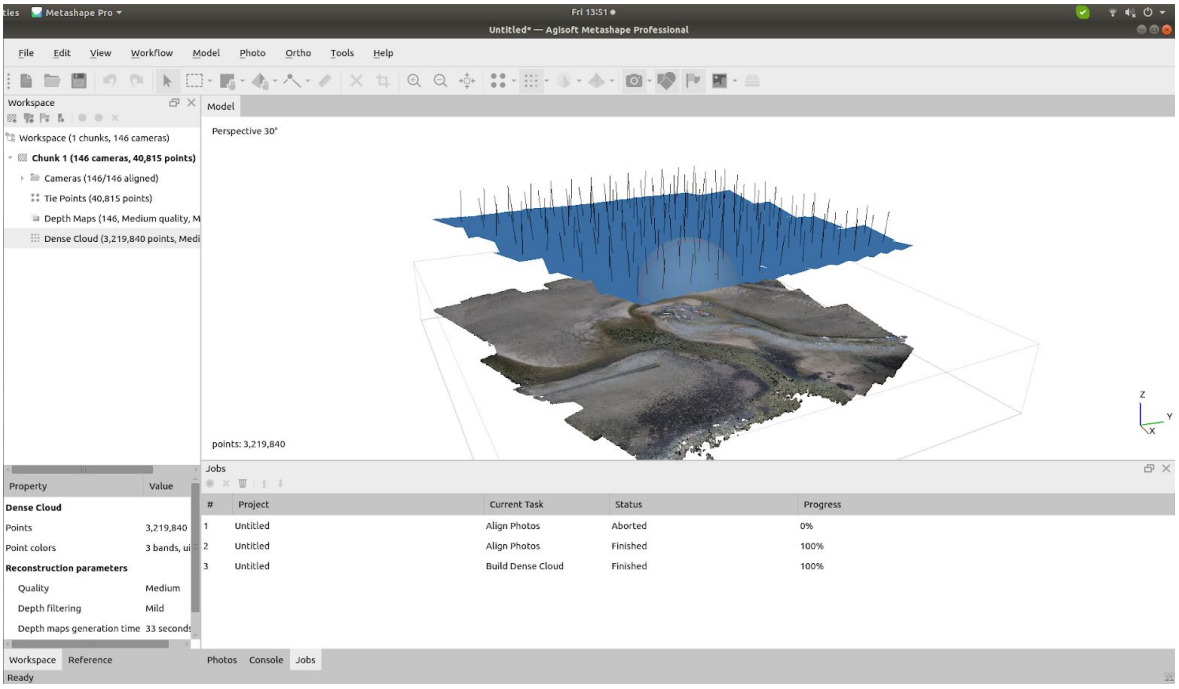



Fig 6: Dense cloud in Metashape with modelled camera poses above.

## Outputs

The same general procedure  is followed in sequence for the following items

- Workflow > Mesh
- Workflow > Texture
- Workflow > Tiled Model
- Workflow > DEM
- Workflow > Orthomosaic

Finally, File > Export provides all the required options for exporting data. A useful option here is the export File > Generate Report which provides a good information resource on the quality of the outputs in PDF format. This is included in the shared folder.

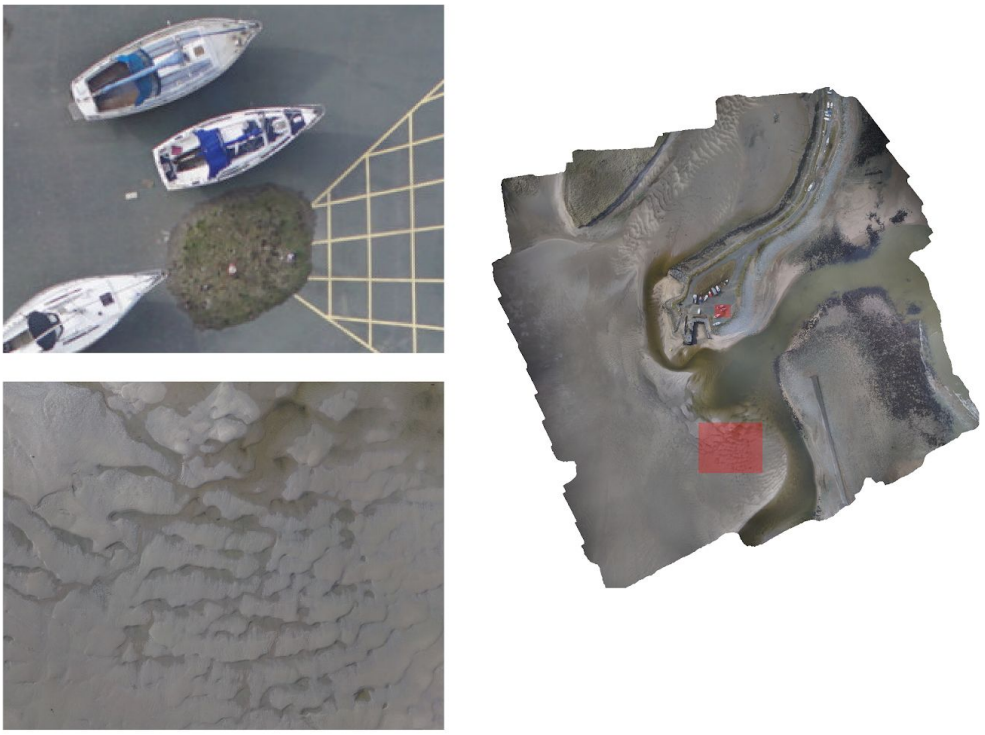


Fig 7: Orthomosaic using Metashape, with insets marked in red.

For multispectral imagery derived from the Micasense Red-edge, such as that mentioned later in the text, the process is largely the same but there are differences at the beginning covered in the how-to courtesy of the Agisoft website Micasense.pdf in identifying the panel images etc. As QinetiQ does not possess such a camera, this is unlikely to be encountered.

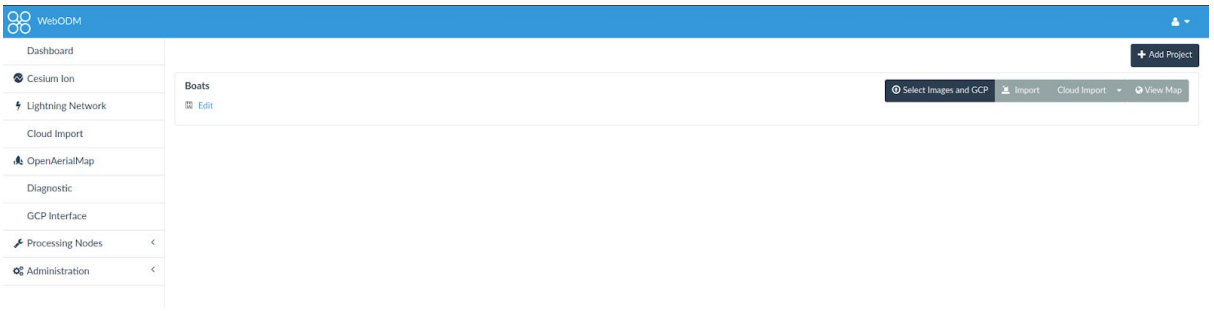
## Open Drone Map

Open Drone Map has both command line and GUI options and provides comprehensive documentation here hence any detailed account is superfluous. Only the GUI is outlined here, which for most cases is all that is required. WebODM can be installed using the instructions here and opens in a web browser. Once you have it installed and opened - it is easiest to bookmark the page in your web browser for future use (it will be something like http://localhost:8000/dashboard/).

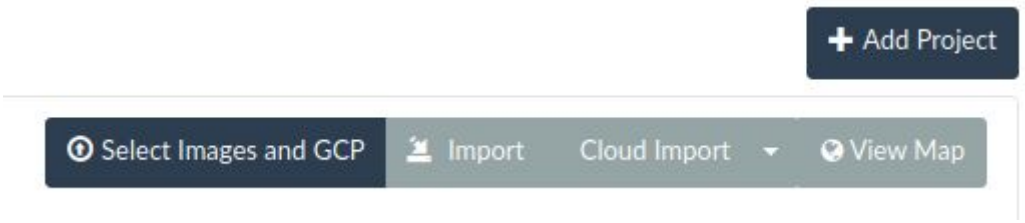
## Open Drone Map workflow

You will be confronted by an interface like the one below.

## Add Photos & GCPs




Click the top right button “add project” then fill in the fields (seen below in detail).




With the newly created project click add “select images and gcp”, in which you can navigate to the folder where your photos are stored. Once selected, leaving the default options and click review.

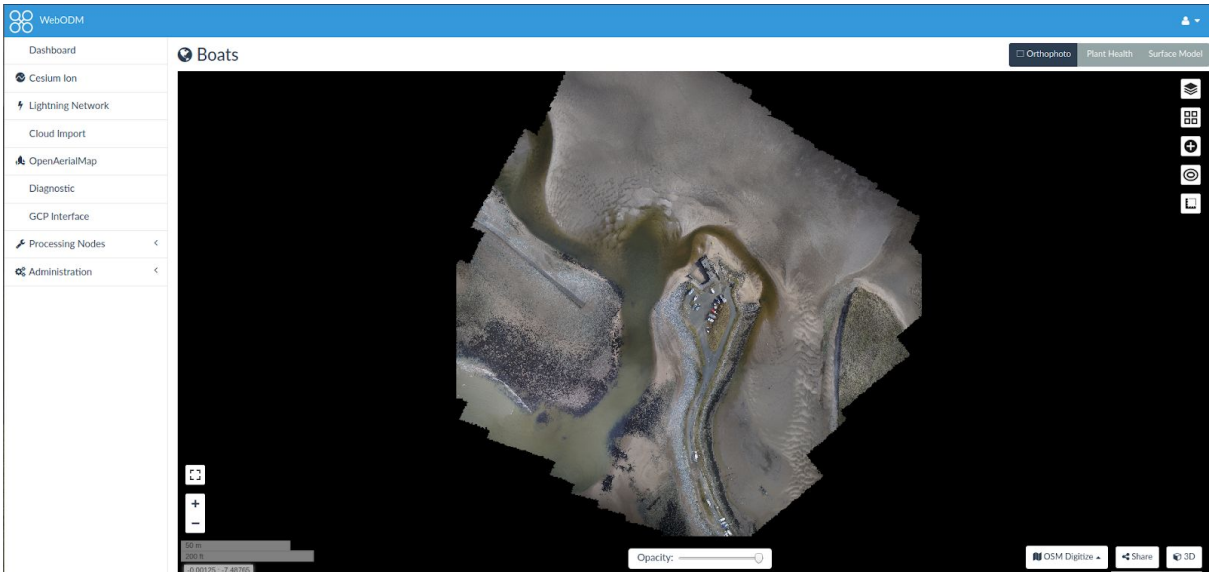
## Review & process

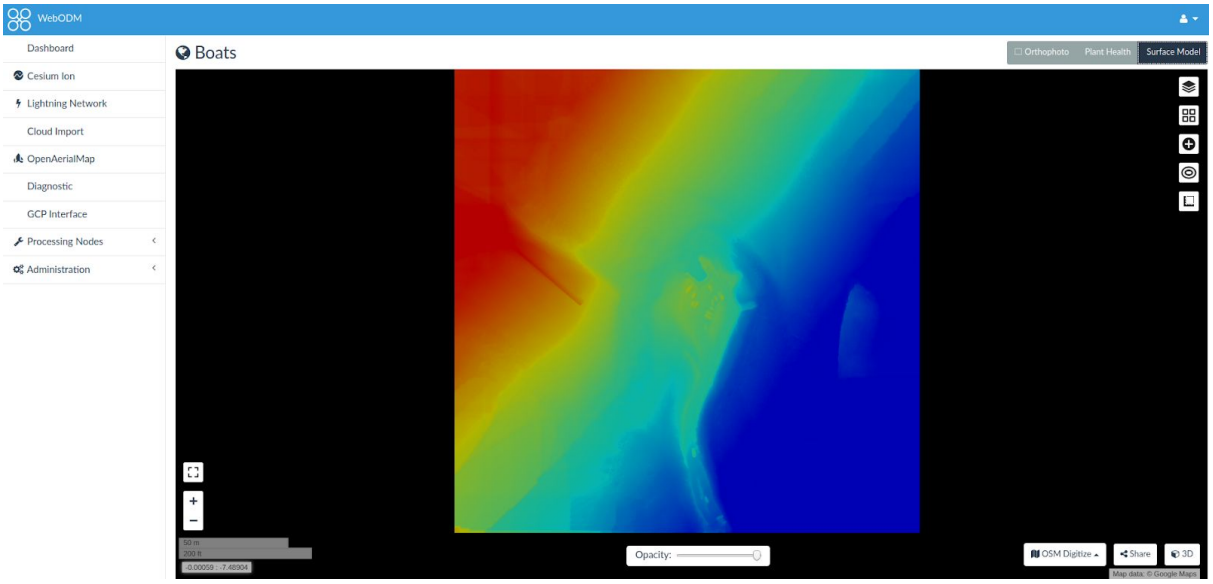

The review button will change to the following - click on start processing.

Click on the process (the blue Task of....  link on the left of the centre pane), then click task output to on on the right to view progress. You could be in for a wait.....


## Outputs

Click on “view map” where you will be able to toggle between ortho mosaic and DSM as seen below.






# MicMac

MicMac is command line based, although there is simple QT-based GUI help, optionally compiled (even this still requires the command line to call the menus). Operations are fine grained to produce ortho-photos, point clouds and DSMs. MicMac consists of a "simplified" high level command line interface and a low level "expert" xml-based command line interface. Despite the early promise, the "simplified" interface is powerful and covers more than is required for user SFM processing. A list of the commands with explanations for most of them is available on the website. Most of the commands are named after drinks/cocktails etc, which does not help much with understanding what they do, so probably just as well there is a good website!

A good pdf is now available in English (also provided in the github folder) with comprehensive descriptions of functionality as well as tutorials. Of particular relevance to the Bramor PPX is a tutorial available online using a fixed wing platform.

A python library by the author named pycmac wraps functionality in Micmac and may be preferred if familiar with python, but is a compromise of the underlying command line. Another named MicaMac is also available here. The sections that follow utilise the Micmac command line directly and are more complex (as all datasets have their quirks) but this is ultimately the most reliable way to use the program. The command line provides the most fine-grained control of the process and debugging.

# MicMac

## Processing chain overview

The routines herein largely assume nadir or pseudo-nadir imagery. Whilst the routines will work on most data provided you possess the lens and sensor info, particular reference is made to the Bramor PPX as this is the primary platform. **It is best practice to ensure lens, sensor and GNSS information is written to the image EXIF for ease of processing**. In the event this is produced separately (not recommended), a Python based script has been written, to convert the PPX GNSS data into a compatible format for MicMac. The python library also provides this functionality. The camera lens information must also be added to the MicMac xml database. Functionality is available in MicMac to embed this information in the JPG's or alternatively a local XML file may be included in the working

folder (provided - LocalChauntierDescriptor.xml). **To avoid this, ensure the GNSS data etc. is embedded in the imagery first using the C-3P (or equivalent).**

The basic processing chain for MicMac and PPX imagery is as follows:

1. Create a list of images with respective coordinates embedded in the imagery
2. Use the coordinates to make a list of overlapping image pairs to reduce overall processing time
3. Perform tie point detection with image pairs (feature/key-point detection - a variety of algorithms are possible here – the trade-off is time vs detail and accuracy)
4. Compute relative orientations (or pose) of images
5. Use tie points and GNSS coordinates to improve modelled orientations
6. Compute depth map(s), from which create DSM and ortho-photos
7. Mosaic the ortho-photos
8. Create a coloured dense point cloud file for visualisation or mesh application (Optional)

The amount of features computed initially dictates the overall time required. As will likely be the case with every application, the trade-off is generally increased detail/accuracy = greater processing burden. MicMac supports GPU (Graphics Processing Unit) acceleration which reduces processing time for smaller amounts of images, but in its current state will only exploit a single GPU and use is only recommended when you possess few cores. Ultimately many CPUs are a more important consideration. Figures 2-6 depict intermediate to final results of the data processing.

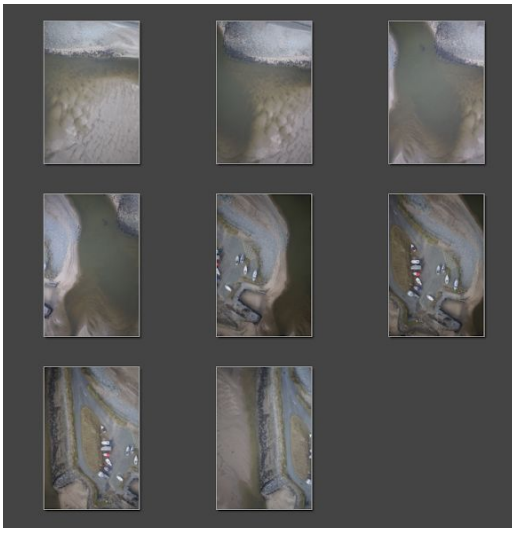


Fig 2: Collection subset of raw photos

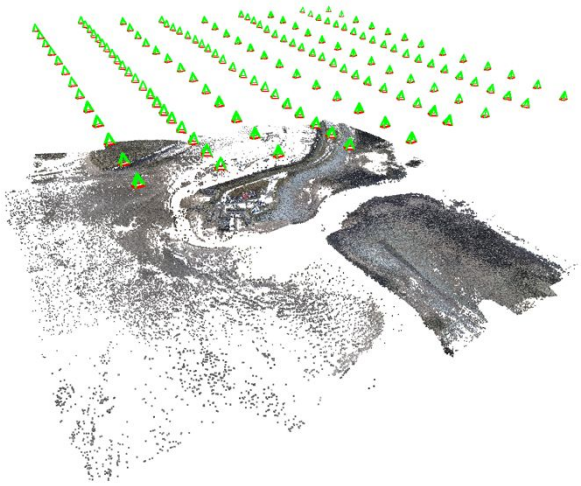


Fig 3: Camera position superimposed over sparse point cloud (an intermediate result of the pipeline) where each point is coloured according to corresponding pixel value. This is useful for evaluating the quality of orientation.

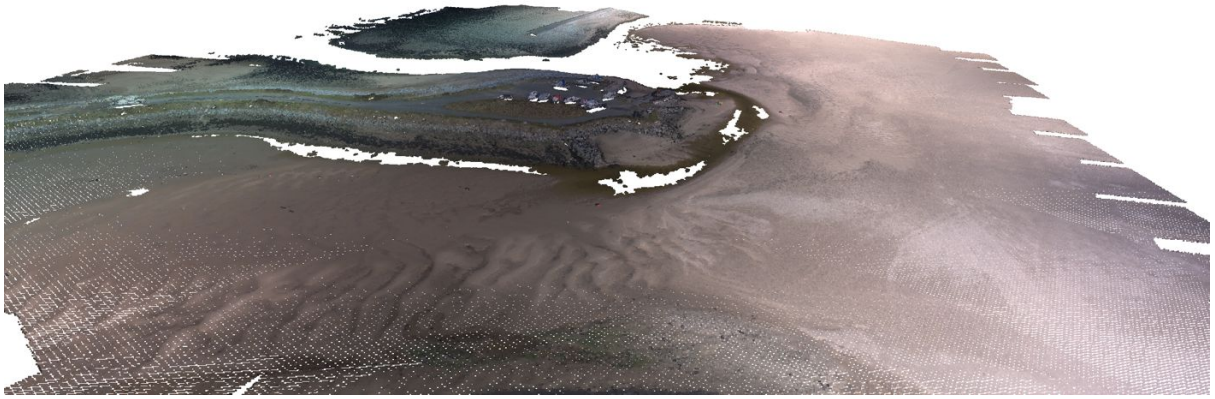


Fig 4: Perspective on the dense point cloud, where each point is coloured according to corresponding pixel value.

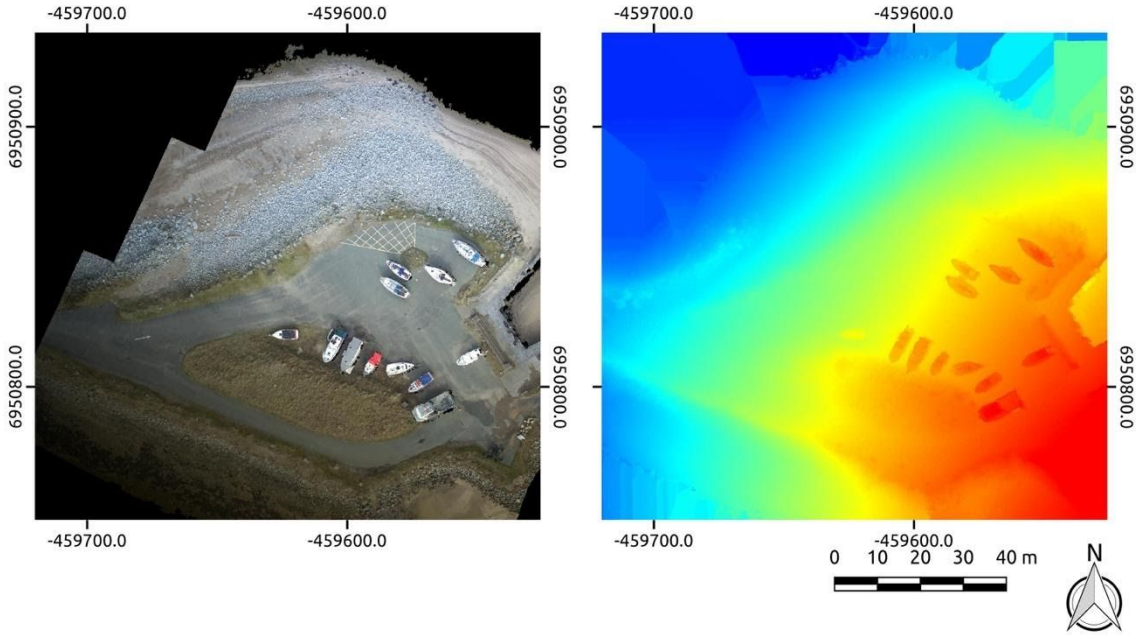
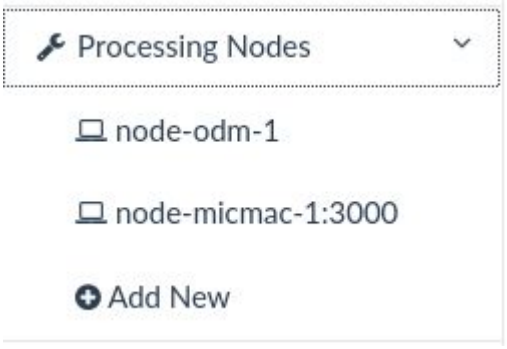



Fig 5: A subset of the final orthophoto and DSM side by side. Warmer colours indicate greater elevation values.

## Node MicMac Open Drone Map workflow

Fortunately, WebODM integrates MicMac as a processing engine and using it this way provides the most accessible (and possibly reliable) way of using it. After installing WebODM as per the instruction linked earlier. Run the docker commands as shown in this link. After which the steps are near-identical to the WebODM workflow, but you must choose the node-micmac backend for processing. To check it is installed click Processing Nodes on the left pane.




If you have node-micmac you may proceed. Follow the same procedure as with WebODM but prior to clicking “review” select the node-micmac processing node.


You may monitor the process the same as before and upon completion click view map on the top right of the centre pane where DSM and ortho mosaic may be viewed.

## MicMac command line workflow

The command line SfM workflow is detailed below. This should be applicable to most datasets, but does not guarantee success!

### Formatting

- Explanatory text in black
- Commands in red
- Links in blue
- Extra info in orange
- Anything in a black box is the alternative (not recommended) .csv based workflow

This workflow is based on a unix platform (either GNU/linux variants or MacOS). Either platform is better suited to scientific computing (particularly GNU/Linux) and open source software. If you insist on using Windows,instructions on installation are here and scripts may be run via Cygwin or better still install in a docker container. A brief account is given to the now deprecated QGIS plugin.

## QGIS plugin (deprecated)

For those unfamiliar with the command line a QGIS plugin has been written with a simple button panel to all the commands detailed below and is left here for posterity as it **is now (1/05/20) rendered somewhat less useful by the Open Drone Map web GUI found here, which is easier to install and less complicated to process data.** The plugin is accessed at the plugins drop-down menu. The plugin is installed by copying the plugin folder into the QGIS plugins folder (the location of this will vary) and adding the MicMac bin folder to your QGIS path.

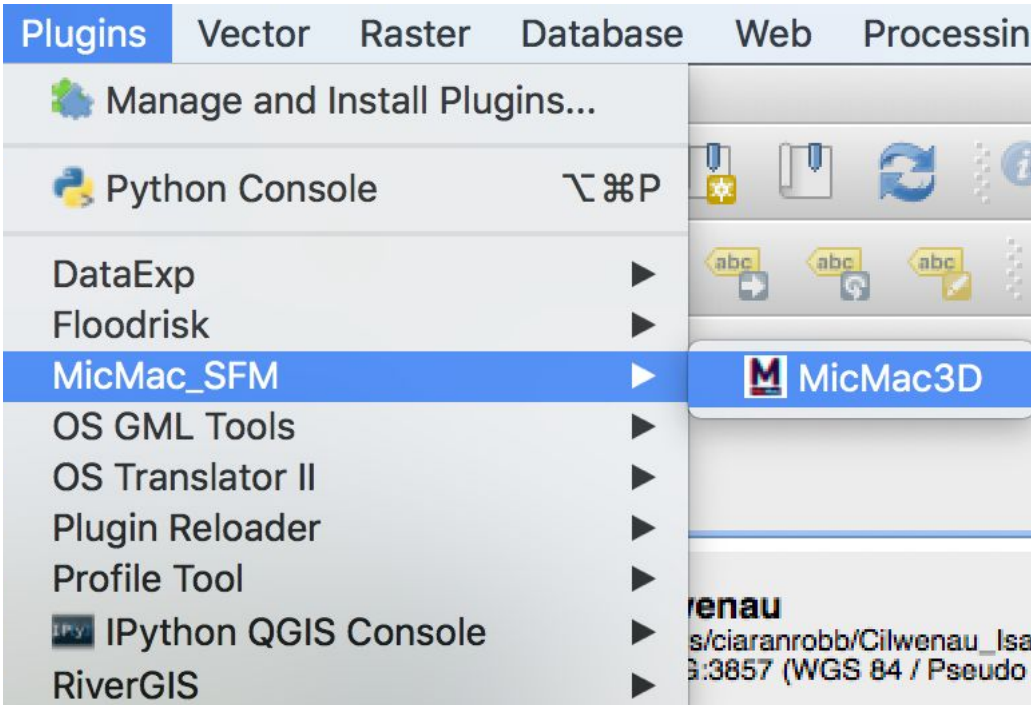



Fig 8: Menu access to the QGIS plugin.

The panel contains all the commands used in the command breakdown.

Upon clicking MicMac3D, a button panel will appear which links to all the commands detailed below, with an intuitive English title of the function with the MicMac command name in brackets. Using these, you can follow the work flow below.

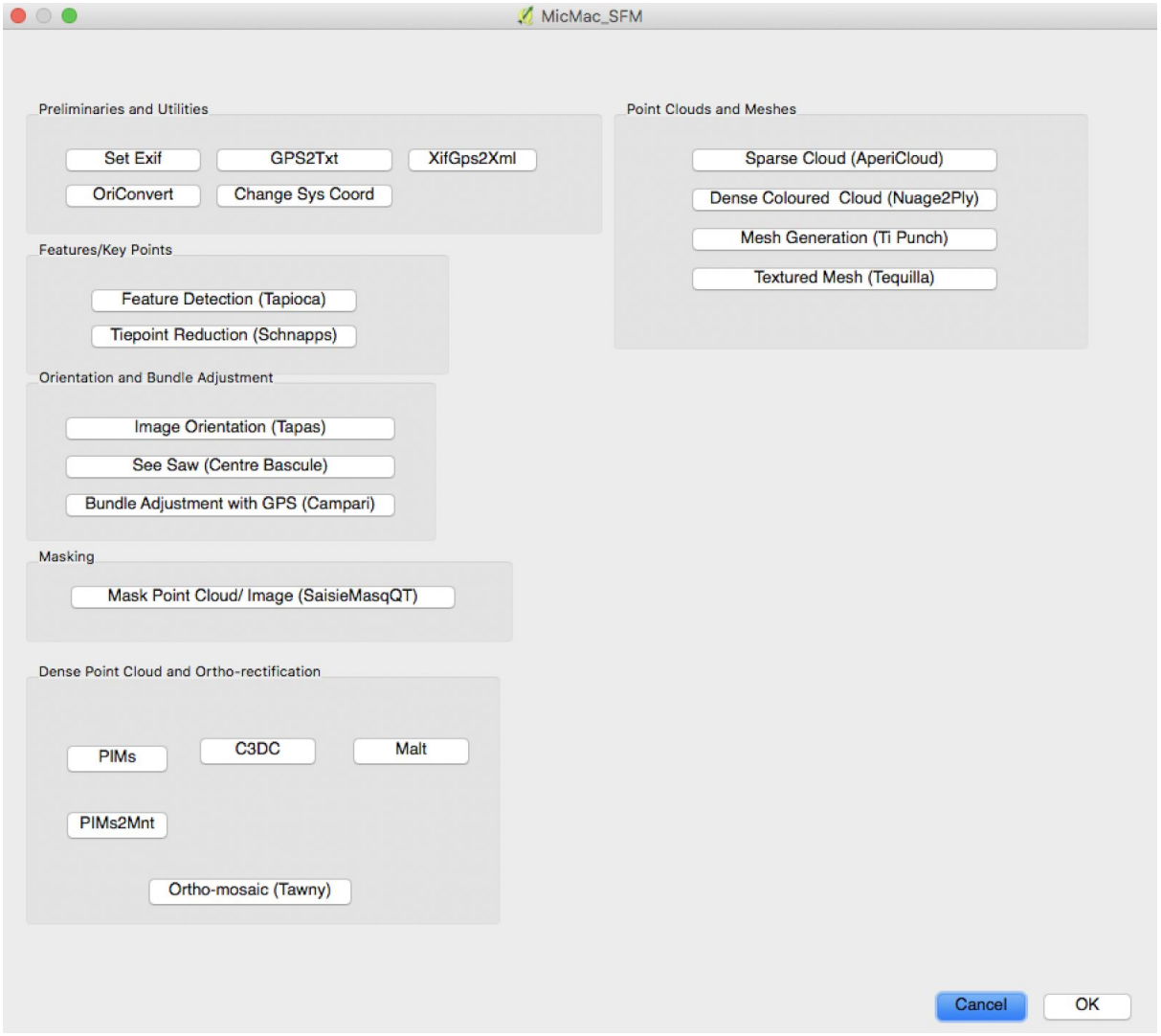



Fig 9: The QGIS plugin panel.

By clicking on the buttons, the MicMac QT menu is summoned automatically (see below).

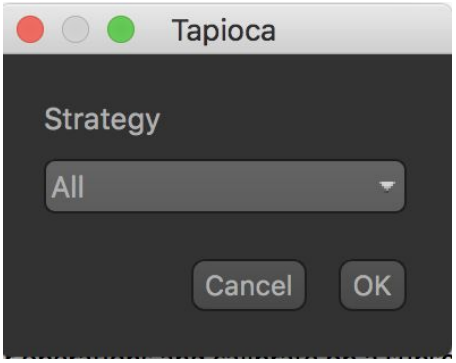



Fig 10: MicMac QT command menu.

## Individual Command break down

As previously stated, prior to processing, two steps may be necessary. The first is to input the lens info if it does not exist in the EXIF of the JPGs. Only do this if you encounter an error in one of the first commands – if this is new PPX imagery, you probably will! If this is a different drone type (e.g. DJI variant), they are normally embedded with the correct tags so is merely here for project completeness.

Here the keyword arguments are:

**Please note this command is using lens parameters for the C-Astral Bramor PPX! Only use for this platform. The PPX camera is a Sony A/ILCE-6000 with a fixed 30mm focal length.**

F35 = 35mm focal length equivalent

F = focal length

Cam = the camera name

```
mm3d SetEXIF "*JPG" F35=45 F=30 Cam=ILCE-6000
```

***The use of a glob style wildcard "*JPG" is common to most commands and indicates all the images within the working directory.*** *If a specific subset/pattern is required, it must be written like this "1.JPG|2.JPG|3.JPG". It is easier to have only the imagery required in your working folder. (Alternatively there is an option in the pycmac python lib to read this from a csv).*

***As it is commonly used, the DJI Phantom 4 focal info is:***

```
mm3d SetEXIF "*JPG" F35=20 F=3.6 Cam=DJI
```

***As a further aside - this command will list the focal length info of present in the exif.***

```
exiftool *.JPG | grep "Focal Length"
```

If a separate csv is used, it is very important to set up a local coordinate xml file (SysCoRTL.xml) with the coordinates of the centre image with its x,y,z entered (see the grand leez demo). This allows MicMac to work in a local Euclidean space for operations and calibrate on a subset of images (See the command list for the first script) prior to later converting results to a cartographic space. This is done automatically by the scripts provided later as well as the entire SFM pipeline outputting ortho-mosaic, DSM (more on which later). For the moment, a complete breakdown is presented so as to understand each step of the process.

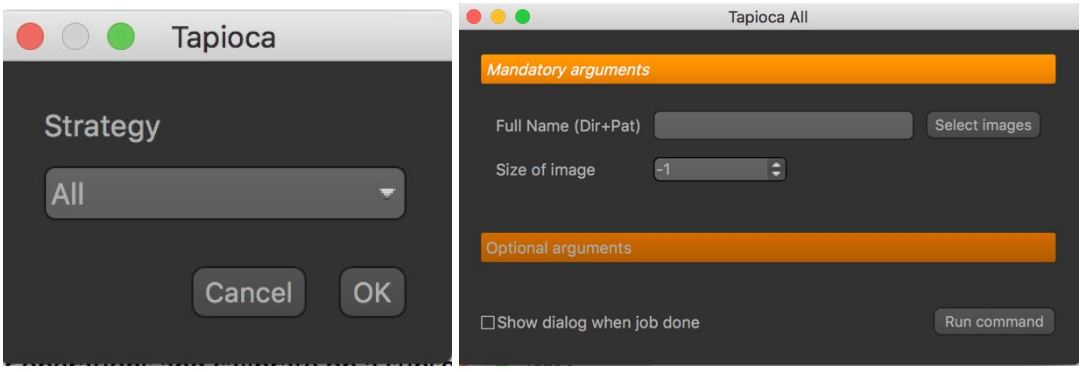
**GUI alternatives from the command line**

By prefixing every command with a '**v**', menus appear to execute the function rather than using commands. Each stage will have a picture of the corresponding GUI menu to the command line. These are also accessible via the QGIS plugin eliminating any command line use.

e.g.

```
mm3d vTapioca
```

A user interface menu for most functions will appear (This is the compiled QT menu interface internal to MicMac). This is useful if you prefer a GUI to command line arguments. The command line arguments are found on the main menu and any optional ones in the optional arguments tab. The layout is the same for each command.




As can be seen above, a menu interface offers drop down options in place of typing commands.

## Tie point detection

Prior to tie point detection, some extraction of photo meta-data is required, such as the GPS coordinates/flight info etc. This is required at various stages during processing and can make the tie point detection more efficient (particularly for a Nadir grid pattern of acquisition). This is a keyword argument choice in the scripts provided later and is available as menus via the QGIS plugin.

**For imagery with embedded information (recommended):**

```
mm3d XifGps2Txt .*JPG
```

This command extracts the GNSS data from the images and converts it to a xml orientation folder (Ori-RAWGNSS), also create a good RTL (Local Radial Tangential) system.

```
mm3d XifGps2Xml .*JPG RAWGNSS_N
```

This command creates pairs of images for tie-point detection based on the GPS data. This step defines which images 'see each other' during tie-point detection. This avoids an exhaustive search for every single image. The default distance from an image centre is 50m (specified here DN=50 so the arg. can be seen), so due consideration should be taken to the original overlap/flying altitude when defining g this parameter.

```
mm3d OriConvert "#F=N X Y Z" GpsCoordinatesFromEXIF.txt RAWGNSS_N
ChSys=DegreeWGS84@RTLFromEXIF.xml MTD1=1
NameCple=FileImagesNeighbour.xml CalcV=1 DN=50
```

**Alternatively, using a separate csv file with metadata (not recommended unless familiar).**

```
mm3d OriConvert OriTxtInFile mycsvfile.csv RAWGNSS_N
ChSys=DegreeWGS84@SysUTM.xml MTD1=1 NameCple=FileImagesNeighbour.xml
CalcV=1
```

A python script to create a relative coordinate system xml file (provided on github site and integral to all scripts).

```
sysCort_make.py -csv mycsvfile.csv
```

Tie-point detection is executed with the function Tapioca. A typical command using the full resolution of available imagery is as follows with –1 denoting full resolution:

```
mm3d Tapioca File FileImagesNeighbour.xml -1
```

This command will be lengthy in processing time particularly with a lot of images at full resolution. **Consequently, it is better to downsize, particularly given there is little gain in full resolution over 1/3 to ½.** Hence the following command resizes imagery to a long axis of 2000 pixels:

```
mm3d Tapioca File FileImagesNeighbour.xml -2000
```

*However – in areas where there is high detail such as forest canopies keeping the resize closer to actual could be preferable. Additionally, resizing the imagery permanently prior to processing reduces processing time at this stage **and** at dense matching, so is a worthy consideration (included in scripts*

*section later. By adding the character @SFS at the end of the Tapioca command the images are 'high-passed' to aid key point detection. The scripts provided automate image re-sizing.*

For exhaustive tie point detection use the 'All' keyword (not recommended unless under special circumstances).

```
mm3d Tapioca All .*JPG -2000
```

*Bear in mind this will take much longer to process and is generally most useful if imagery has been collected at multiple angles and/or in a haphazard fashion.*

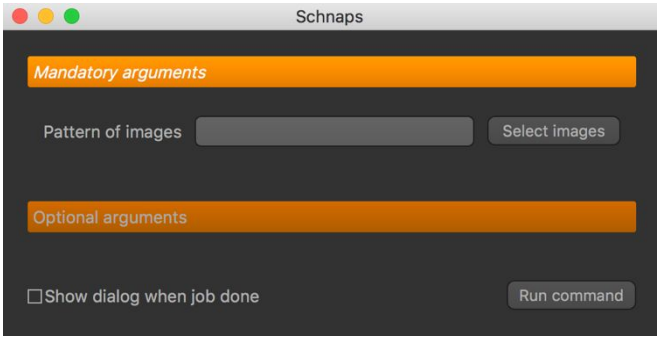
The resizing of imagery is an important aspect of SFM and as noted above, there appears to be rapidly diminishing returns at higher values. A downsize reduces the computational time spent searching for tie-points between images (SIFT++). There is a loss of detail as images are downsized further. SIFT is parallelised in MicMac (It utilises the Sift++ library only thinly wrapped – see the MicMac manual for alternatives in the "Advanced Matching" section).

If processing time and data volume are limiting factors, is also worth considering re-sizing the imagery permanently for the later production of dense point cloud and DSM. This can be achieved via Image Magick (installed as a dependency of MicMac) with the following command:

```
mogrify -resize 2000 *.JPG
```

Once completed, the tie point information is stored in a 'Homol' (short for homologous points) folder in xml format. At this point, the Schnapps tool can be used to create a reduced tie points collection prior to running Tapas and is recommended (this is integral to the complete workflow scripts). This function also removes images with no tie points, useful for avoiding errors being returned when orientations are calculated. Results are saved in a folder named "Homol_mini" by default.

```
mm3d Schnaps ".*JPG" VeryStrict=1 MoveBadImgs=1
```




The Schnaps command does not guarantee a lack of errors later, but helps nonetheless.

## Image Orientation

A range of options are available for the computing of relative orientations (or pose) of images, with the Fraser model offering the most sophisticated approach, again at the cost of increased processing time. A calibration run on a subset of images can be run to save processing time known as Internal Orientation Parameters (IOP), but for the sake of completeness the full dataset can be processed with the following:

```
mm3d Tapas Fraser ".*JPG" Out=All-Rel SH=_mini
```

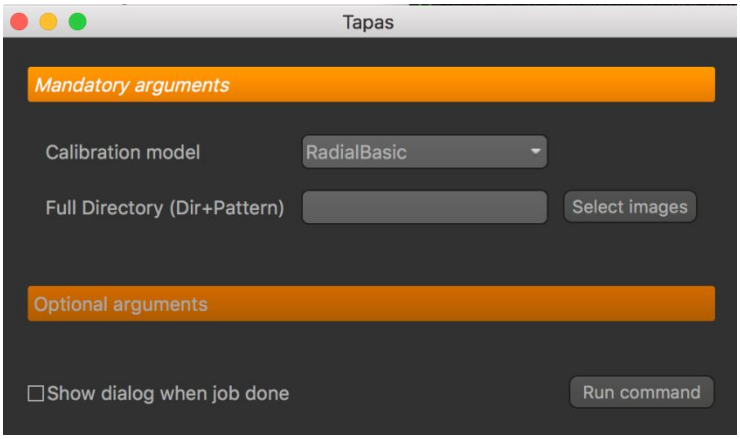
Or a simpler model (there are several to choose from, see website and docs):

Where the SH parameter denotes reduced tie point set created with the Schnaps command.

*In the unlikely event an error still occurs running Tapas after Schnaps, remove the images listed at the end of the error message and run Tapas again.*

*A calibration subset can be useful when the dataset is very large as this provides an initialisation for the global operation. A central subset has been used as per online examples and literature. If the calibration fails, a larger or different image block may be necessary.*

.



The RMSE (residual) for the bundle adjustment (orientation) is displayed at the end of the output like this:

| | Residual = 0.552963 ;; Evol, Moy=3.85781e-07 ,Max=1.19406e-06

| | Worst, Res 0.841739 for R0040632.JPG, Perc 96.5986 for R0040472.JPG

| | Cond , Aver 3.73611 Max 28.3138 Prop>100 0

Further bundle adjustment (orientation) using the GNSS data can be executed using Campari following a coordinate change with CentreBascule as follows (the commands may vary depending on the manner in which the GNSS values were initially stored). The commands below will work with exif-based workflows.

```
mm3d CenterBascule .*JPG Arbitrary RAWGNSS_N Ground_Init_RTL
mm3d Campari .*JPG Ground_Init_RTL Ground_UTM EmGPS=[RAWGNSS_N,1]
AllFree=1 SH=_mini
```

As with Tapas, the final error will be displayed at the end of the process.

## Dense cloud and ortho generation

For dense cloud and ortho-generation there is a choice of 2 algorithms.

**Malt:**

This is the original dense matching command, recommended for ortho-mosaics. Malt utilises a multi-view stereo approach, matching the imagery in sequence in ground geometry.

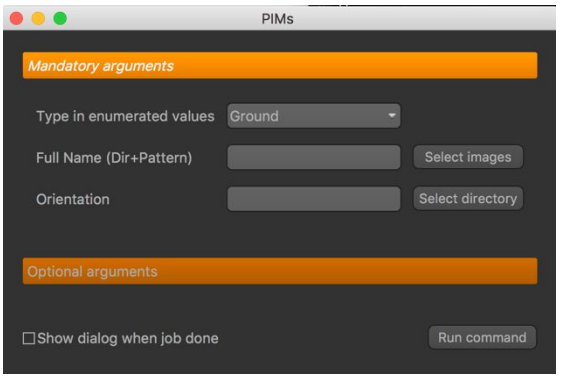
**PIMs:**

Per-image-matching (PIMs), is a more automated command with some arguments specific to data types such as Forest etc. PIMS seems to be better for DSMs of forested areas, but may omit areas of low incidence in ortho-photo production. It operates in epipolar geometry during processing for Forest & Statue modes and it is these modes where it is most useful.

If the PIMs command is executed, it performs the **p**er-**i**mage-**m**atching for the pairs (see the manual for an in-depth description of the algorithms). This produces folders containing the individual depth maps. The algorithm choice BigMac will suffice in most cases and Forest (unsurprisingly) for forested imagery. The Forest algorithm matches pairs of images in epipolar geometry unlike the Malt or other PIMs options.

```
mm3d Pims Forest ".*JPG" Ground_UTM SH=_mini
```

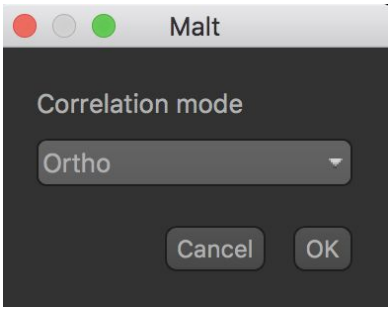
*The SH parameter is not mandatory, just to use the filtered key point pairs from earlier produced by the Schnapps command. The scripts do this automatically and rename the original file.*



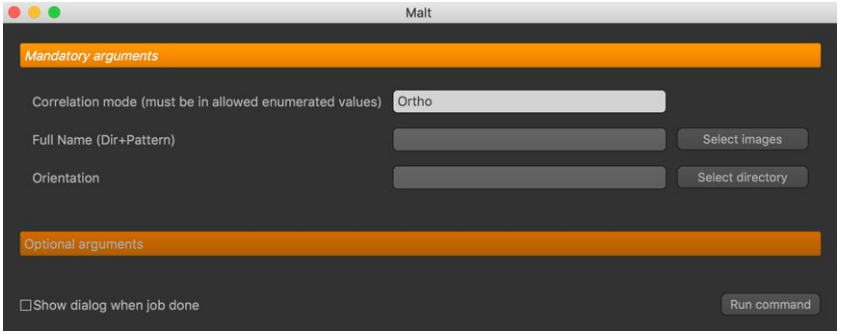

The Malt equivalent:

```
mm3d Malt Ortho ".*JPG"  Ground_UTM ResolTerrain=1 EZA=1 ZoomF=1
```

*Or use the UrbanMNE option for DSM oriented data.*

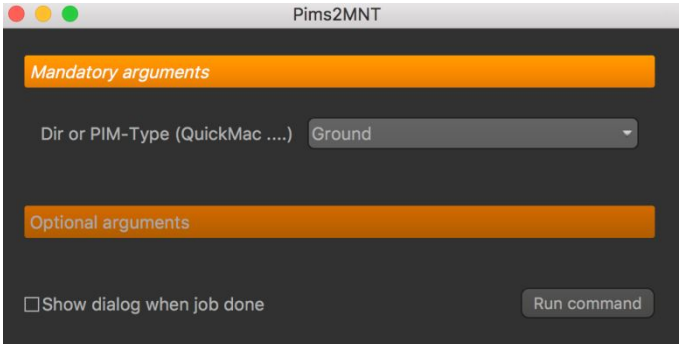



*The EZA parameter ensures the output is in a physical unit (e.g. metres).*




Following this, the DSM and individual ortho-production is run with (Skip this step if using Malt):

```
mm3d Pims2MNT MicMac DoOrtho=1
```




The keyword denotes the parameters used, as this is the simplified interface (Others include Forest, Ground, BigMac...). Then an ortho-mosaic (tiled with larger areas) is created. For most cases BigMac is sufficient, with Forest (which uses epipolar geometry) being best for forestry and urban settings where there is a lot of vertical variation in relatively small areas.

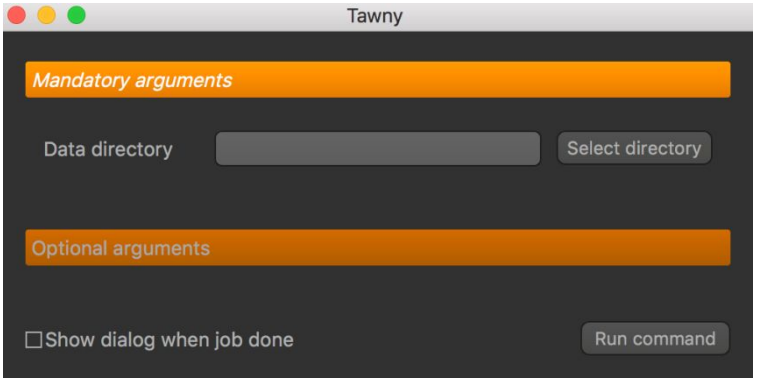
The ortho-mosaicking command is Tawny, which is suitable for produce mosaics from up to 2-300 images in near-ideal acquisition conditions. Beyond this, it is highly recommended to use one of the batch scripts provided to produce large-scale mosaics

```
mm3d Tawny PIMs-ORTHO/ RadiomEgal=1 Out=Orthophotomosaic.tif
```

OR

```
mm3d Tawny Ortho-MEC-Malt DEq=1
```

*The DEq command is the degree of illumination matching between orthophotos. Provided the survey is carried out in stable illumination conditions, the default (omit the argument for a value of 1) usually works. The parameter 'DegRap=' also helps to mitigate for a radiometric drift from occurring (up to a point – see later!). Tawny is really only suitable for small datasets where there is unlikely to be much illumination variation or where the capture conditions on a larger dataset have been uniform (or largely at least!).*




*Mosaicing can be performed with a number of open-source libraries other than MicMac (e.g. Ossim, GDAL, RSGISlib) as well as commercial packages. Bear in mind that experimentation will be required to obtain "good" results with external packages.*

The last step is to produce a point cloud for visualisation in software such as Meshlab or CloudCompare.

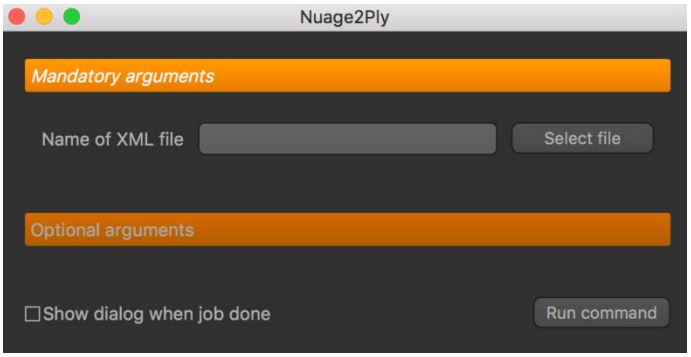
```
mm3d Nuage2Ply PIMs-TmpBasc/PIMs-Merged.xml
Attr=PIMs-ORTHO/Orthophotomosaic.tif Out=pointcloud.ply
```

Nuage2Ply sometimes struggles with larger outputs. Hence Pims2Ply is a useful alternative.

```
mm3d Pims2Ply MicMac Out=Final.ply
```

OR

```
mm3d Nuage2Ply MEC-Malt/NuageImProf_STD-MALT_Etape_8.xml
Attr=Ortho-MEC-Malt/Orthophotomosaic.tif
Out=OUTPUT/PointCloud_OffsetUTM.ply
```




Assuming a successful result, end-user production begins from this point. One initial consideration is the production of DEM via classification of the resulting point cloud.

## Extra functions of note

### SaisieMasqQT and C3DC

SaisieMasqQT is a masking tool for either images or point clouds and C3DC (Culture 3D Cloud), is the 'most automated' point cloud generation command, similar to PIMs above. They are usually used in combination. The script Oblique.sh utilises these in its pipeline. Unfortunately, SaisieMasqQT only operates in relative geometry at present if used directly for the dense cloud generation, hence is only useful for experimentation/evaluation of results. The command *can* be used earlier in the process to mask invalid points, but the orientation and bundle adjustment must be run again, which will result in a lengthier wait processing wise.

**Direct use for dense point clouds (no geo-reffing).**

```
mm3d SaisieMasqQT Arbitrary.ply
```

A GUI will appear in which a mask can be drawn on the point cloud, which may be used to limit the correlation zone to the area selected. See the drop-down menus and/or MicMac manual for further details on the masking options.

The resulting polygon (Arbitrary_polyg3d.xml ) may be used to constrain the dense correlation in the C3DC (Culture 3D Cloud) command. This command is largely similar to the PIMs command and the same algorithmic choices are available (e.g. Statue, Forest, BigMac etc.).

```
mm3d C3DC Forest .*JPG Arbitrary ZoomF=2 Masq3D=Arbitrary_polyg3d.xml
Out=Dense.ply
```

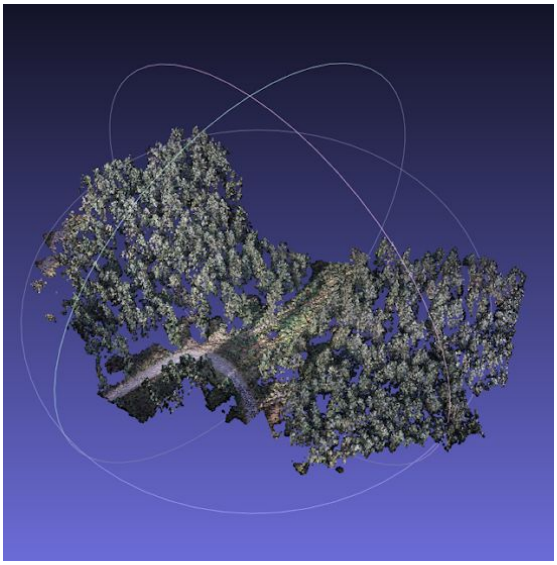


Fig 11: The point cloud from C3DC.

Following the point cloud generation, the tools TiPunch can be used to generate a mesh from the point cloud. A mesh is equivalent to a Triangular Irregular Network (TIN), an alternative representation of a surface based on a tessellation of triangles as opposed to a raster grid.

```
mm3d TiPunch Dense.ply Mode=Statue Out=MeshOot.ply Pattern=.*JPG
```

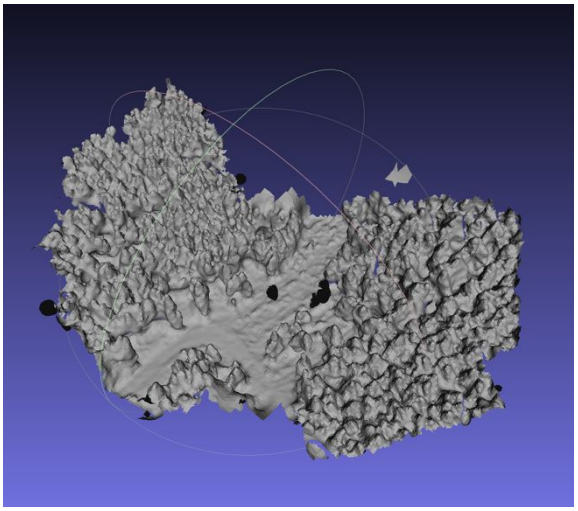


Fig 12: The mesh created with TiPunch.

Finally, the command Tequila is used to texture the mesh with the imagery.

```
mm3d Tequila .*$EXTENSION Statue MeshOot.ply Filter=1
```

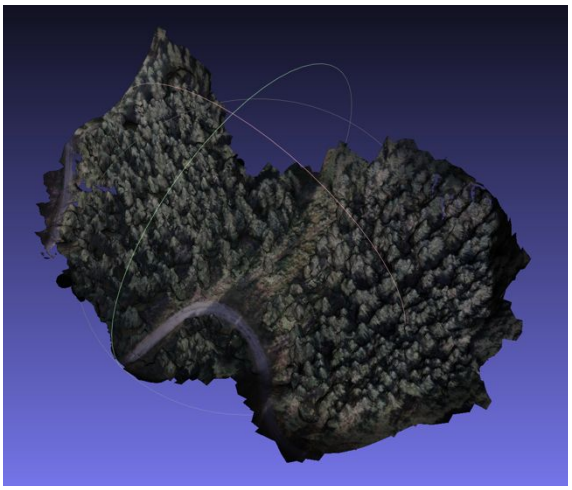


Fig 13: The textured mesh created with Tequila.

**Use for masking invalid points (hence geo-reffed dense matching is available later)**

After processing an initial orientation as normal, create a mask from the sparse cloud created with AperiCloud. This 3d information can then be used to filter the original tie-points and the orientation run again. First select/mask the points.

```
mm3d SaisieMasqQT AperiCloud_Arbitrary.ply
```

The point selection info is then implicitly picked up by MicMac at the next stage (a corresponding xml file), hence the sparse cloud is specified.

```
mm3d HomolFilterMasq .*png OriMasq3D=Ori-Arbitrary/ Masq3D=
AperiCloud_Arbitrary.ply
```

Rename the original tie points, then the filtered one will be seen as the default.

```
mv Homol HomolInit
mv HomolMasqFiltered/ Homol
```

## Typical Output Folder Structure

After processing with individual commands or a script the working folder should have the following directories. MicMac generates a lot of intermediate stuff – this does however ensure each process is well documented.

| Folder | Contents |
|---|---|
| **Tmp-MM-Dir/** | Internal working directory |
| **Pyram/** | Image pyramids |

| | |
|---|---|
| **Pastis/** | Tie point processing |
| **Homol/** | Image pairs / tie point data |
| **Homol_mini/** | Cleaned image pairs and tie points from Schnapps command |
| **Ground-RTL/** | Image orientation data in a relative coordinate system |
| **Ground-UTM/** | Image orientation in a geographic coordinate system |
| **AperiCloud_Rel.ply** | Sparse cloud viewable with Meshlab/Cloud compare |
| **PIMs-Forest/** | The PIMS equivalent folder to Malt |
| **PIMs-Ortho/** | Ortho imagery processed with PIMs |
| **PIMs-TmpBasc/** | PIMs folder containing final DSMs, masks and cloud |
| **Ortho-MEC/** | Ortho imagery processed with Malt |
| **MEC-Malt** | Folder containing final DSMs, masks and cloud |
| **Poubelle/** | Rejected images from Schnapps |

There may be omissions/ name variation depending on the dense cloud method used, hence PIMs outputs are in yellow and Malt in cyan. With the use of the scripts an additional OUTPUT folder is generated with end user products. With large datasets this can take up a lot of space due to all the intermediate data. If space is an issue and **processing has been done to a satisfactory level**, only the final DSM(s) and ortho-mosaic need to be retained.

## Scripts

As promised earlier, a number of scripts have been written (hosted here), where instructions on use and access to the full source code is available. This repository is private and requires an invite to the repo. Please consult the README and -help for each script. The following quickly summarises basic execution. **It is likely easiest when using C3P (C-astral Software) to write all the attributes to the JPEG EXIF when using the RGB camera.** It is my preference to use separate csv with the Red-Edge camera as the C-Astral tag writing is questionable, post GPS correction. This is not an exhaustive account of every script argument, please use the –help argument to see them as they are all self-explanatory. Lastly – the key to high quality results are a good acquisition strategy (e.g. flight plan and/or pattern of oblique imagery) and good conditions for photography (e.g. diffuse light, minimal shadows etc.).

### Preparation

**Please ignore this first stage if the image EXIF has GNSS embedded!**

If you wish to use csv information rather than exif, we use the pycmac library (more on which later), which has been written for keeping everything to one language, of which python is the choice here due to simplicity and readability.

Open an ipython terminal and type:

```
mm3d SetEXIF ."*JPG" F35=45 F=30 Cam=ILCE-6000.
```

For other drone types you will need F35 (focal length 35mm equivalent), F (focal length), and Cam(era) parameters. These are relatively easy to find online. Those above are for the PPX.

## Complete workflow shell scripts

Complete workflow scripts are particularly useful if the parameters are already established and it is simply a case of leaving processes to run. For experimentation, either individual commands (see previous sections) or substage scripts are recommended – particularly if the error of different stages is to be derived. With all cases below – using the –h flag will display all the parameter options. The basic parameters are described here, but full explanation is provided from the script.

### Drone.sh

This script currently utilises the Malt-based pipeline. After execution, the final results are stored in a folder named OUTPUT, and will consist of a DSM, ortho-mosaic and coloured point cloud (.ply file).

The script Drone.sh can use either csv-based or EXIF information (e.g GNSS coordinates) on the EXIF. Use the –csv 1 if this is the case and it will search for the csv in question in the directory. This is the case in all subsequent scripts. This script is suitable for relatively small datasets (e.g. 100s of images at most) as larger dataset ortho-mosaics will be unevenly illuminated.

```
Drone.sh -e JPG -u "30 +north" -r 0.1 -g 1 -z 2
```

*Where:*

*–e = the file extension*

*-u = the UTM zone, for the UK this is typically 30 +north*

*-r = the output pixel resolution in metres (omit this to process at maximum)*

*-g = whether or not to use a GPU – this is limited to relatively small datasets due to inefficient memory use*

*-z = the point density of the cloud*

- *1 = 1 point per image pixel*
- *2 = 1 point per 4 pixels etc.*

*For help on all parameters:*

Drone.sh -h

This script currently utilises the Malt-based pipeline. After execution, the final results are stored in a folder named OUTPUT, and will consist of a DSM, ortho-mosaic, coloured point cloud (.ply file) and correlation surface.

### Drone_PIMs.sh

Largely the same as the previous script, but uses the PIMs pipeline which is best suited to highly detail DSM production and/or multiple view angles. It is not necessarily always best suited to ortho-mosaic production and as such the other scripts are available for this (the Malt-based pipelines). It can be applied from small-medium (10s-3000s) to large (>3000s) datasets for which it has optional tiling functionality. The ortho-mosaic is tiled by default using either option (see python scripts section.

```
DronePIMs.sh -e JPG -a Forest -u "30 +north" -r 0.02
```

Largely similar to the above:

*Where:*

*–a = Algorithm type (Forest, BigMac, MicMac – detail in descending order)*

*–e = the file extension*

*-u = the UTM zone, for the UK this is typically 30 +north*

*-r = the output pixel resolution in metres (omit this to process at maximum)*

*-z = the point density of the cloud*

*(higher for smoother DSMs –lower when high detail may be required)*

- *1 = 1 point per image pixel*
- *2 = 1 point per 4 pixels etc.*

*For most cases BigMac will suffice (1 point per 4 pixels), use Forest for more complex terrain/veg/buildings. A good example is coniferous forest, which may be done greater justice with the Forest mode (surprisingly enough!).*

## Griproc.sh

The underlying process is getting more complex now! There are two algorithmic choices to attempt generation of a near-seamless ortho-mosaic. Sometimes, In order to generate a more evenly illuminated ortho-mosaic from a very large dataset (e.g 1000s of images), an alternative strategy is required as mosaicking individual ortho-photos using the native MicMac command tends to produce artefacts and uneven illumination over large areas (effectively a 'radiometric drift').

Basically, this script divides the imagery into a grid defined by the user (e.g. 6, 6 = 36 tiles) and processes these smaller mosaic tiles that are merged into a larger mosaic by the ossim library, which can produce superior (though not always – data specific) large-scale results via this command. The feather and max options for the ossim command are most effective for boxed and non-uniform mini-mosaics respectively.

```
gridproc.sh -e JPG -u "30 +north" -g 6,6 -w 2 -gpu 1 -b 10
```

*Where:*

*–e = the file extension*

*-u = the UTM zone, for the UK this is typically 30 +north*

*-x = x,y grid values e.g. 6,6 is a 36-tile grid*

*-w = correlation window size*

*-g = use the GPU (1)*

*--b = no of batch jobs to process at once*

*For help on all parameters: -h*

This command may occasionally be better for ortho-mosaics for big datasets, but does not guarantee satisfactory results! It is recommended the tiles are checked and any artefacts cropped prior to the last stage of the large-scale mosaic. **As always good quality image acquisition is the key to good results!**

If the GPU is being used, the number of batch jobs and CPU cores will be limited depending on multiple factors. (image size, GPU memory and no. of CPU cores). Using the python joblib module, each CPU is assigned one image batch to pass to the GPU. It is best to stick to the number of physical cores (as opposed to threads/logical cores) when using the GPU support.

## Workflow stage scripts

The following shell and python scripts process the key stages of the SfM process. Whilst these scripts are integral to Grid_proc.sh and DronePIMs.sh respectively, they can be used in isolation on the assumption data has been generated correctly previously. This is useful for checking results and rerunning various stages and having greater control of input parameters.

### Orientation.sh

```
Orientation.sh -e JPG -u "30 +north" -c Fraser
```

*Where:*

*-e = image file extension*

*-u = the UTM zone*

*-c = the lens calibration model*

In the above command the key point detection, orientation and GNSS aided orientation are performed with .txt outputs of the orientation results and and a sparse cloud in .ply format. This is the precursor to the dense matching scripts below.

### Dense_cloud.sh

```
dense_cloud.sh -e JPG -a Forest -m PIMs -
```

*Where:*

*-e = image file extension*

*-a = the algorithm type eg Forest, Ortho*

*-m = the MicMac matching mode – either Malt or PIMs*

*-u = the UTM zone*

In the above command the key point detection, orientation and GNSS aided orientation are performed with .txt outputs of the orientation results and and a sparse cloud in .ply format. This is the precursor to the dense matching scripts below.

### MaltBatch.py

```
MaltBatch.py -folder $PWD -algo UrbanMNE -num 3,3 -zr 0.01 -g 1 -nt 4
-bbox False
```

*Where:*

*-folder = the input directory (the processing directory)*

*-algo = the dense cloud algorithm in this case optimised for urban landscapes (see MicMac docs)*

*-num = the grid pattern of tiles (here a 3 x 3 grid)*

*-zr = regularization factor in the Z plane (default is 0.02)*

*-g = use the gpu or not (bool)*

*-nt = number of CPU threads (important to balance this against GPU useage)*

*-bbox = crop the tile to a uniform box or not (bool)*

In the above command, we are using the UrbaMNE algorithm on a 3x3 grid of tiles, with the gpu and processing 3 tiles in parallel.

### PimsBatch.py

```
PimsBatch.py -folder $PWD -algo Forest -num 3,3 -zr 0.01 -g 1
```

The PIMs version of MaltBatch, inputs are much the same.

*Where:*

*-folder = the input directory (the processing directory)*

*-algo = the dense cloud algorithm in this case optimised for urban landscapes (see MicMac docs)*

*-num = the grid pattern of tiles (here a 3 x 3 grid)*

*-zr = regularization factor in the Z plane (default is 0.02)*

*-g = use the gpu or not (bool)*

Whilst GPU use is an option here**, it is not recommended if you have lots of CPUs as the GPU memory will be overloaded rapidly in the current MicMac implementation**.

### Orthomosaic.sh

Creates a large orthomosaic **based on the outputs of MaltBatch.py, PIMsBatch.py or selected folder**.

```
orthomosaic.sh -f $PWD -u '30 +north' -mt ossimFeatherMosaic -o
outmosaic.tif
```

*Where:*

*-f = the input directory (the processing directory)*

*-u = UTM Zone of area of interest. Takes form 'NN +north(south)'"*

*-mt =OSSIM mosaicing type*

*(e.g. ossimBlendMosaic ossimMaxMosaic ossimImageMosaic ossimClosestToCenterCombiner ossimBandMergeSource ossimFeatherMosaic")*

*-o = Output mosaic e.g. mosaic.tif"*

### Dsmmosaic.sh

Creates a large DSM mosaic **based on the outputs of MaltBatch.py or PimsBatch.py**.

```
dsmmosaic.sh -f $PWD -u '30 +north' -mt ossimMaxMosaic -o
outdsmosaic.tif
```

*Where:*

*-f = the input directory (the processing directory)*

*-u = Zone of area of interest. Takes form 'NN +north(south)'"*

*-mt = OSSIM mosaicing type*

*(e.g. ossimBlendMosaic ossimMaxMosaic ossimImageMosaic ossimClosestToCenterCombiner ossimBandMergeSource ossimFeatherMosaic")*

*-o = Output mosaic e.g. mosaic.tif"*

## Complete workflow scripts in practice

**Whilst it is likely better to use either the python library workflow stage scripts for greater control over the overall process, the complete workflow scripts provide a single command solution when the user is confident of the parameters required. In the following notes the workflow stage alternatives are in brackets (refer to –h or the earlier sections on parameters). These may be harder to debug if things go wrong midway!**

If the main goal is to generate a DSM and ortho the following script is enough for most cases on modestly sized datasets (<200 images):

```
Drone.sh -e JPG -u "30 +north" -r 0.1 -g 1
```

In some cases, per-image matching *may* be preferable (e.g. coniferous forest, using epi-polar geometry). In this case DronePIMs.sh is used (Orientation.sh -> dense_cloud.py in sequence).

```
DronePIMs.sh -e JPG -u "30 +north" -a Forest -u "30 +north"
```

It is recommended to use “Forest” or Statue for complex features such as forests and urban landscapes where epipolar geometry *may* be favoured over ground geometry (see below). This script has a tiling feature similar to gridproc.sh which can be utilised if the dataset is large (>1000 images).

For generation of a large scale ortho-mosaic and DSM, gridproc.sh imay be useful for very large datasets, though manually splitting up the data is ultimately safer (Orientation.sh -> MaltBatch.py in sequence).

```
gridproc.sh -e JPG -u "30 +north" -x 6,6 -g 1 -w 1 -b 10
```

This is a grid of 36 tiles, using GPU support, in batches of 10. This script also merges the DSMs. It is context specific as to whether or not this is favoured over the PIMs-based pipeline. The most important consideration is that Malt-based matching operates in multi-view stereo, ground-based

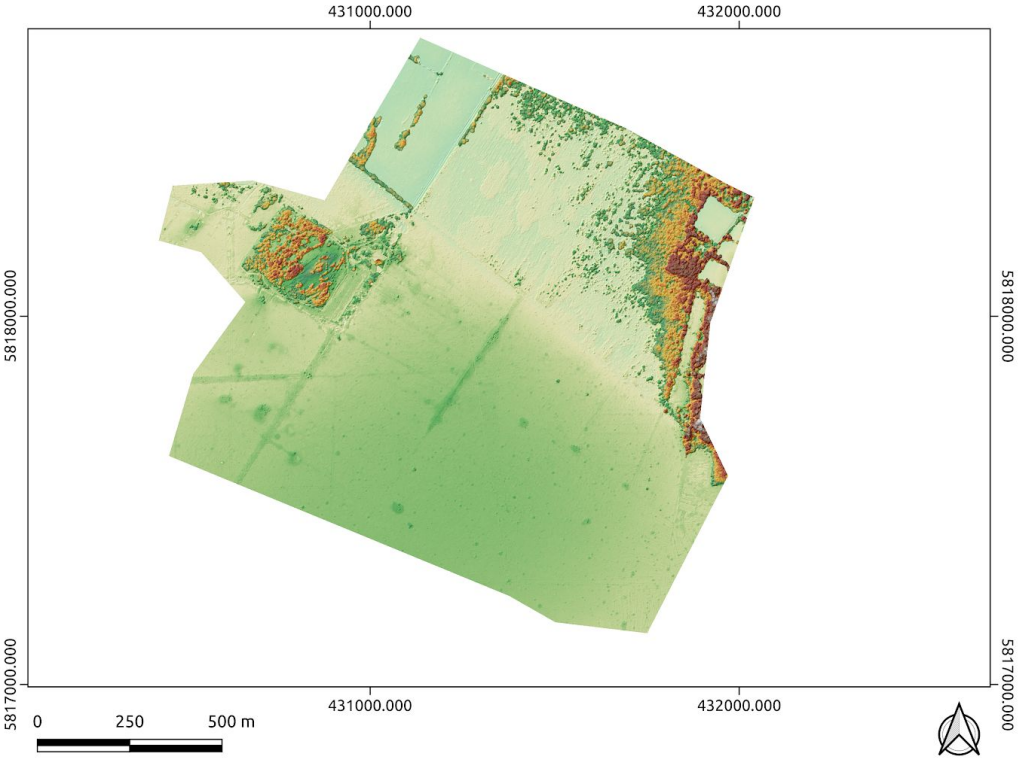
geometry for automated matching, where we are effectively extruding features from a plane ('2.5D'), whereas PIMs 'Forest' or 'Statue' process the point cloud via pairs in epi-polar geometry.

With either the substage or all-in-one scripts, there are some parameter choices related to mosaicing. There is no definitive answer as to whether to apply radiometric equalisation when using the native Tawny-based approach. Ultimately a judgement has to be made based on the contents of the imagery, and the uniformity of lighting conditions during the flight. Often, no equalisation is required and it is probably best to start with this (e.g. RadiomEgal=0), which is the default for PimsBatch.py. If results are not satisfactory, the parameters can be experimented with by only repeating the Tawny command rather than the entire scripts.

## Suggested overall strategies

### Nadir Imagery

Nadir is imagery where we are looking directly down at the subject - even imagery looking diagonally down is good (pseudo-nadir), particularly in rendering facades. This is the 'best' mode of acquisition for UAVs as it combines 'top down' map-like viewing with the ability to render 2.5D information such as the sides of buildings and trees in high detail. This also makes it 'simpler' to process from an algorithmic point of view. Most nadir/pseudo-nadir acquisitions should be a grid like pattern with a minimum of 60-70% overlap but ideally 80 - 90%. Most planning software (e.g. C3P for the Bramor) facilitate this. It is likely imagery captured on banking will not be useful due to blurring and poor overlap with the rest of the imagery. Typically a ground geometry approach (Malt algorithm) should be used or attempted in the first instance and will usually produce good results such as those seen below in relatively "smooth" landscapes.


431000.000
432000.000
5818000.000
5817000.000
0
250
500 m

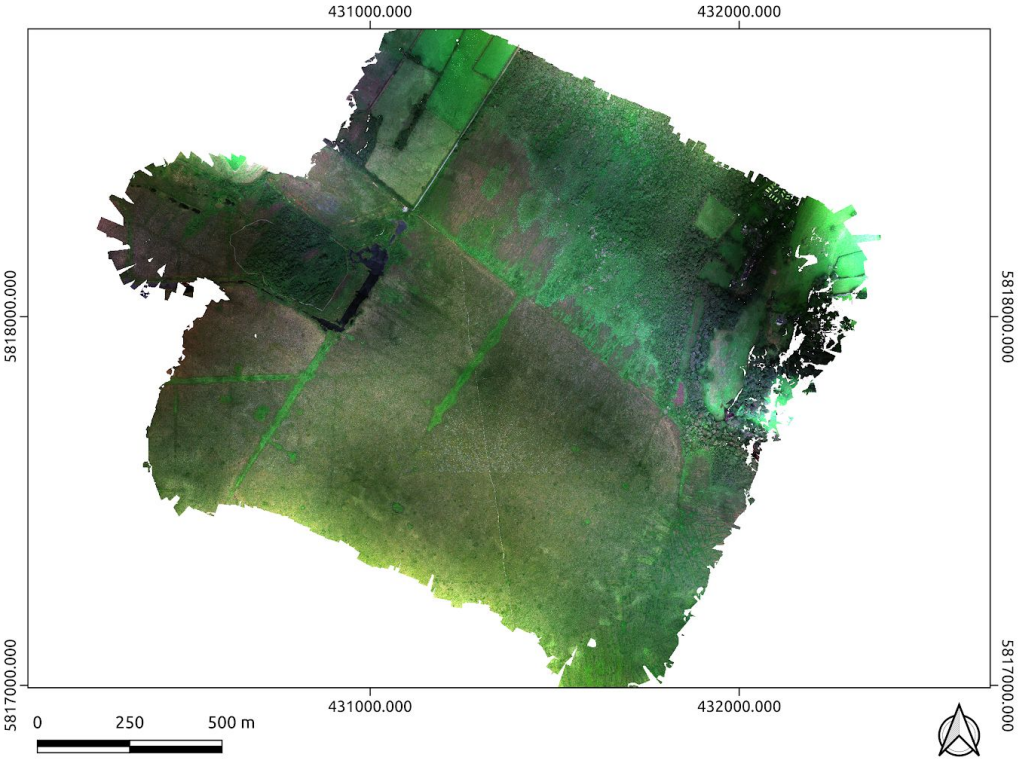

431000.000
432000.000
5818000.000
5817000.000
0
250
500 m

Fig 10: DSM and RGB Ortho-mosaic derived from ground-geometry-based reconstruction.

Where the land surface is more complex, image/epipolar geometry will likely produce better results as can be seen in Fig. 11 of forest canopy structure.

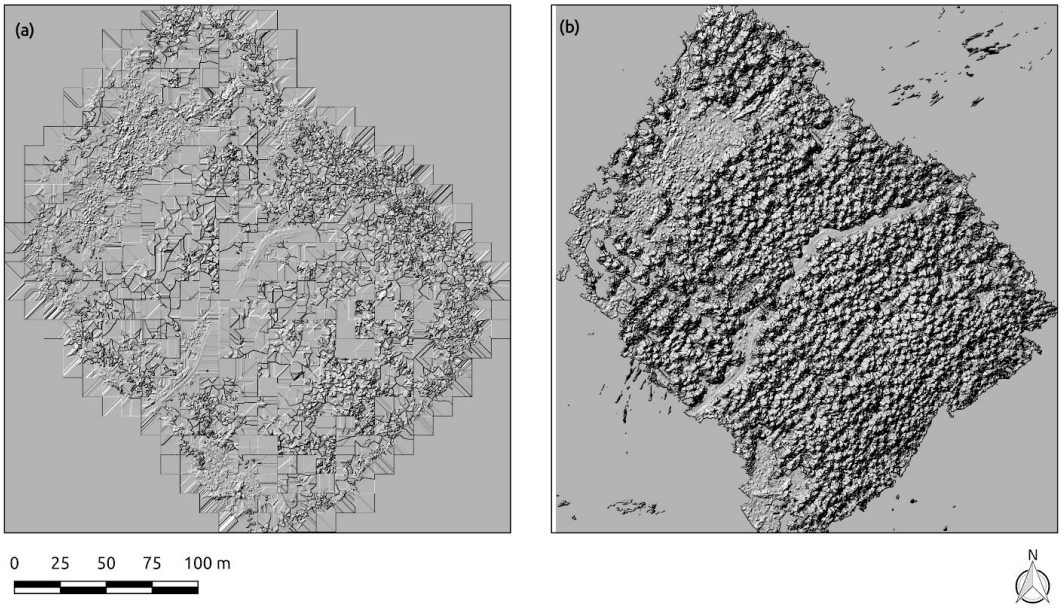



Fig 11. Ground vs. epipolar geometry in the reconstruction of complex terrain.

## Non-nadir imagery

Whilst nadir/pseudo-nadir imagery is almost always best from a UAV, there may be instances where oblique imagery is collected. Typically, this would occur using a multi rotor with a gimble, where some acquisitions were taken viewing towards the horizon (or side view of an object) or upwards to the sky. For non-nadir/pseudo-nadir imagery such as side views of buildings etc, the manual solution is to carefully order the imagery. Using GPS data is of limited utility for matching due to objects blocking views. Hence a predefined ordering (e.g. radial, or 'in a line') definitions are required for efficient semi-automated matching. Most Sfm software provide utilities to this end (including MicMac).

Fig 12: Some examples of sub-optimal acquisition of building facades – exhaustive tie point detection may be necessary here for best results (assume the rest of the dataset is similar).

A good acquisition strategy in the above example of a building could be radial at rising elevations to ease the processing (through a linear pattern of overlap) and enhance the quality of results. It is best

to have the entire object of interest (from the specified pose) at all times within the field of view to ensure a good quality orientation and dense cloud.

Alternatively, perhaps the most robust approach is to allow corresponding features to be detected automatically, using the 'All' keyword in the Tapioca command, an option in the **Oblique.sh** script, which has been written to produce point clouds and meshes from such imagery. This is also an option (-m) in the other scripts above. The input arguments for Oblique.sh are similar to the other scripts (use -help to see the arguments).

```
Oblique.sh -e JPG -a Statue -d 10 -u "30 +north" -z 2
```

It is worth noting that by allowing an exhaustive search for matching features in imagery will take longer than using GPS coordinates to define neighbours but is probably more robust. Oblique.sh has been written principally with experimentation in mind with the other scripts to produce large scale datasets.

## Micasense camera scripts

The following is intended for the processing of data from the Micasense RedEdge (MRE) camera variants. The surface reflectance processing is based on the Micasense python API found here. General information on the sensor can be found here. **As AU EO-lab possesses a Micasense camera, this has been documented, though QinetiQ do not possess such a camera, hence this is unlikely to be encountered, but is here as a resource in the event. Metashape and WebODM support this camera (and others) and provide an easier to operate solution from their GUIs.**

The MRE captures photos in 5 bands – blue, green, red, red-edge (RE) and near-infra-red (NIR). The latter two are particularly useful for monitoring vegetation vigour/health and basic classification of vegetation land cover types and enhance the detection of other cover types (water, soil) via the preferential absorption and reflection of certain bands. The MRE is calibrated before each flight using a reference panel as seen below.


Fig 12 : Reference panel as captured via the Micasense App in the field.

The blank panel on the left is used to calibrate the sensor before the flight with the code panel on the right the ID unique to the camera model, which is used later during surface reflectance conversion. This must be repeated for every flight.

## MSpec.py

This script converts band imagery to surface reflectance and aligns the band imagery (which are offset by default due to the camera design), using a feature rich image chosen by the user. The alignment image should contain lots of reliable features such as roads, vehicles, buildings for a good alignment (see the figure below). It is important to include a capture of such features during the flight for this purpose.  A basic usage using mandatory args would be:

```
MSpec.py -precal pathto/calimgaery -img pathto/rawimagery -o
pathto/outputdir -alIm 0007
```

*Where:*

*-precal = path to pre flight calibration images*

*-img = path to flight images*

*-o = path to output reflectance directory*

*-alIm = path to alignment image*

*Optionally:*

*-stk = 1 or 2. This will stack the image bands for each capture, making later Sfm processing quicker and easier. Option 1 produces a 5-band stack whereas option 2 produces RGB and Red, RedEdge,NIR composites respectively.*

Execution will result in a folder for each band (B, G, R, NIR, RE) with aligned surface images in .tif format, ready for Sfm processing (or in the case of the stack option simply the capture stacks). Previews of the alignment imagery are also available for assessment before further processing (examples below).

Fig 13 : An example of a good alignment image result displayed in R,G,B and NIR, G, R respectively.

There are two options for SfM processing of the MS imagery. The first is to process each band separately, which is ultimately more time consuming from a computational point of view. The second and much more efficient option is to process the stacked output from the MSpec.py script. In which case, the SfM routines described earlier are directly applicable to the stacked imagery, provided –stk 2 is chosen as MicMac only supports 3-band processing at a time at present. The –stk 1 option is applicable to Photoscan/Metashape if desired. It is recommended to apply the DronePims.sh script to the 3-band images using the single tile option. The separate 3 band images may then be merged using MStack.py, which produces a 5-band composite of the data from the initial 3-band outputs.

## MStack.py

This script merges the 3-band Sfm outputs from MicMac into a 5 band raster consisting of the (in band order) Blue, Green, Red, Near-infra-red, red-edge bands.

*Where:*

*-in1 = the path to the RGB mosaic (e.g. RGB/PIMS-ORTHO/Orthophotomosaic.tif )*

*-in2 = the path to the Red, Red Edge, NIR mosaic (e.g. RRENir/PIMS-ORTHO/Orthophotomosaic.tif )*

*-o = the output 5-band raster e.g. (MicasenseStack.tif)*

*Note the 3-band folders in brackets are named as they would be outputted from the MSpec.py script.*

### Multi-spectral workflow using scripts (deprecated)

The following describes the recommended workflow for the Micasense camera. The multi spectral workflow is executed in stages. First the MSpec.py script is applied with the –stk 1 flag indicating the intention to produce RGB and RRENIR composites for SfM processing.

```
MSpec.py -precal calib -img 0005SET/000 -o ootest -alIm 0012 -stk 2
```

Next Orientation.sh script is applied to each composite folder.The calibration subset can be produced using the subset document instructions. Either the GNSS csv or the exif data can be used, hence these are optional arguments chosen here.

```
cd RGB
Orientation.sh -e tif -u "30 +north" -c Fraser -t log.csv -s sub.csv
cd RRENir
Orientation.sh -e tif -u "30 +north" -c Fraser -t log.csv -s sub.csv
```

## Python alternative

This python library is potentially useful if you are familiar with python. This comes at the cost of some assumptions and details about the underlying MicMac commands however as well debugging. API documentation is available here and is written and maintained by the author and colleagues. A set of workflows via the jupyter notebook environment is available to QQ staff who wish to use the library to interactively run. This method is less verbose than those previously documented but it is equally prone to the vagaries of SfM processing.

### Modules

The library has a modular structure consisting of:

```
pycmac.utilities
```

Files sorting and transferring and sub-setting, xml parsing (e.g. XifGps2Txt etc.)

```
pycmac.dense_match
```

Wraps all the dense matching algorithms (Malt, PIMs).

```
pycmac.orientation
```

Wraps all the commands associated with image orientation and bundle adjustment, as well sub-setting and xml parsing operations.

```
pycmac.mspec
```

Provides the same functionality as the MSpec.py script documented earlier, which converts Micasense multispectral imagery to surface reflectance with a choice of output formats using the micasense lib.

```
pycmac.sfm
```

Provides commands for complete workflow of either multispectral or RGB imagery (warning: simplifications and assumptions are made here).

```
pycmac.utilities
```

Miscellaneous  functionality.

### Typical usage and workflow

This material is also available in the documentation. Typical usage is as with all python libraries and a simple workflow on RGB data would be:

```
from pycmac import orientation, dense_match
```

Perform the relative orientation of images (poses).

```
orientation.feature_match(folder, proj="30 +north",  ext="JPG",
schnaps=True)
```

Perform the bundle adjustment with GPS information.

```
orientation.bundle_adjust(folder, algo="Fraser", proj="30 +north",
                  ext="JPG", calib=pathtocsv.csv, gpsAcc='1')
```

Perform the dense matching using the malt algorithm. The args for the dense matching algorithms are largely identical to the MicMac commands (Malt & PIMs), but carry out additional masking, georeferencing and subsetting.

```
dense_match.malt(folder, proj="30 +north", mode='Ortho', ext="JPG",
orientation="Ground_UTM",
         DoOrtho='1',  DefCor='0')
```

Mosaicing can be performed using Tawny or seamline feathering.

```
dense_match.tawny(folder, proj="30 +north", mode='PIMs')
```

This algorithm currently only applies to single-band imagery.

```
dense_match.feather(folder, proj="ESPG:32360", mode='PIMs', ApplyRE="1")
```

The complete workflows can be executed via the sfm module with either:

```
mspec_sfm(wrkdir, csv=log, sub=sub, gpsAcc='3')
```

or:

```
rgb_sfm(wrkdir, csv=log, sub=sub, gpsAcc='3')
```

# Appendices

## GUI-based software/alternatives

These applications are interesting and relatively easy to use, though do not provide geospatial outputs.

### Colmap

Colmap is a library with command line, GUI and GPU support. With a GUI it may represent a more useable solution for those at the very least from an exploratory perspective with the caveat that outputs are of limited geospatial value. For a quick look however, it can be helpful.

The documentation can be found here which contains further links to the source code site etc. The official site with binaries for windows and mac can be found here. Outputs are limited to a depth map and point cloud. The point cloud could be converted and gridded to DSM using a combination of Cloud Compare and/or QGIS if desired. Colmaps use of GPU processing is also more complete than either MicMac or WebODM as it is used during key point detection and dense cloud generation. As with every application at present, it must be a CUDA supporting card.

### Alicevision

Alicevision/Meshroom is a sophisticated collection of software for Structure from motion and mesh generation, found here. It can be used as an executable GUI or as independent libs accessible system wide depending on preference. The intuitive nature of the GUI makes this appealing for those without programming experience. The focus is on modelling of objects and computer vision, but it is a useful tool nonetheless, with full GPU integration.