sPyMicMac Documentation

Release 0.2.2.dev0+gec16355.d20231019

Robert McNabb

Oct 19, 2023

CONTENTS

This is the documentation for spymicmac, a set of python tools for processing historic spy satellite and aerial imagery using MicMac.

If you encounter any issues, use the issue tracker on the GitHub page. If you found any of the tools helpful, let me know!

CHAPTER

ONE

INSTALLATION AND SETUP

The following is a (non-exhaustive) set of instructions for getting setup to run spymicmac on your own machine. Note that this can be an **extremely** computationally intensive process, so we don't really recommend trying to run this on your personal laptop.

As this is a (non-exhaustive) set of instructions, it may not work 100% with your particular setup. We are happy to try to provide guidance/support, **but we make no promises**.

1.1 Installing MicMac

Detailed installation instructions for MicMac on multiple platforms can be found here, but we've added a short summary to help guide through the process.

First, clone the MicMac repository to a folder on your computer (you can also do this online via github):

```
/home/bob/software:~$ git clone https://github.com/micmacIGN/micmac.git
...
/home/bob/software:~$ cd micmac
/home/bob/software/micmac:~$ git fetch
/home/bob/software/micmac:~$ git checkout IncludeALGLIB
```

This will clone the MicMac git repository to your machine, fetch the remote, and switch to the *IncludeALGLIB* branch. Check the **README.md** (or **LISEZMOI.md**) file to install any dependencies, then:

```
/home/bob/software/micmac:~$ mkdir build && cd build/
/home/bob/software/micmac/build:~$ cmake .. -DWITH_QT5=1 -DWERROR=0 -DWITH_CCACHE=OFF
...
/home/bob/software/micmac/build:~$ make install -j$n
```

where `$n` is the number of cores to compile MicMac with. The compiler flag `-DWERROR=0` is needed, as some of the dependencies will throw warnings that will force the compiler to quit with errors if we don't turn it off.

Finally, make sure to add the MicMac bin directory (`/home/bob/software/micmac/bin` in the above example) to your `$PATH` environment variable, in order to be able to run MicMac. You can check that all dependencies are installed by running the following:

```
/home/bob:~$ mm3d CheckDependencies
git revision : v1.0.beta13-844-g21d990533

byte order   : little-endian
address size : 64 bits

```

(continues on next page)

(continued from previous page)

```
micmac directory : [/home/bob/software/micmac/]
auxilary tools directory : [/home/bob/software/micmac/binaire-aux/linux/]

--- Qt enabled : 5.9.5
    library path:  [/home/bob/miniconda3/envs/bobtools/plugins]

make:  found (/usr/bin/make)
exiftool:  found (/usr/bin/exiftool)
exiv2:  found (/usr/bin/exiv2)
convert:  found (/usr/bin/convert)
proj:  found (/usr/bin/proj)
cs2cs:  found (/usr/bin/cs2cs
```

In a nutshell, the basic idea is: clone the MicMac git repository, then build the source code. Simple!

1.2 Installing spymicmac

spymicmac is available in a number of ways - either installing from source or packaged via PyPI or conda-forge.

1.2.1 via PyPI

As of version 0.1, spymicmac is available via PyPI. To install the latest packaged version into your python environment, simply run:

```
pip install spymicmac
```

1.2.2 via conda-forge

As of version 0.1.1, spymicmac is available via conda-forge. To install the latest version, run:

```
conda install -c conda-forge spymicmac
```

1.2.3 from source

To get started, clone the repository, then navigate to the directory where the repository is downloaded:

```
git clone https://github.com/iamdonovan/spymicmac.git
```

Optional: Preparing a python environment

If you like, you can set up a dedicated python environment for your spymicmac needs. This can be handy, in case any packages required by spymicmac clash with packages in your default environment. Our personal preference is conda, but your preferences may differ.

The git repository has a file, environment.yml, which provides a working environment for spymicmac and conda. Once you have conda installed, simply run:

```
conda env create -f environment.yml
```

This will create a new conda environment, called spymicmac, which will have all of the various python packages necessary to run spymicmac. To activate the new environment, type:

```
conda activate spymicmac
```

And you should be ready to go. Note that you will have to activate this environment any time you wish to run spymicmac scripts and tools, if it is not already activated in your terminal.

Installing via pip

Once you have the environment prepared (or not), run pip from inside the `spymicmac` directory:

```
pip install .
```

Alternatively, you can install a development version, which allows you to make changes to the code (either via git updates or your own tinkering) without having to re-install each time. To install a development version, use the `-e` option:

```
pip install -e .
```

1.2.4 Checking the installation

Assuming that you haven't run into any errors, you should be set up. You can verify this by running:

```
register_relative -h
```

From the command line. You should see the following output (or something very similar):

```
usage: register_relative [-h] [-ort FN_ORTHO] [-ref FN_REF] [-glacmask GLACMASK] [-
↪landmask LANDMASK]
                         [-footprints FOOTPRINTS] [-im_subset IM_SUBSET [IM_SUBSET ...]]␣
↪[-b BLOCK_NUM] [-ori ORI]
                         [-ortho_res ORTHO_RES] [-imgsource IMGSOURCE] [-density␣
↪DENSITY] [-no_allfree] [-useortho]
                         dirmec fn_dem

Register a relative DEM or orthoimage to a reference DEM and/or orthorectified image.

positional arguments:
  dirmec                the name of the MEC directory to read the relative DEM from (e.g.
↪, MEC-Relative)
  fn_dem                path to reference DEM
```

(continues on next page)

(continued from previous page)

```
options:
  -h, --help            show this help message and exit
  -ort FN_ORTHO, --fn_ortho FN_ORTHO
                        path to relative orthoimage (optional)
  -ref FN_REF, --fn_ref FN_REF
                        path to reference orthorectified image (optional)
  -glacmask GLACMASK    path to shapefile of glacier outlines (i.e., an exclusion mask)
  -landmask LANDMASK    path to shapefile of land outlines (i.e., an inclusion mask)
  -footprints FOOTPRINTS
                        path to shapefile of image outlines. If not set, will attempt to␣
↪download from USGS.
  -im_subset IM_SUBSET [IM_SUBSET ...]
                        subset of raw images to work with (default all)
  -b BLOCK_NUM, --block_num BLOCK_NUM
                        Block number to use if multiple image blocks exist in directory.
  -ori ORI              name of orientation directory (after Ori-) [Relative]
  -ortho_res ORTHO_RES  approx. ground sampling distance (pixel resolution) of ortho␣
↪image. [8 m]
  -imgsource IMGSOURCE  USGS dataset name for images [DECLASSII]
  -density DENSITY      pixel spacing to look for GCPs [200]
  -no_allfree           run Campari with AllFree set to False
  -useortho             use the orthomosaic in Ortho-{dirmec} rather than the DEM␣
↪(default: False). If fn_ortho
                        is set, uses that file instead.
```

CHAPTER

TWO

MICMAC EXAMPLE FILES

On this page, you can download some example files to get started with your MicMac project, though note you will most likely need to modify them as explained in the *micmac processing files* tutorial.

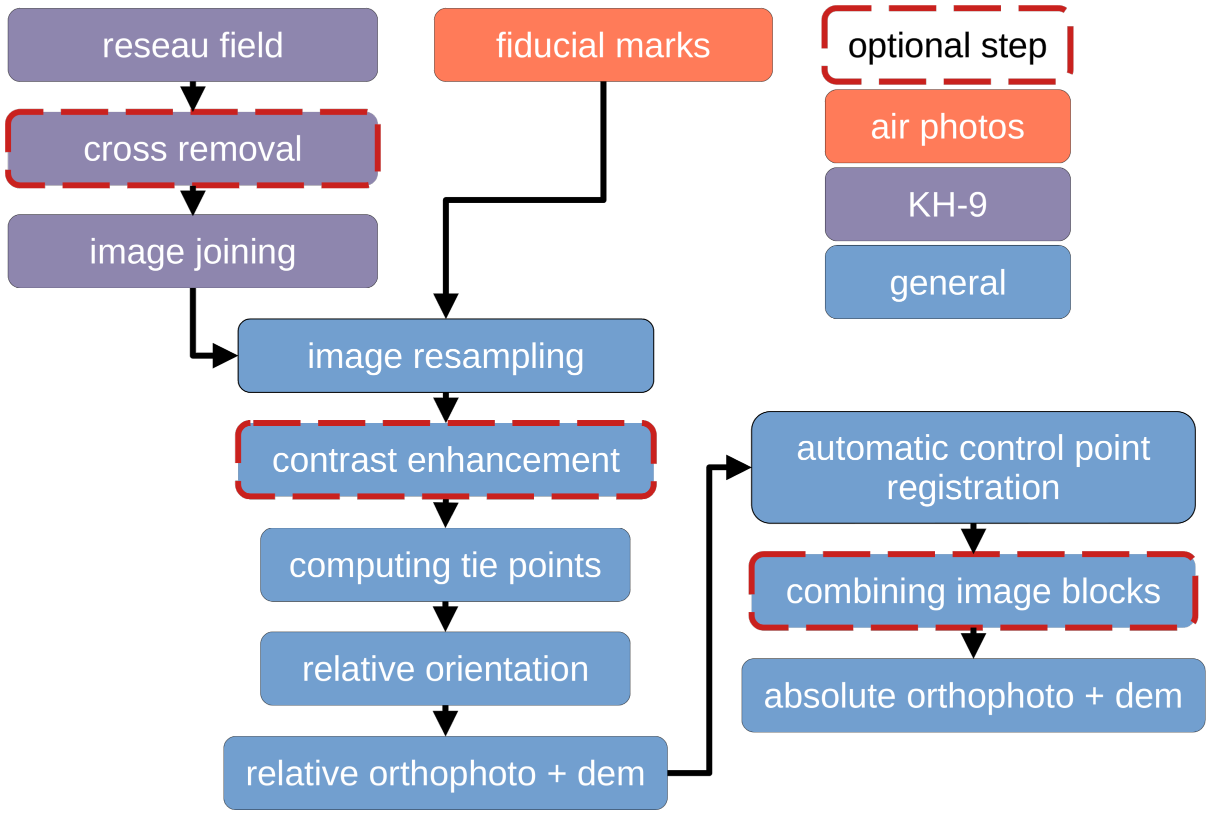
- `id_fiducial.txt`
- `MeasuresCamera.xml`
- `MicMac-LocalChantierDescripteur.xml`

CHAPTER

THREE

TUTORIALS

Here, you will find information about the workflow for using spymicmac to process KH-9 and historic air photos.



3.1 tl;dr guide

This page is designed to provide a "bare bones" user guide to spymicmac/MicMac for processing KH-9 Hexagon Mapping Camera images or scanned air photos. It is presented primarily as a list of commands to run; if you would like a longer explanation of the different steps, see the additional guides *here*.

3.1.1 KH-9 processing

To begin, run *generate_micmac_measures* to generate `id_fiducial.txt` and `MeasuresCamera.xml`.

Next, create a new directory, `Ori-InterneScan`, and move `MeasuresCamera.xml` into it.

Finally, run *`spymicmac.micmac.create_localchantier_xml()`* to create `MicMac-LocalChantierDescripteur.xml`, or `download` the example provided.

Note: The rest of this guide assumes that you have the following directory structure:

```
project
├── id_fiducial.txt
├── Img1_a.tif
├── Img1_b.tif
├── Img2_a.tif
├── Img2_b.tif
...
├── MicMac-LocalChantierDescripteur.xml
├── Ori-InterneScan
│   └── MeasuresCamera.xml
```

where `Img1_a.tif` and `Img1_b.tif` are the two halves of one of the KH-9 images, `Img2_a.tif` and `Img2_b.tif` are two halves of the next image in the acquisition, and so on. For stereo processing, you need a minimum of two images.

joining image halves

To join image halves use the command line tool *join_hexagon*:

```
join_hexagon
```

This will search for all files that fit the pattern `DZB*N001_(a|b).tif`, and join them into a single image, `DZB*N001.tif`. After joining the images, create a new directory, `halves`, and move the image halves into it.

reseau grid

Next, use *find_reseau_grid* to find the location of the 1081 Reseau marks in the image:

```
find_reseau_grid DZB*.tif
```

This will create a file, `Ori-InterneScan/MeasuresIm-DZB*N001.tif.xml`, for each of the images in the directory. It will also create an image in `match_imgs` that shows the estimated distortion pattern.

removing crosses (optional)

To erase the Reseau marks from each image, run *remove_crosses*:

```
remove_crosses DZB*.tif
```

This will move the original images to a new directory, `original`, and save the erased images to the working directory.

resampling

After this step, you can use *resample_hexagon* to resample the images to the same size, and correct the distortion using the Reseau marks:

```
resample_hexagon DZB*.tif -s SCALE
```

where scale is the scale of the resampled image, in pixels per mm (default is 70). This will create a new file, `OIS-Reech_DZB*N001.tif` for each of the original images.

3.1.2 general micmac processing

Once the images have been re-sampled, the rest of the workflow is largely the same for both KH-9 images and scanned air photos.

tapioca

Once the images are re-sampled, run `mm3d Tapioca MulScale` to find tie points in downscaled versions of the images:

```
mm3d Tapioca MulScale "OIS.*tif" 400 1200
```

In the above command, the numbers after "OIS.*tif" are the size, in pixels, of the longest dimension of the downscaled image.

tapas

To find the relative orientation of the images, and calibrate the camera parameters, use `mm3d Tapas`:

```
mm3d Tapas RadialBasic "OIS.*tif" Out=Relative LibFoc=0
```

The `LibFoc=0` flag will keep the focal length fixed at the value provided in `MicMac-LocalChantierDescripteur.xml`.

relative dem and orthomosaic

Create a relative orthophoto and DEM using `mm3d Malt`:

```
mm3d Malt Ortho "OIS.*tif" Relative DirMEC=MEC-Relative NbVI=2 ZoomF=8 DefCor=0␣
↪CostTrans=1 EZA=1
```

If you have used an image mask, run the following command instead:

```
mm3d Malt Ortho "OIS.*tif" Relative DirMEC=MEC-Relative NbVI=2 MasqImGlob=filtre.tif␣
↪ZoomF=8 DefCor=0 CostTrans=1 EZA=1
```

By default, `mm3d Malt` only orthorectifies the individual images; to create an orthomosaic, use the `mm3d Tawny` command:

```
mm3d Tawny Ortho-MEC-Malt Out=Orthophotomosaic.tif RadiomEgal=0
```

Note: If the image is very large, you may need to run `mosaic_micmac_tiles.py` to combine the tiles into a single image. For the relative DEM, this will probably not be needed.

For the orthophoto, run the following from the `Ortho-MEC-Relative` directory:

```
mosaic_micmac_tiles.py -filename Orthophoto
```

registering the image

Note: This step requires a DEM and an orthoimage to find control points and estimate the absolute orientation of the images. In the examples below, I assume that these are named `DEM.tif` and `Landsat.tif`, respectively.

The main command to run here is *register_relative*:

```
register_relative MEC-Malt DEM.tif
```

Note: If you have a shapefile of the image footprints, use the `-footprints` flag; otherwise, they will be downloaded from USGS Earth Explorer:

```
register_relative MEC-Malt DEM.tif -footprints Footprints.shp
```

The shapefile should have a polygon for each image, with the name of the original image (minus the file extension) included in an `ID` column.

dem and orthomosaic

After the absolute orientation has been estimated by registering the image, run `mm3d Malt` again with this new orientation to extract the final DEM and orthophoto:

```
mm3d Malt Ortho "OIS.*tif" TerrainFirstPass DirMEC=MEC-Malt NbVI=2 ZoomF=1 DefCor=0␣
↪CostTrans=1 EZA=1
```

If you have used an image mask, run the following command instead:

```
mm3d Malt Ortho "OIS.*tif" TerrainFirstPass DirMEC=MEC-Malt NbVI=2 MasqImGlob=filtre.tif␣
↪ZoomF=1 DefCor=0 CostTrans=1 EZA=1
```

To generate an orthomosaic, run the following command:

```
mm3d Tawny Ortho-MEC-Malt Out=Orthophotomosaic.tif RadiomEgal=0
```

Note: If the image is very large, you may need to run `mosaic_micmac_tiles.py` to combine the tiles into a single image. For the DEM, run the following from the `MEC-Relative` directory:

```
mosaic_micmac_tiles.py
```

And for the orthophoto, run the following from the `Ortho-MEC-Relative` directory:

```
mosaic_micmac_tiles.py -filename Orthophoto
```

You can also run *post_process_micmac* to apply the AutoMask to the DEM and Orthomosaic, and georeference the correlation mask.

And that's it. You should now have an orthorectified KH-9 (or air photo) mosaic, and a DEM. Enjoy.

3.2 image pre-processing

This section describes a few of the different tools available for pre-processing scanned images.

Note: For working with KH-9 images, the reseau field re-sampling and image joining steps are required.

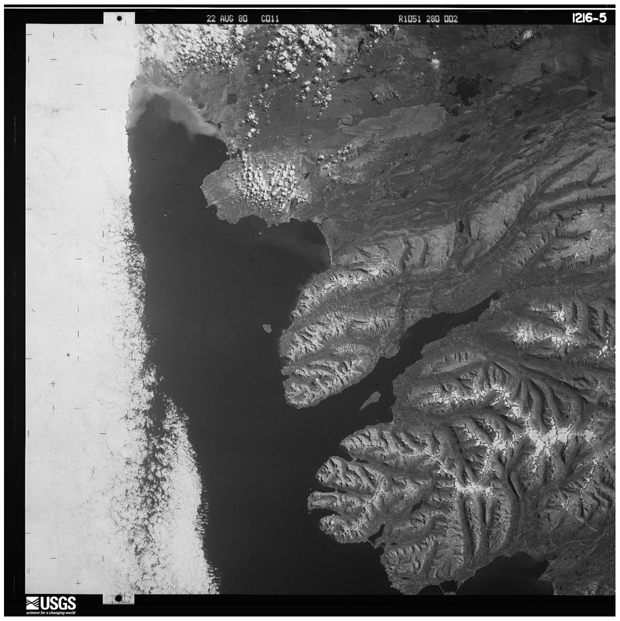
3.2.1 KH-9 pre-processing steps

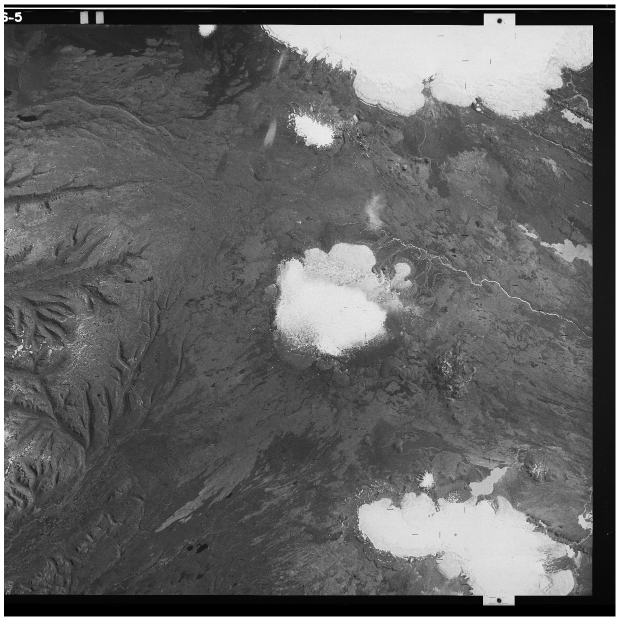
There are a number of necessary pre-processing steps when working with KH-9 imagery. Distortions in the film, caused by storage and other conditions, must be corrected to be able to accurately process the images. After resampling, it is also possible to remove the Reseau marks from the images, in order to improve the final results.

Finally, because of the size of the film (approx. 9"x18"), the images are scanned in two halves that must be joined together. And finally, it can also be helpful to improve contrast in the images, in order to help improve the final results.

image joining

Because of the large size of the film, USGS scans the images in two halves with a small amount of overlap, as shown in the example below.

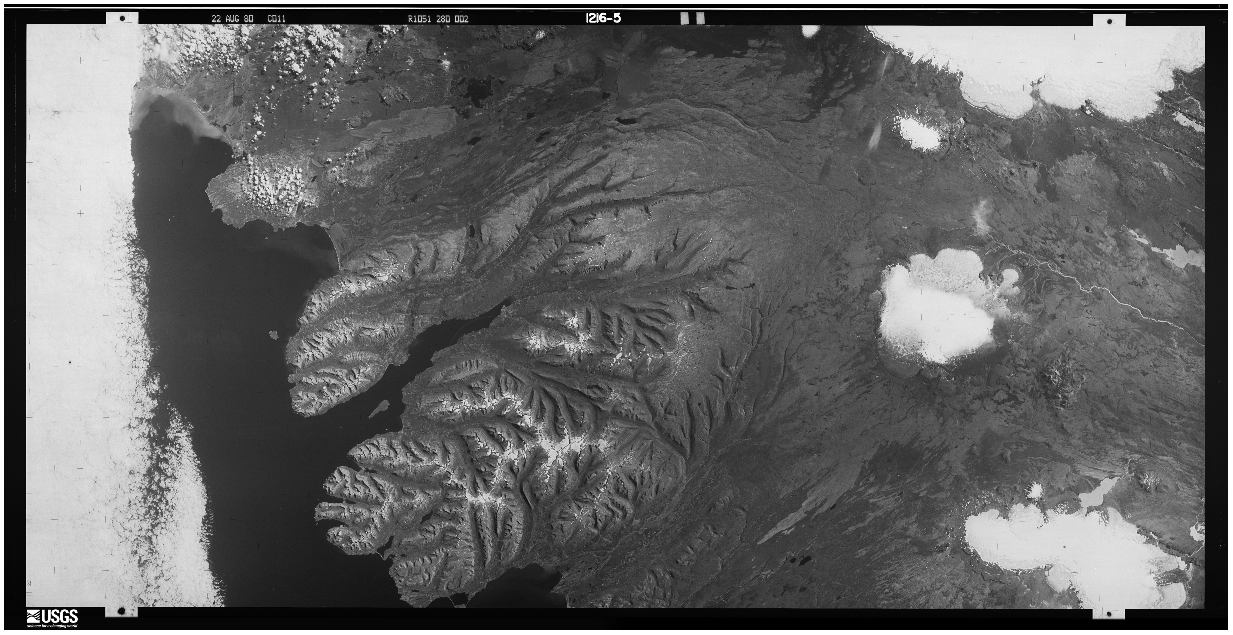




In spymicmac, the function to join the images is *spymicmac.image.join_hexagon()*, with a corresponding command-line tool *join_hexagon*.

Normally, the scans are labelled ‘a’ and ‘b’, with ‘a’ corresponding to the left-hand scan, and ‘b’ corresponding to the right-hand scan. This is what *spymicmac.image.join_hexagon()* is expecting - that the overlap between the two halves is the right-hand side of image ‘a’, and the left-hand side of image ‘b’.

After calling join_hexagon, the image should look something like this:

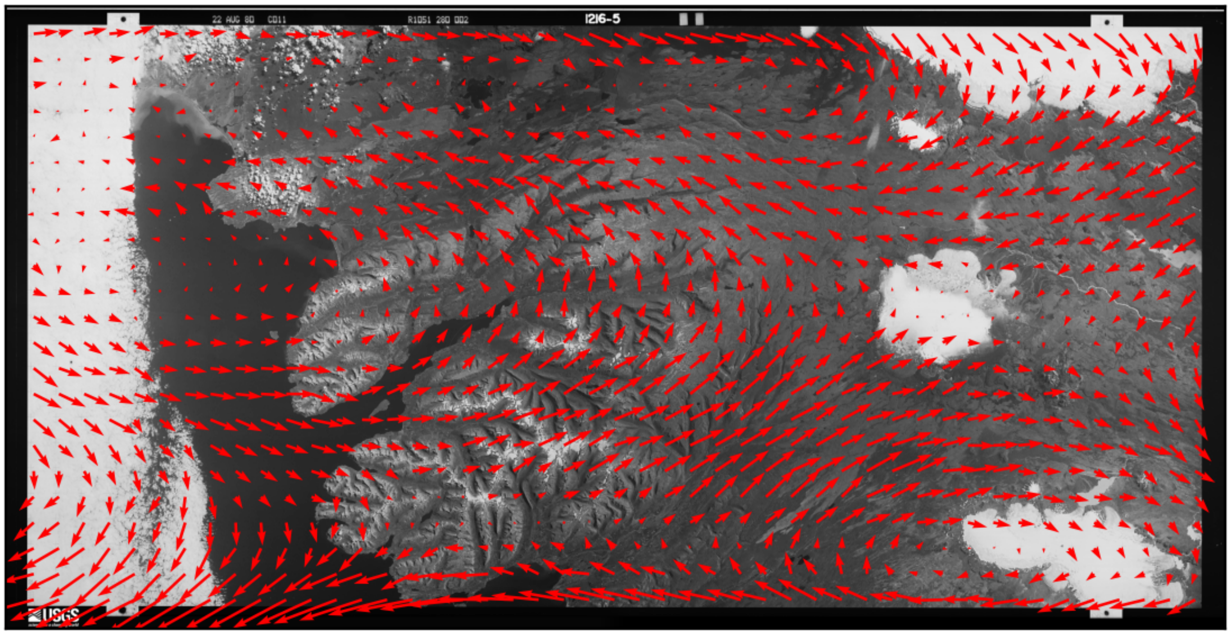


As there is sometimes a difference in brightness between the two halves, *spymicmac.image.join_hexagon()* has the option to blend the two halves over the overlap by averaging the values from the two halves, starting from 100% of the value of image 'a', linearly increasing to 100% of the value of image 'b' at the end of the overlapping part.

reseau field

To help correct some of the distortion in the images caused by film storage, *spymicmac.matching()* includes a routine to automatically find the Reseau markers in the image and use their locations to resample the images using *spymicmac.resample.resample_hex()*.

In the images below, you can see the difference between the expected location of each Reseau marker and the automatically detected locations:

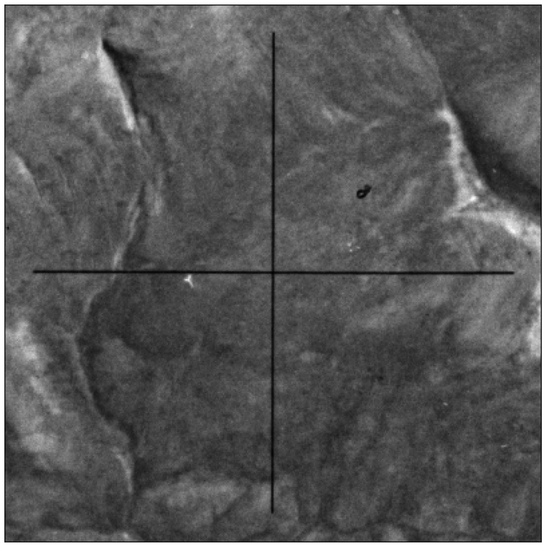


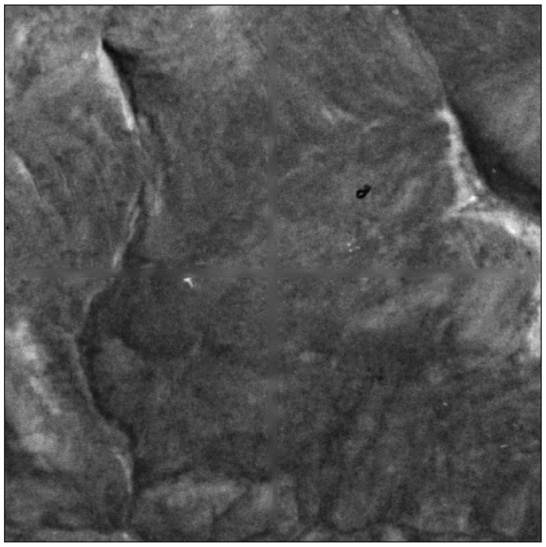
To run the routine, use either `spymicmac.matching.find_reseau_grid()` or *find_reseau_grid*. This will produce a `MeasuresIm` file that will be read by `spymicmac.resample.resample_hex()`.

Note: Before running `spymicmac.resample.resample_hex()`, you will also need to run *generate_micmac_measures* in order to generate the `MeasuresCamera.xml` file needed, then move `MeasuresCamera.xml` to the `Ori-InterneScan` directory in the correct folder.

cross removal

Once you have found the Reseau marks in each image half, you can "remove" the Reseau marks using either `spymicmac.matching.remove_crosses()` or *remove_crosses*.





After this step, you can use *resample_hexagon*.

Note: Because `mm3d ReSampFid` calculates an affine transform based on the fiducial marker locations, it does not actually correct the image using the marker locations. For KH-9 Mapping Camera images, it's better to use *resample_hexagon*.

contrast enhancement

Most of the scanned KH-9 images provided by USGS do not have issues with striping. However, they can still be low contrast, and it can help to use either of `spymicmac.image.stretch_image()` or `spymicmac.image.contrast_enhance()` for this.

For examples of these functions applied to a historical aerial image, see *contrast enhancement*.

3.2.2 air photo pre-processing

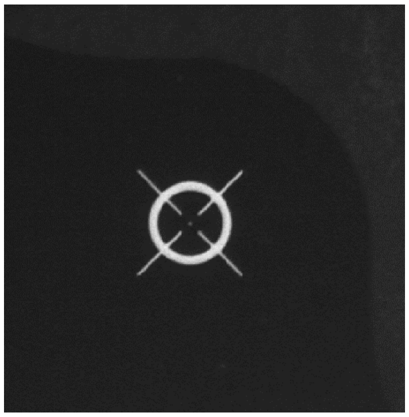
This section describes a few of the different tools available for pre-processing scanned images, including de-striping, contrast enhancement, and de-noising.

fiducial marks

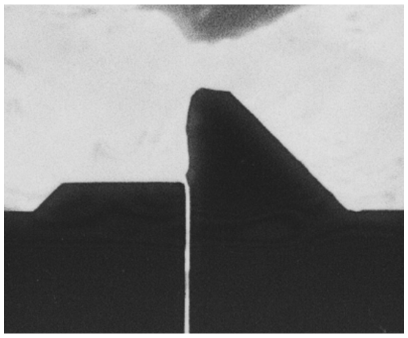
Once you have finished pre-processing the images, you need to resample them to a common geometry using mm3d ReSampFid. For historical aerial images, follow the steps below to find the fiducial markers for resampling.

SaisieAppuisInitQT

Depending on the camera setup, there will be between 4 and 8 (or more) fiducial markers. They may look like a crosshairs:



They may also be something like this:

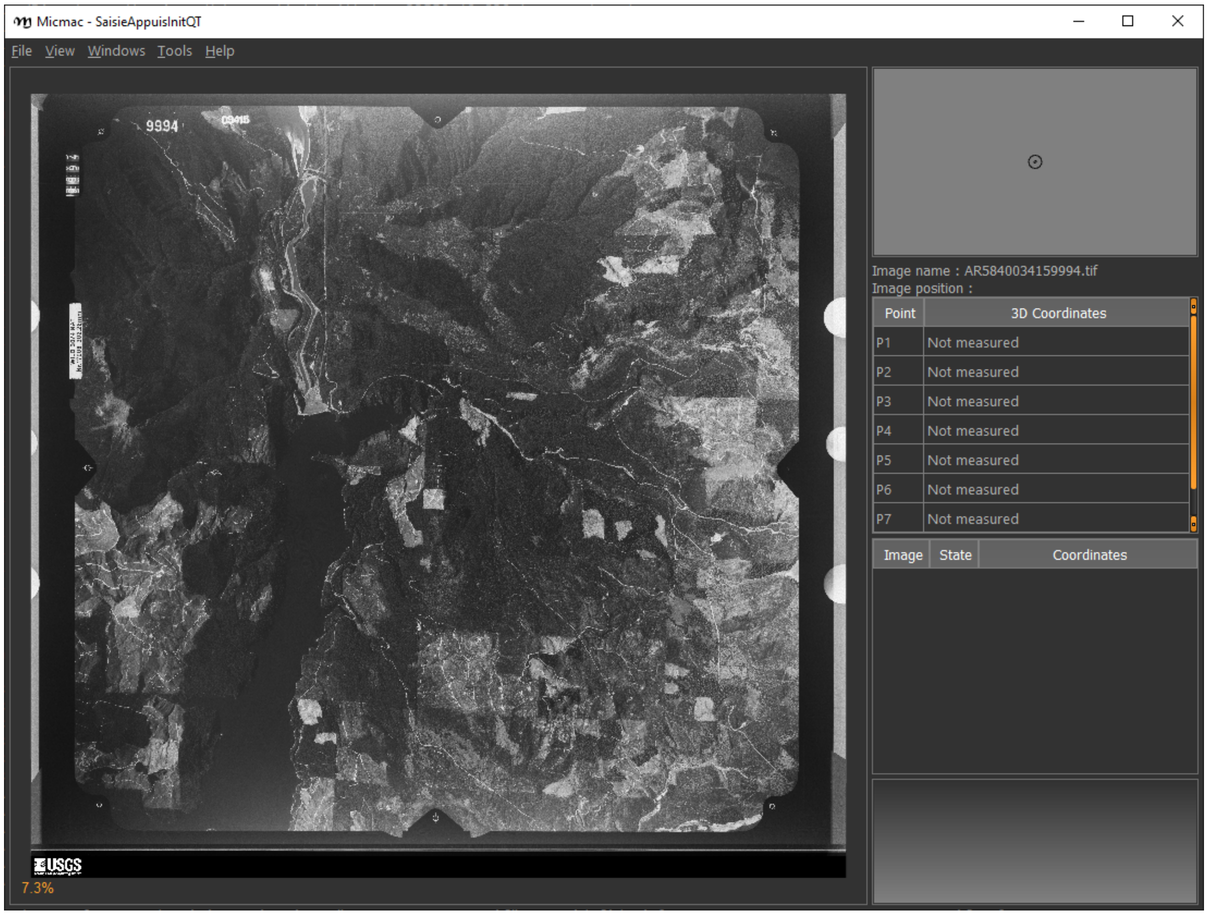


Or any number of other alternatives. To see examples of fiducial patterns for some common camera types, check out *example camera models*.

The MicMac program for inputting fiducial mark locations is `SaisieAppuisInitQT` (or just `SaisieAppuisInit` if you don't have the QT version set up). For each image, you'll need to run the following:

```
mm3d SaisieAppuisInitQT <Img> NONE id_fiducial.txt MeasuresIm-<Img>.xml
```

This will open a window like the following:



For each fiducial marker listed in `id_fiducial.txt`, you'll need to zoom in on the marker, click the name in the table on the right, and then click on the fiducial marker in the main window. If necessary, you can also move the marker once you've placed it (because we all make mistakes).

Note: `id_fiducial.txt` is a simple text file with the 'name' of each fiducial mark listed on each line. The names of the fiducial markers in `id_fiducial.txt` should match the names in `MeasuresCamera.xml`.

See `here` for an example.

Once you have selected the location for each fiducial marker, select `File > Exit` to save the point locations to `MeasuresIm-<Img>.xml`.

Note that this will actually create two files, `MeasuresIm-<Img>-S2D.xml` and `MeasuresIm-<Img>-S3D.xml`. As these are only two-dimensional points, you can discard the `S3D.xml` file. You'll need to move the `S2D.xml` file to a new folder, `Ori-InterneScan`, and rename it to remove the `-S2D` part of the name. In other words:

```
rm MeasuresIm-AR5840034159994.tif-S3D.xml
mkdir Ori-InterneScan
mv MeasuresIm-AR5840034159994.tif-S2D.xml Ori-InterneScan/MeasuresIm-AR5840034159994.tif.
↪xml
```

You can also use the shell script `move_fiducial_xml.sh`, which will do this for any file matching the pattern `MeasuresIm*S2D.xml`.

Kugelhupf

If you have a number of images, and the fiducial marks are in approximately the same place, you might want to give `mm3d Kugelhupf` a try:

```
Kugelhupf (Klics Ubuesques Grandement Evites, Lent, Hasardeux mais Utilisable pour␣
↪Points Fiduciaux): Automatic fiducial point determination
*****************************
*  Help for Elise Arg main  *
*****************************
Mandatory unnamed args :
  * string :: {Pattern of scanned images}
  * string :: {2d fiducial points of an image}
Named args :
  * [Name=TargetHalfSize] INT :: {Target half size in pixels (Def=64)}
  * [Name=SearchIncertitude] INT :: {Search incertitude in pixels (Def=5)}
  * [Name=SearchStep] REAL :: {Search step in pixels (Def=0.5)}
  * [Name=Threshold] REAL :: {Limit to accept a correlation (Def=0.90)}
```

As an example:

```
mm3d Kugelhupf AR5840034159994.tif Ori-InterneScan/MeasuresIm-AR5840034159994.tif.xml
```

This command will take the locations from the MeasuresIm file specified by the second argument and search any of the remaining images using template matching, to try to find their locations automatically.

Note that it does not always work, especially for images where the fiducial mark is a dot rather than a crosshair or target.

Once you have all of the image points, you can move on to the next step: re-sampling the images using `ReSampFid`.

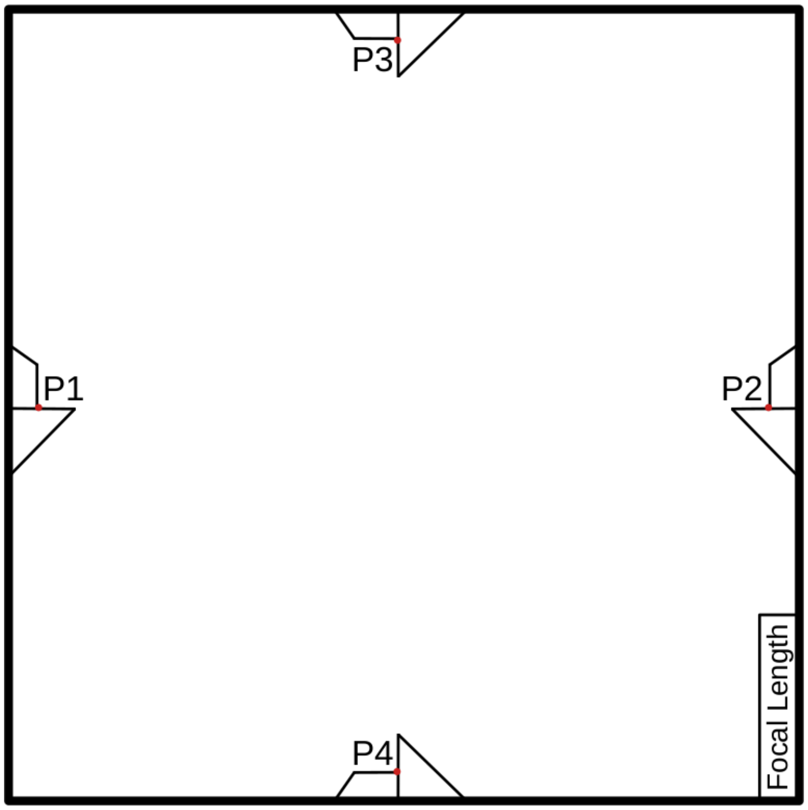
example camera models

This section is a list of different camera models that have acquired historical aerial photos, what their fiducial marks look like, and the approximate coordinates of the fiducial marks that you can use to populate the `MeasuresCamera.xml` file.

Note: If you have a calibration report that corresponds to your specific images, **you should use that instead**.

The information provided here is for those cases where a calibration report does not exist, or has been lost to time.

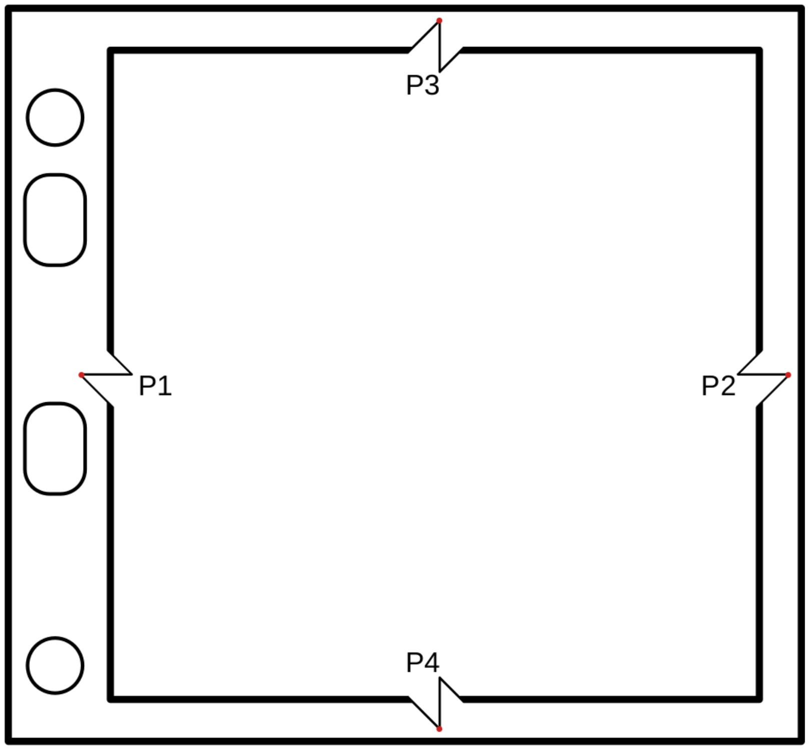
Fairchild F224, K17B (Metrogon Lens)



	x	y
P1	1	113
P2	225	113
P3	113	1
P4	113	225

Note: These are the approximate coordinates for the corners of the fiducial marker, as these tend to be more stable than the tips of the "wings"

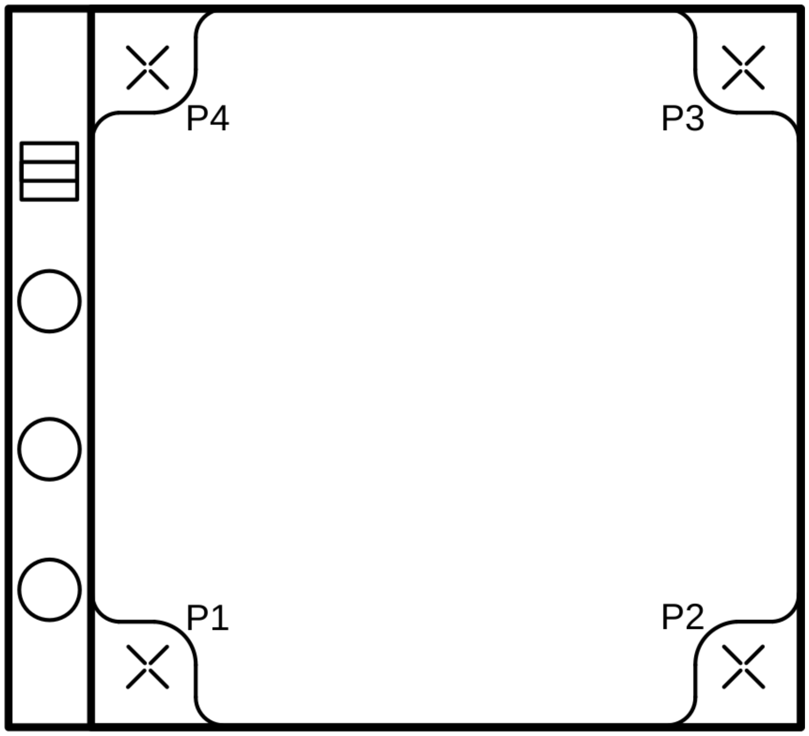
Fairchild KC-1, KC-1B, T-11 (Metrogon Lens)



	x	y
P1	0	117.5
P2	238	117.5
P3	120	0
P4	120	235

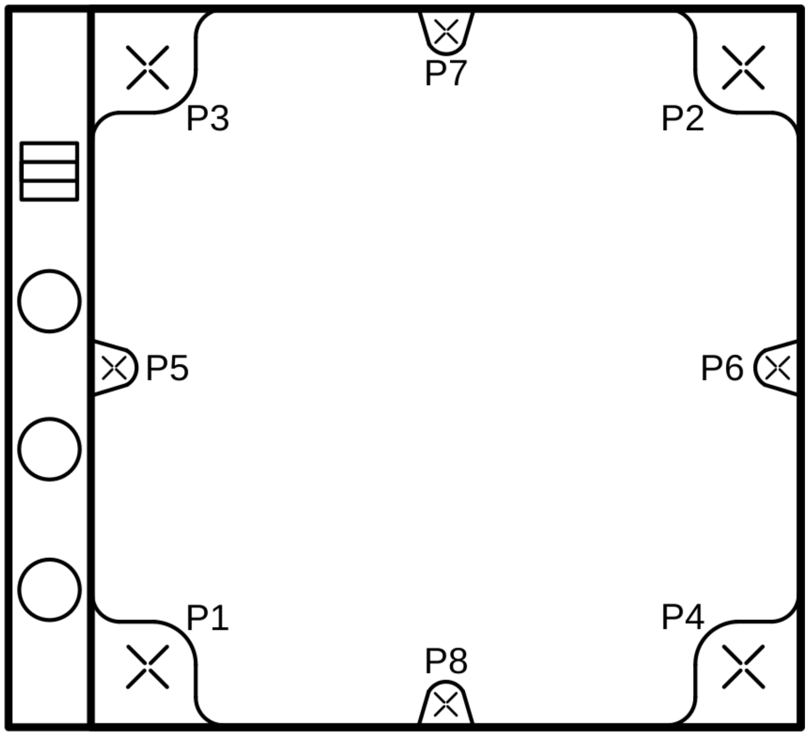
Note: These cameras may have different fiducial patterns.

Wild RC5, RC8 (Aviogon Lens)



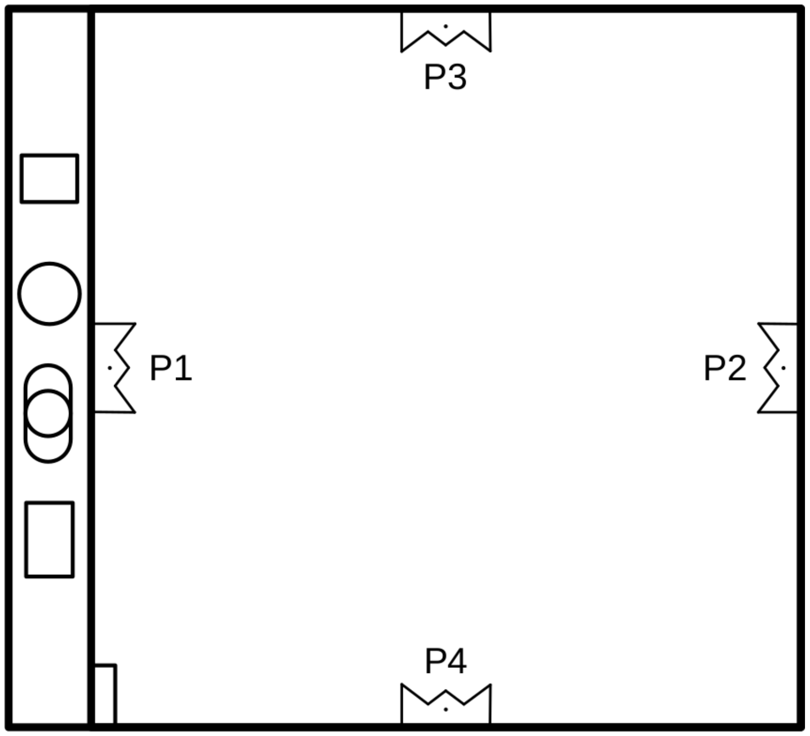
	x	y
P1	212	0
P2	212	212
P3	0	212
P4	0	0

Wild RC10 (Aviogon Lens)



	x	y
P1	4	216
P2	216	4
P3	4	4
P4	216	216
P5	0	110
P6	220	110
P7	110	0
P8	110	220

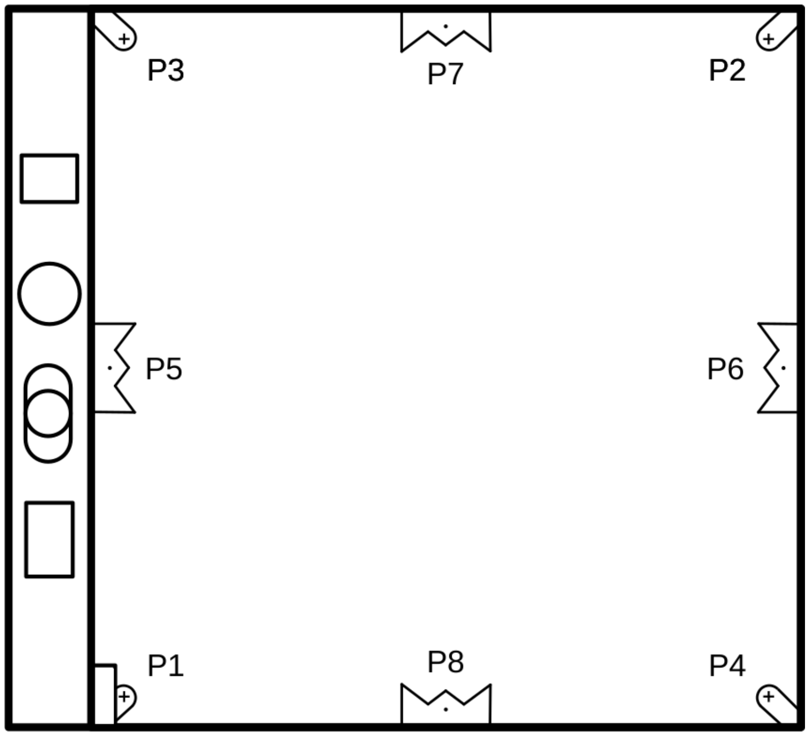
Zeiss RMK 15/23 (Pleogon Lens)



	x	y
P1	0	113
P2	226	113
P3	113	0
P4	113	226

Note: The coordinates above correspond to the center of the small dot near the tip of the fiducial marker.

Zeiss RMK A 15/23 (Pleogon Lens)



	x	y
P1	9	217
P2	217	9
P3	9	9
P4	217	217
P5	0	113
P6	226	113
P7	113	0
P8	113	226

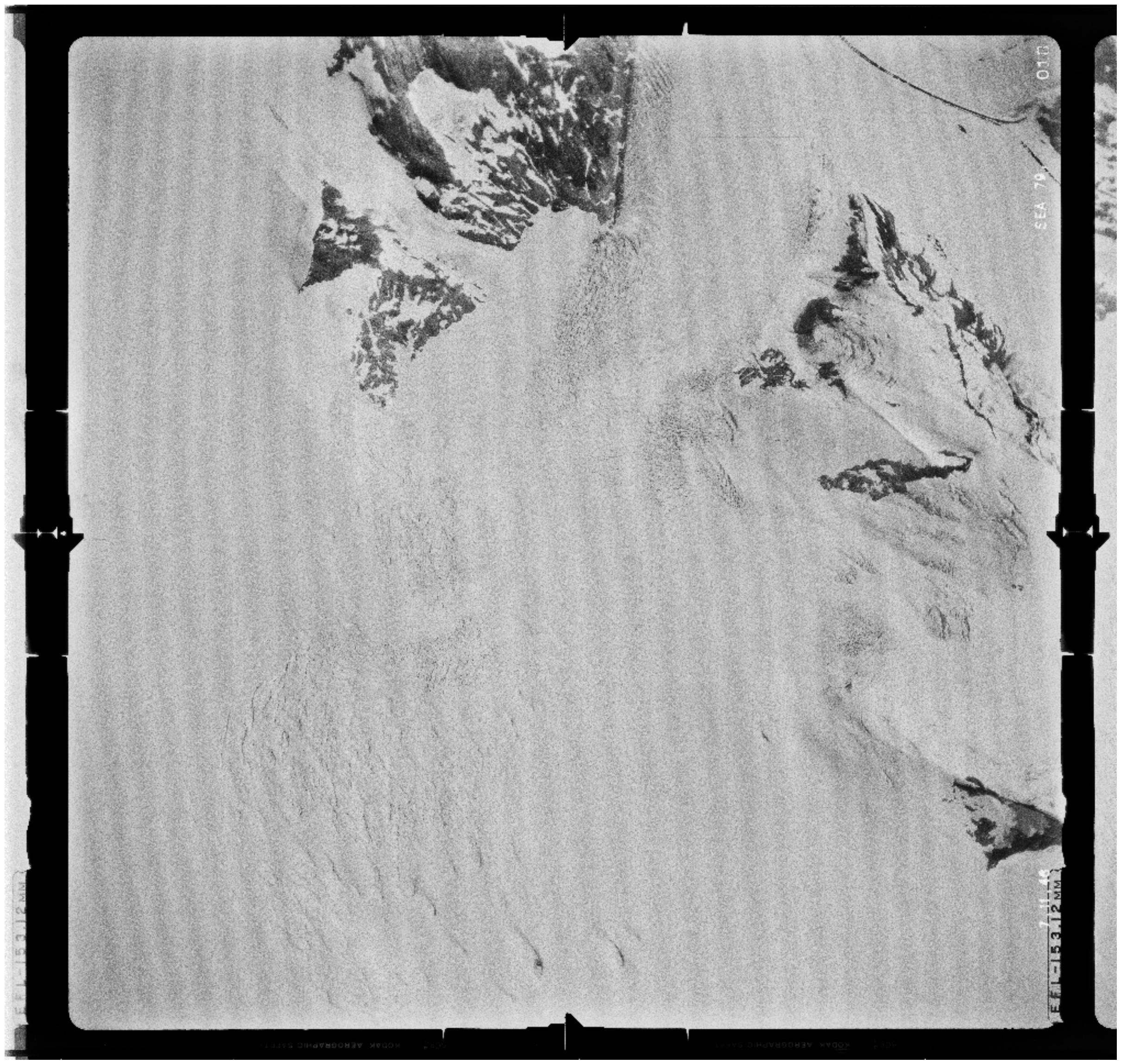
Note: The coordinates for P5-P8 above correspond to the center of the small dot near the tip of the fiducial marker.

historic air photo preprocessing

It may be worthwhile to perform some different pre-processing steps on scanned historic air images. For example, it can help remove artefacts or increase contrast in images, in order to help improve both the visual results, as well as the correlation and DEM extraction.

image de-striping

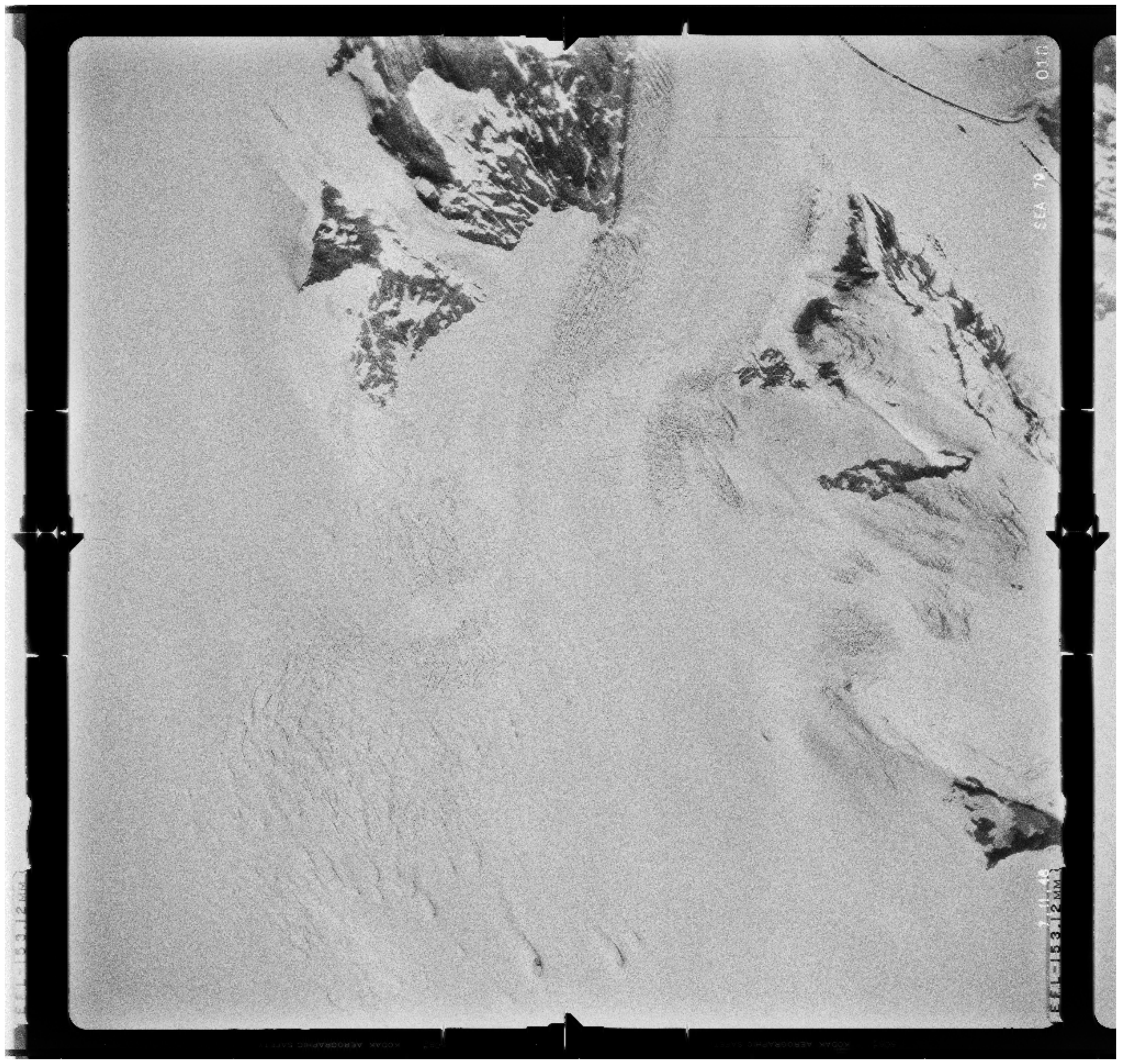
Occasionally, scanned images will have artefacts, such as the prominent striping shown here:



The *spymicmac.image()* function *spymicmac.image.remove_scanner_stripes()*, based on a technique described in [Crippen1989] (pdf), can be used to remove these stripes - note the use of `scan_axis=1` to indicate a scan from left to right, rather than top to bottom:

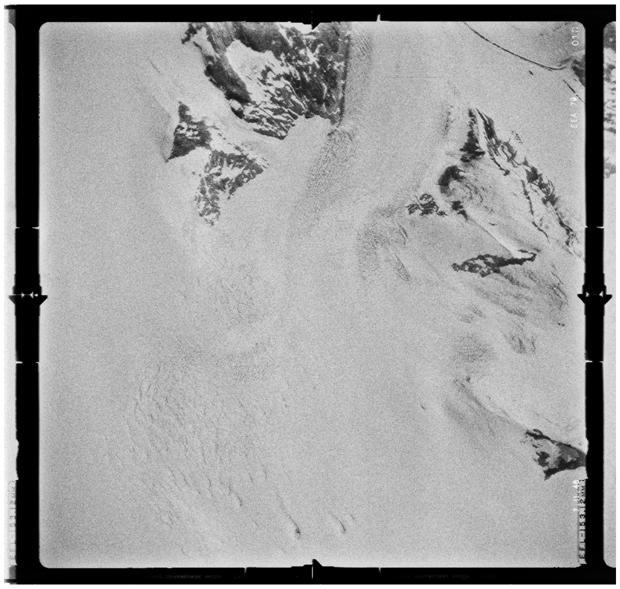
```
from spymicmac.image import remove_scanner_stripes
filtered = remove_scanner_stripes(img, scan_axis=1)
```

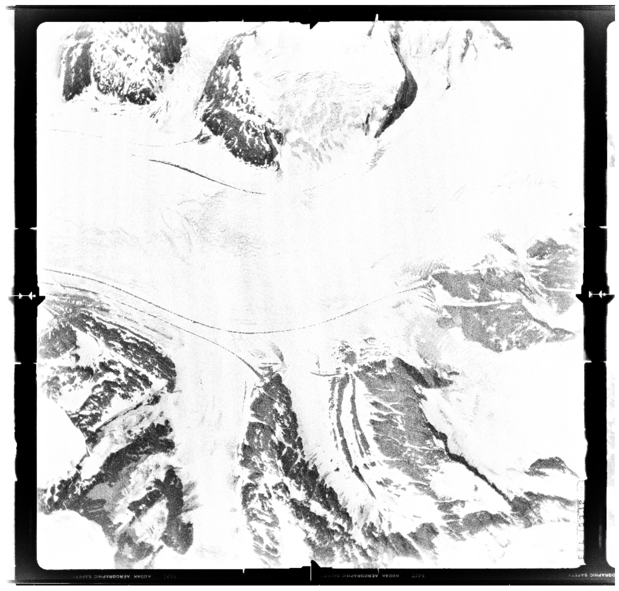
The result shows a significant reduction in the striping:



contrast enhancement

Another common issue with scanned aerial images is that they can have inconsistent brightness between different images:

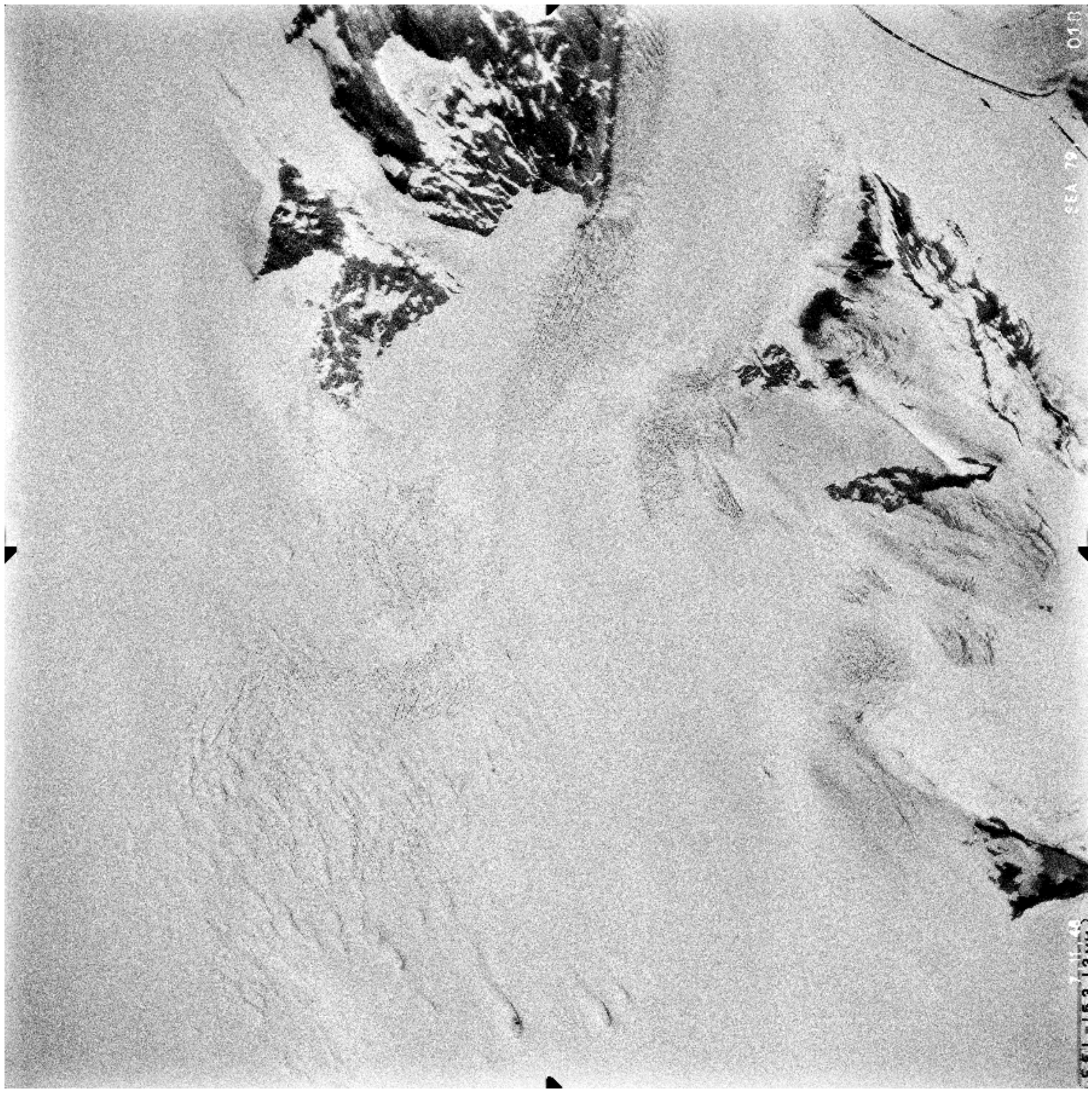




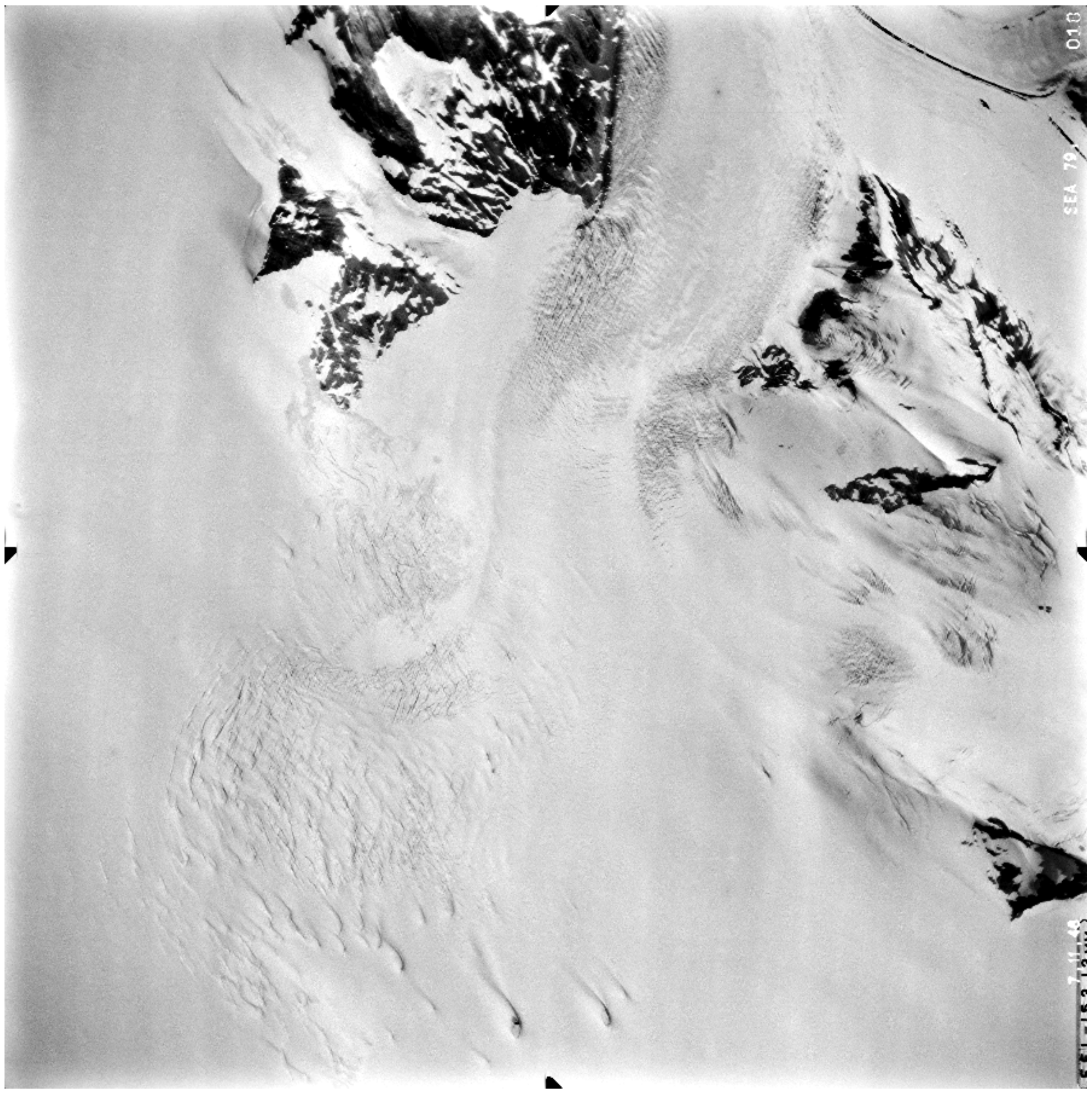
spymicmac has two main functions available for enhancing contrast: *spymicmac.image.stretch_image()* and *spymicmac.image.contrast_enhance()*.

stretch_image() performs a linear contrast stretch on the image to a given quantile, while contrast_enhance() performs a median filter to de-noise, before calling stretch_image() and performing a gamma adjustment on the stretched image.

For the image on the left above, here is the result of applying stretch_image() clipped to (0.01, 0.99) - that is, 1% and 99% of the image values:



And here is the result using `contrast_enhance()` (note that this also enhances the residual striping that was not corrected earlier):



Different images/surveys may require different levels of contrast enhancement - for example, it may not be advisable to perform this kind of contrast enhancement on images that are mostly bright snow, as this will primarily enhance noise in the image.

de-noising

In many cases, there may also be some noise in the images - this can be seen above, for example. One way to reduce this noise is to use a median filter, similar to what is done in *spymicmac.image.contrast_enhance()*:

```
from skimage.filters import median
from skimage.morphology import disk
filtered = median(img, selem=disk(3))
```

Here, a smaller filter (max size 3x3) will help to remove the salt-and-pepper noise, while preserving most of the features.

3.3 general workflow

The workflow and steps here are built partly from the steps outlined in the excellent Historical Orthoimage tutorial, written by Luc Girod. It has been modified to include specific *KH-9 pre-processing steps*, as well as optional *air photo pre-processing* steps for historic air photos.

The other main difference is the use of *spymicmac.register* to find control points automatically, using an orthorectified image and external DEM.

3.3.1 micmac processing files

There are a number of files that you will need to process your images using MicMac, which are detailed below. If you are working with KH-9 images, *generate_micmac_measures* will automatically create the necessary files - you only need to move `MeasuresCamera.xml` into the `Ori-InterneScan` directory.

You can use the provided *MicMac example files* as a starting point, though note that you will probably need to make some modifications as detailed below.

MicMac-LocalChantierDescripteur.xml

Note: For KH-9 Hexagon Mapping Camera images, you can use `spymicmac.micmac.create_localchantier_xml()` or *create_localchantier_xml* to generate this file.

Alternatively, you can `download` and edit the example provided.

You will need to set the image size (in mm) by changing the values in `SzCaptMm` under `CameraEntry` (line 8 of the example). You will also need to set the image matching pattern (`PatternTransform`, line 29 of the example) and focal length in mm (`CalcName`, line 30) for each set of images - if, for example, different focal lengths were used.

To improve tie point density and matching, especially in low-contrast images, you can try copying the block below into your `MicMac-LocalChantierDescripteur.xml` file:

```
<KeyedNamesAssociations>
    <Calcs>
         <Arrite>  1 1 </Arrite>
         <Direct>
               <PatternTransform> .*  </PatternTransform> <!-- Regular expressions of␣
↪the group of images with the following camera model -->
               <CalcName> SFS </CalcName> <!-- Name of the camera for these images -->
         </Direct>
     </Calcs>
     <Key> NKS-Assoc-SFS </Key>
</KeyedNamesAssociations>
```

Note: Be sure that when you paste the block, you paste it so that it is in between the `ChantierDescripteur` tags (lines 2, 36 in the provided example file), and also not within one of the existing `KeyedNamesAssociations` blocks. (i.e., paste it at line 23 of the provided example file).

MeasuresCamera.xml

Warning: This file must be placed in a directory called `Ori-InterneScan`

Ideally, you will have a camera calibration report, that will tell you the location of the different fiducial markers in the image geometry. Note that using `ReSampFid` **requires** a file, `Ori-InterneScan/MeasuresCamera.xml`, that tells MicMac what the location of each fiducial mark is.

Note: The image coordinates are defined with the origin in the upper left corner, rather than the center of the image used by most calibration files. You can translate from one system to the other with the following:

```
xp = x - min(x)
yp = (-y) - min(-y)
```

If you do not have a calibration report for your particular camera, you can have a look at some *example camera models* for approximate locations of fiducial markers.

Rather than editing the `MeasuresCamera.xml` file with the fiducial marker locations, you can also put the fiducial marker locations into a CSV file, then use *`spymicmac.micmac.create_measurescamera_xml()`* to convert this create `Ori-InterneScan/MeasuresCamera.xml`.

id_fiducial.txt

This is just a plain text file, with the "names" of the different fiducial marks:

```
P1
P2
P3
```

… and so on.

Note: The names in the file should match the names written in `MeasuresCamera.xml`.

file structure

Before starting, your file structure should look something like this:

```
project
├── id_fiducial.txt
├── Img1.tif
├── Img2.tif
...
├── MicMac-LocalChantierDescripteur.xml
├── Ori-InterneScan
│   └── MeasuresCamera.xml
```

Once you have this set up, you can work on the preprocessing steps.

3.3.2 re-sampling the images

air photos

After you have found each of the fiducial marks in each image and generated a MeasuresIm file for each image, either by hand or using `mm3d Kugelhupf`, you can run `ReSampFid`:

```
*****************************
*  Help for Elise Arg main  *
*****************************
Mandatory unnamed args :
  * string :: {Pattern image}
  * REAL :: {Resolution of scan, mm/pix}
Named args :
  * [Name=BoxCh] Box2dr :: {Box in Chambre (generally in mm, [xmin,ymin,xmax,ymax])}
  * [Name=Kern] INT :: {Kernel of interpol,0 Bilin, 1 Bicub, other SinC (fix size of␣
↪apodisation window), Def=5}
  * [Name=AttrMasq] string :: {Atribut for masq toto-> toto_AttrMasq.tif, NONE if unused,
↪ Def=NONE}
  * [Name=ExpAff] bool :: {Export the affine transformation}
```

For example, to re-sample the images to 14 microns (0.014 mm):

```
mm3d ReSampFid "AR5.*tif" 0.014
```

The re-sampled images will have OIS-Reech_ appended to the filename:

```
AR5840034159994.tif -> OIS-Reech_AR5840034159994.tif
```

These are the images that you will use for the remaining steps - you might want to create a new folder to place the original images.

kh-9 hexagon mapping camera

To resample KH-9 Hexagon Mapping Camera images, use either *`spymicmac.resample.resample_hex()`* or *resample_hexagon*.

next step

The next step is to find tie points using `Tapioca`.

3.3.3 computing tie points

The basic command for computing tie points is Tapioca. If you don't have very many images, you can use `Tapioca` to find the tie points by matching all pairs of images:

```
mm3d Tapioca MulScale "OIS.*tif" 400 1200
```

kh-9 hexagon mapping camera

When working with KH-9 images, the single Tapioca call will usually be sufficient, and you can move on to the next step: *finding the relative orientation*.

air photos

the neighbours file

With a large number of images, this will be a very slow process. If you have vector data (e.g., a shapefile) of the image footprints, you can use *spymicmac.micmac.write_neighbour_images()* to narrow down the number of image pairs where Tapioca will search for pairs. This will create a file, FileImagesNeighbour.xml, that specifies which images overlap based on their footprints.

For more modern images where more precise location information is available, you can also use the OriConvert tool:

```
mm3d OriConvert OriTxtInFile GPS_sommets.txt Sommets NameCple=FileImagesNeighbour.xml
```

Then, you can run Tapioca using the File option:

```
mm3d Tapioca File FileImagesNeighbour.xml 1200
```

creating a mask

You can also create a mask to filter out tie points that are created due to the presence of fiducial marks or inscriptions on the images. The basic command for this is:

```
mm3d SaisieMasqQT "OIS-Reech_<Img>.tif"
```

A slightly more detailed instructional video can be found here. Once you have created the mask, be sure to rename the file:

```
mv OIS-Reech_<Img>_Masq.tif filtre.tif
```

filtering tie points

Once you have created a mask, you can use it to filter tie points:

```
mm3d HomolFilterMasq "OIS.*tif" GlobalMasq=filtre.tif
```

Once this is done, you can move on to computing the relative orientation using Tapas.

3.3.4 finding the relative orientation

The basic tool for computing relative orientation in MicMac is Tapas:

```
mm3d Tapas <CameraModel> <Pattern>
```

MicMac has a number of camera models. For many applications, a basic radial distortion model, RadialBasic, will probably suffice. This model is especially useful in cases where models with additional degrees of freedom, such as RadialExtended, are likely to diverge owing to issues with distortion in the images.

The basic syntax used for `RadialBasic` is:

```
mm3d Tapas RadialBasic <Pattern> Out=<OutDir> SH=HomolMasqFiltered LibFoc=0
```

Here, it might help to use `LibFoc=0` (i.e., hold the focal length fixed at the value specified in `MicMac-LocalChantierDescripteur.xml`), as the calibration for older images can often be unstable.

With a large number of images, it might help to create an initial calibration based on a few "nice" images (i.e., plenty of tie points/contrast) before trying to run the calibration on the entire set of images:

```
mm3d Tapas RadialBasic "<Image>(1-5).tif" Out=CalibInit SH=HomolMasqFiltered LibFoc=0
```

This will create an initial directory, `Ori-CalibInit`, based on the images specified by the search pattern. Once you have a stable initial calibration, you can use this to seed the calibration for the entire block of images:

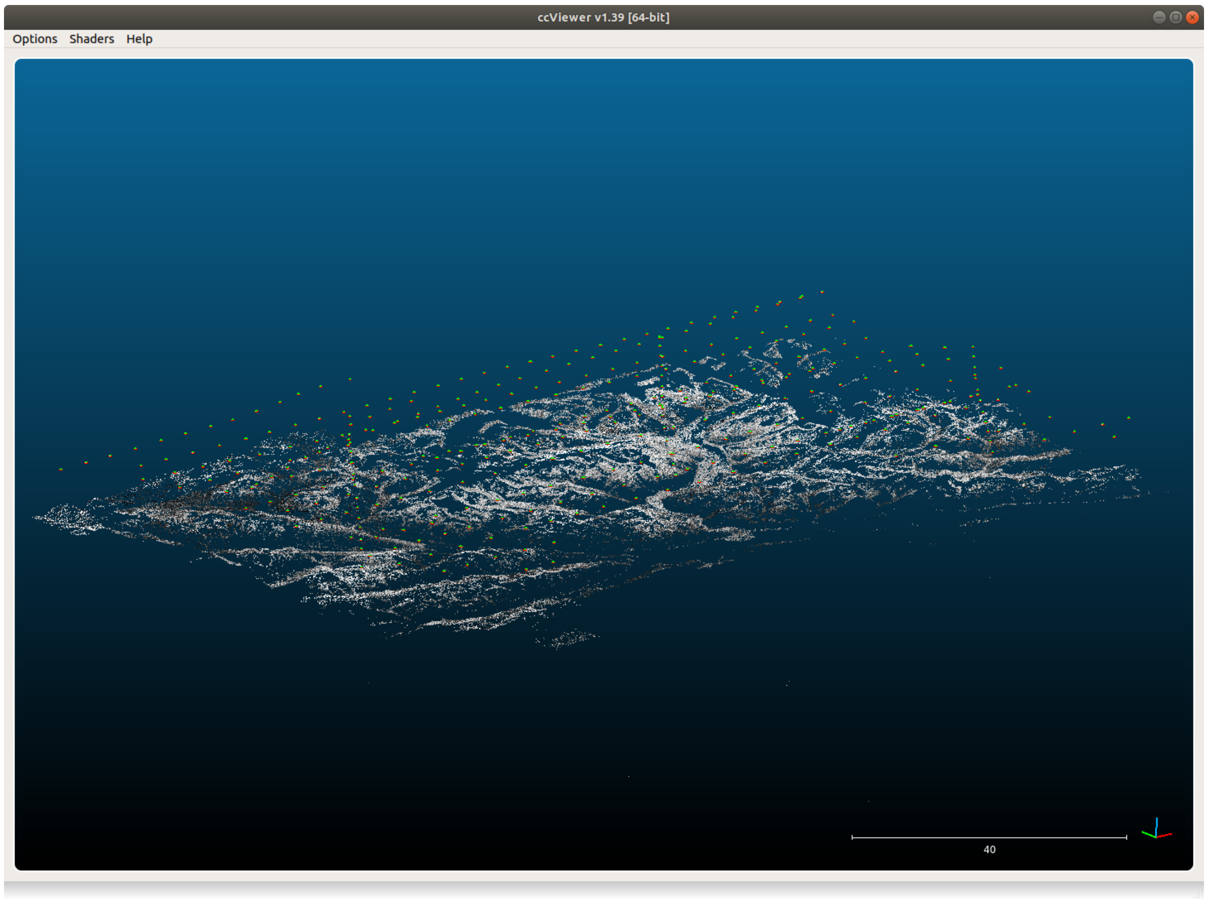
```
mm3d Tapas RadialBasic "OIS.*tif" InCal=CalibInit Out=Relative SH=HomolMasqFiltered␣
↪LibFoc=0
```

This will create a new directory, `Ori-Relative`, that contains orientation files for each of the different images in the directory, as well as the calibration for each "camera" specified in `MicMac-LocalChantierDescripteur.xml`.

If `Tapas` successfully completes, you can then create a point cloud to visualize the relative orientation and inspect it for any errors:

```
mm3d AperiCloud "OIS.*tif" Relative SH=HomolMasqFiltered
```

You can then open the `.ply` file using, for example, Cloud Compare or Meshlab:



fixing the orientation

If your *Tapas* output looks okay, you can move on to the next step, *computing the relative orthophoto*.

If you are unlucky, however, there are some tools in `spymicmac.orientation()` to help manipulate the orientation files to help `Tapas` converge. For example, occasionally cameras will be positioned in an incorrect location, especially images with lots of ice/snow.

If the absolute camera positions are (approximately) known, `spymicmac.orientation.fix_orientation()` will estimate an affine transformation between the known absolute positions and the relative positions estimated by Tapas.

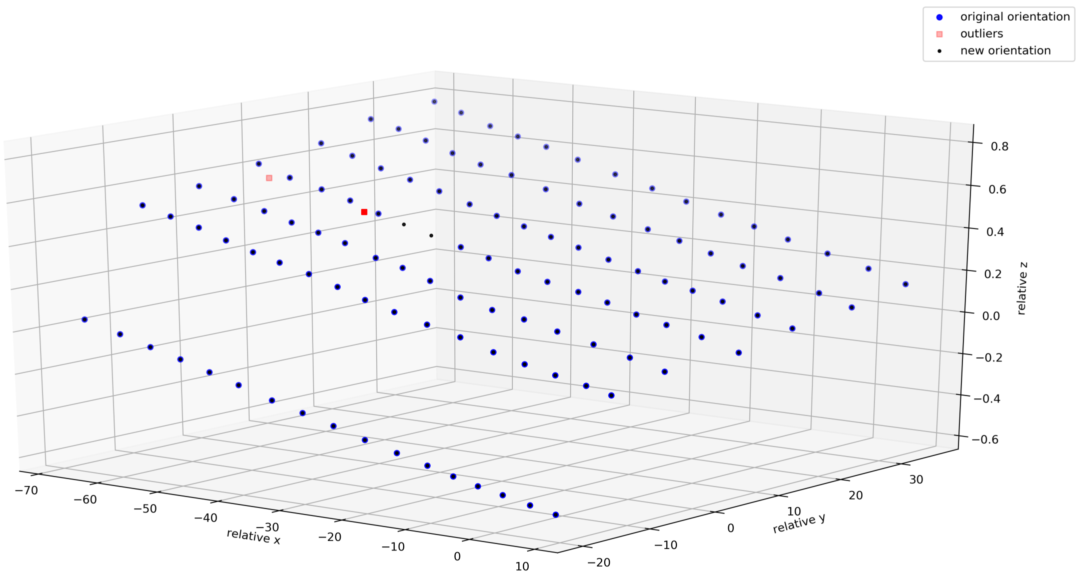
Outliers are identified by comparing the normalized median absolute deviation (NMAD) of the residuals, and the camera positions in the orientation file are overwritten with the position estimated from the transformation using `spymicmac.orientation.update_center()`.

Note: Once you have updated the center locations with the new estimated positions, you should re-run `Tapas` using `InOri=<Updated Orientation>`:

```
mm3d Tapas RadialBasic "OIS.*tif" InOri=Relative Out=Relative LibFoc=0
```

In most cases (but not always!), seeding the locations in this way will help `Tapas` converge to a more accurate solution.

In the example shown below, two images (marked with red squares) have been identified as outliers using the estimated transformation. The positions have been updated using *spymicmac.orientation.fix_orientation()*, and re-running `Tapas` has helped position the cameras correctly:



If the camera positions are not well-known (often the case for historic air photos), you can use *spymicmac.orientation.interp_line()* or *spymicmac.orientation.extend_line()* to estimate the positions based on an assumed flight line, using positions that have converged properly.

Once you have the new positions estimated, you should update the positions in the orientation files using *spymicmac.orientation.update_center()*, and re-run `Tapas` as shown above.

3.3.5 computing the relative orthophoto

Once you have the relative orientation, you can use Malt to compute a relative DEM and orthoimages:

```
mm3d Malt Ortho "OIS.*tif" Relative DirMEC=MEC-Relative NbVI=2 MasqImGlob=filtre.tif␣
↪ZoomF=8 DefCor=0 CostTrans=1 EZA=1
```

This will run `Malt` on all of the images, using the orientation described in `Ori-Relative`. `Malt` defaults to only running where 3 or more images are visible (`NbVI=3`), but it is usually fine to go with 2 images. At this stage, we don't need the DEM to be processed to full resolution - a lower-resolution version (`ZoomF=4` or `ZoomF=8`) will suffice. The `EZA=1` argument ensures that the values in the DEM are in the correct units.

Once this command finishes, you will have two new directories: `MEC-Relative` and `Ortho-MEC-Relative`. The DEM and associated correlation masks are found in `MEC-Relative`, while the orthophotos are found in `Ortho-MEC-Relative`.

creating the orthomosaic using Tawny

Note that the orthoimages are not mosaicked - they are just the individual images orthorectified using the extracted DEM. To generate an orthomosaic, we use Tawny:

```
mm3d Tawny Ortho-MEC-Relative Out=Orthophotomosaic.tif RadiomEgal=0
```

Here, we use `RadiomEgal=0` to use the images as-is, rather than attempting to balance the radiometry (as this can lead to undesirable results). Finally, you might need to re-combine the image tiles using *`spymicmac.micmac.mosaic_micmac_tiles()`* (or *mosaic_micmac_tiles*) depending on how large they are:

```
mosaic_micmac_tiles Orthophotomosaic -imgdir Ortho-MEC-Relative
```

Once this is complete, you can move on to the next step: registering the orthoimage and automatically finding control points using an external DEM and satellite image.

3.3.6 automatically finding control points

At this point, you should have successfully run `mm3d Malt` and `mm3d Tawny` to generate a relative orthophoto and DEM. You should also have run *`spymicmac.micmac.mosaic_micmac_tiles()`* (or *mosaic_micmac_tiles*) if needed - otherwise, *`spymicmac.register.register_relative()`* (or *register_relative*) will most likely fail.

In order to run *`spymicmac.register.register_relative()`*, you will need a number of files, detailed in the section below. After that, this document will describe the process that *`spymicmac.register.register_relative()`* uses to find GCPs and iteratively refine the orientation.

necessary files

reference orthoimage and DEM

At the risk of stating the obvious, the reference orthoimage and DEM should cover your study area. The reference orthoimage can be a high-resolution orthomosaic, or it can be a comparatively low-resolution satellite image. *`spymicmac.register.register_relative()`* has been tested on both Landsat-8 and Sentinel-2 images, with satisfactory results for both.

In general, the results will depend in part on the accuracy of the control points - so if your reference orthoimage is highly accurate and of a similar resolution to your air photos, the more accurate the result will be.

Similar rules apply for the reference DEM - the more accurate the DEM, the better the final result.

image footprints

This should be a vector dataset (e.g., shapefile, geopackage - anything that can be read by geopandas.read_file). The footprints do not have to be highly accurate - most of the routines in *`spymicmac.register()`* have been developed using USGS datasets/metadata, which are only approximate footprints.

The main use for the footprints in *`spymicmac.register.register_relative()`* is in the call to *`spymicmac.orientation.transform_centers()`*, which uses RANSAC to estimate an affine transformation between the footprint centroids and the relative camera centers estimated by `mm3d Tapas`.

As long as the distribution of the footprints/centroids is approximately correct, this step should work according to plan.

Note: Some USGS Aerial Photo Single Frames (as well as the KH-9 Hexagon images) have footprints that are incorrectly placed, or the images have been scanned but not georeferenced. In this case, you may need to create your own footprints. It is most likely worth checking the metadata for these images before you start working.

exclusion and inclusion masks (optional)

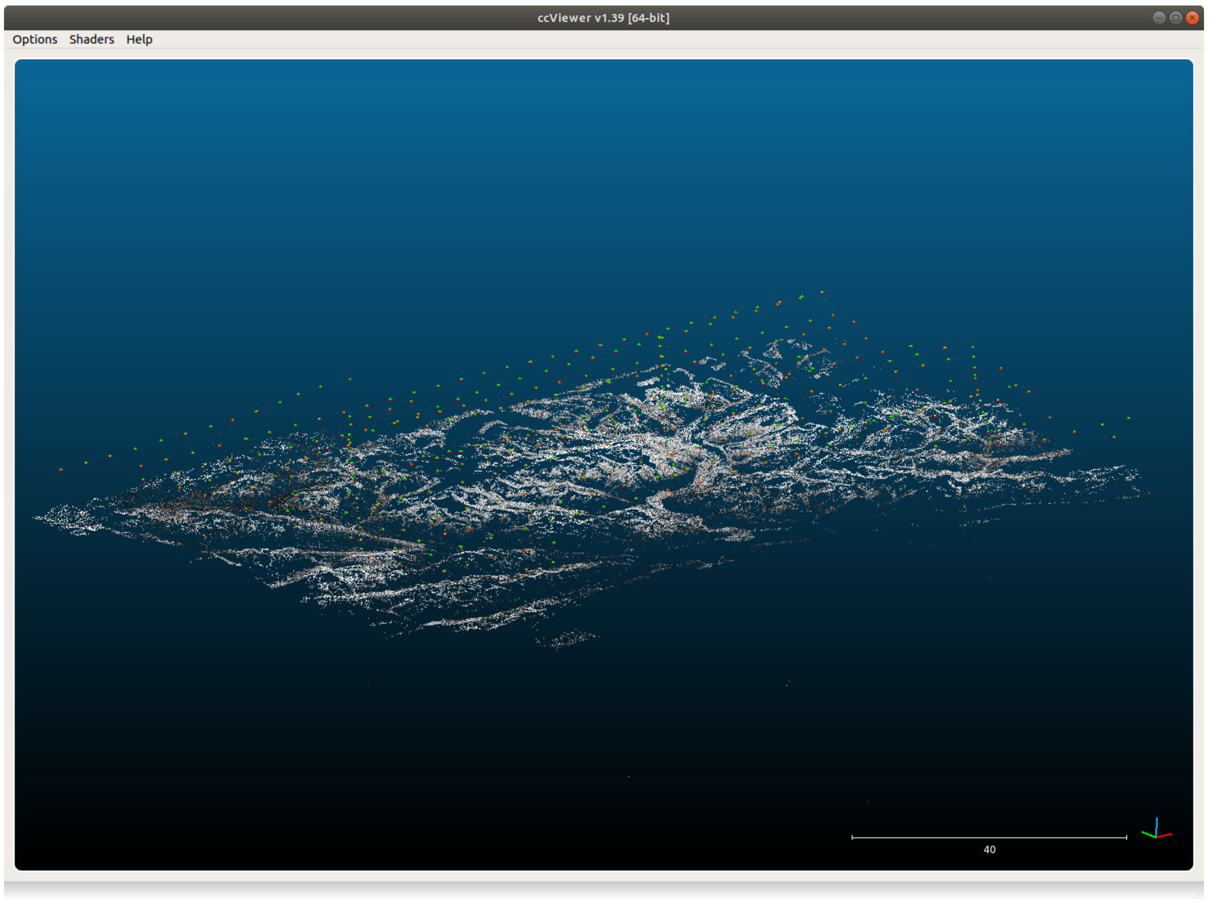
Finally, in areas with lots of water or terrain that has changed substantially (e.g., glaciers), *spymicmac.register.register_relative()* can use both exclusion and inclusion masks to avoid searching for matches on unstable terrain. Just like with the footprints, these files should be any data format that can be read by geopandas.read_file.

The **exclusion** mask should be polygons of glaciers, landslides, or other terrain that should be *excluded* from the search. Areas covered by this mask will be excluded from the search.

The **inclusion** mask should be polygons of land - any terrain that should be *included* in the search. Areas that are not covered by this mask will be excluded from the search.

relative to absolute transformation

The first step in *spymicmac.register.register_relative()* to use *spymicmac.orientation.transform_centers()* to transform between the relative and absolute spaces, using the centroids of the footprint polygons and the camera positions estimated by `mm3d Tapas`.



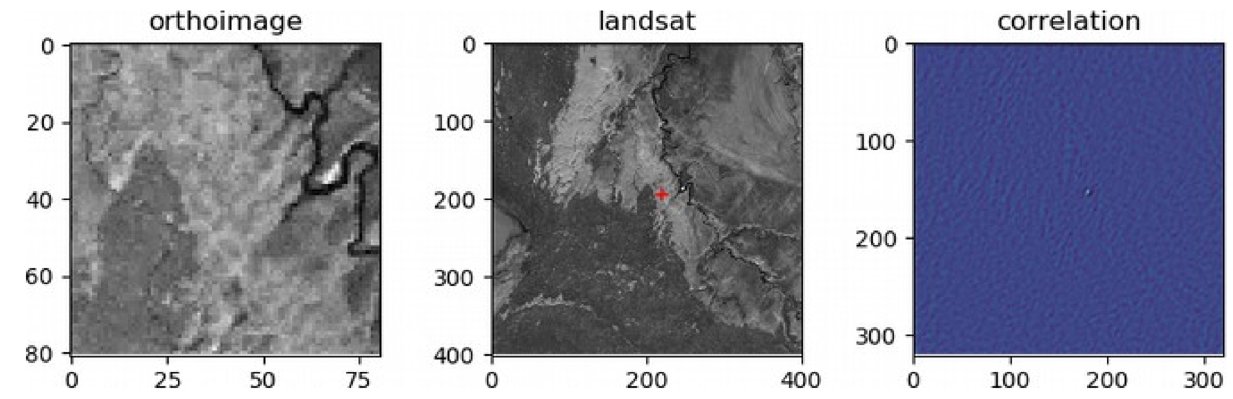
Because the footprints are most likely approximate, especially for historic datasets, this step uses RANSAC with a fairly large residual threshold. The goal is to create a rough transformation of the relative orthophoto that can be used for the gridded template matching step.

gridded template matching

Once the relative orthophoto has been roughly transformed to absolute space, *`spymicmac.register.register_relative()`* find matches between the orthophoto and the reference image using *`spymicmac.matching.find_grid_matches()`*. The size of each search window is set by `dstwin`, and the templates (of size 121x121 pixels) are taken from a grid with spacing determined by the `density` parameter.

Each template and search image are first run through *`spymicmac.image.highpass_filter()`*, to help minimize radiometric differences between the two images (and maximizing the high-frequency variation). After that, the template and search image are passed to OpenCV matchTemplate, and the best match is found using normalized cross-correlation.

The correlation value of each potential match is then compared to the standard deviation of all of the correlation values from the search image. This value (`z_corr`) is then used to filter out poor matches later on, as higher quality matches are more likely to represent larger departures from the background correlation value:



iterative outlier removal

After the potential matches are found, a number of filtering steps are used to refine the results. First, any matches where the DEM does not have a value are removed. Then, an affine transformation between the relative orthoimage and reference orthoimage locations is estimated using RANSAC, to help remove obvious blunders.

Next, mm3d GCPBascule is called, which transforms the camera locations to the absolute space. The residuals for each GCP are then calculated, and outliers more than 2 normalized median absolute deviations (NMAD) from the median residual value are discarded, and `mm3d GCPBascule` is called again.

This is followed by a call to mm3d Campari using *`spymicmac.micmac.campari()`*, and again residuals more than 2 NMAD from the median residual value are discarded.

After this, this process (`mm3d GCPBascule` -> `mm3d Campari` -> outlier removal) is run up to 4 more times, until there are no further outliers found.

final result

Once the outliers have been removed, the final GCP locations are stored in a number of files:

- auto_gcps/AutoGCPs.xml
- auto_gcps/AutoGCPs.txt
- auto_gcps/AutoGCPs.shp (+ other files)
- AutoMeasures.xml – the GCP locations in each of the individual images

If there are still problematic GCPs, you can manually delete them from `AutoMeasures.xml` and re-run `mm3d GCPBascule` and `mm3d Campari`.

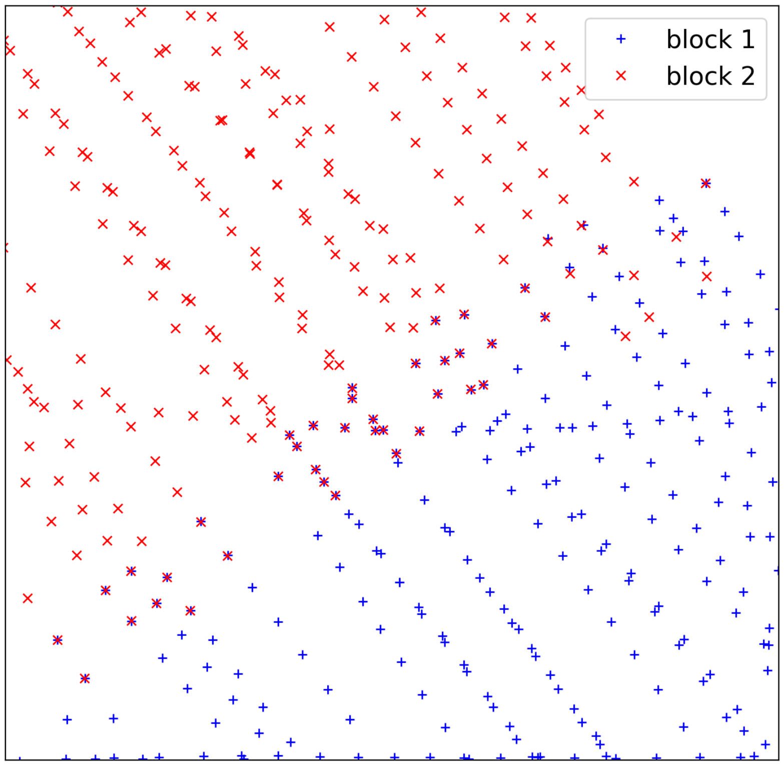
The next step will be to run mm3d Malt using the `Ori-TerrainFirstPass` directory, to produce the absolute orthophoto and DEM.

3.3.7 combining image blocks

Note: For "small" (<500) numbers of images, you can usually run all of the processing steps using all of the images. With more and more images, however, it can be advantageous to divide the images into sub-blocks.

The *finding the relative orientation* step should be run with all of the images, in order to get the best possible relative orientation, especially when working with areas with poor tie point coverage (for example, large glaciers or ice fields).

If you divide the study area into smaller sub-blocks, you should run the previous steps (*computing the relative orthophoto*, *automatically finding control points*) on each sub-block individually. Unfortunately, this can often lead to issues in the areas where the blocks overlap, as seen below.

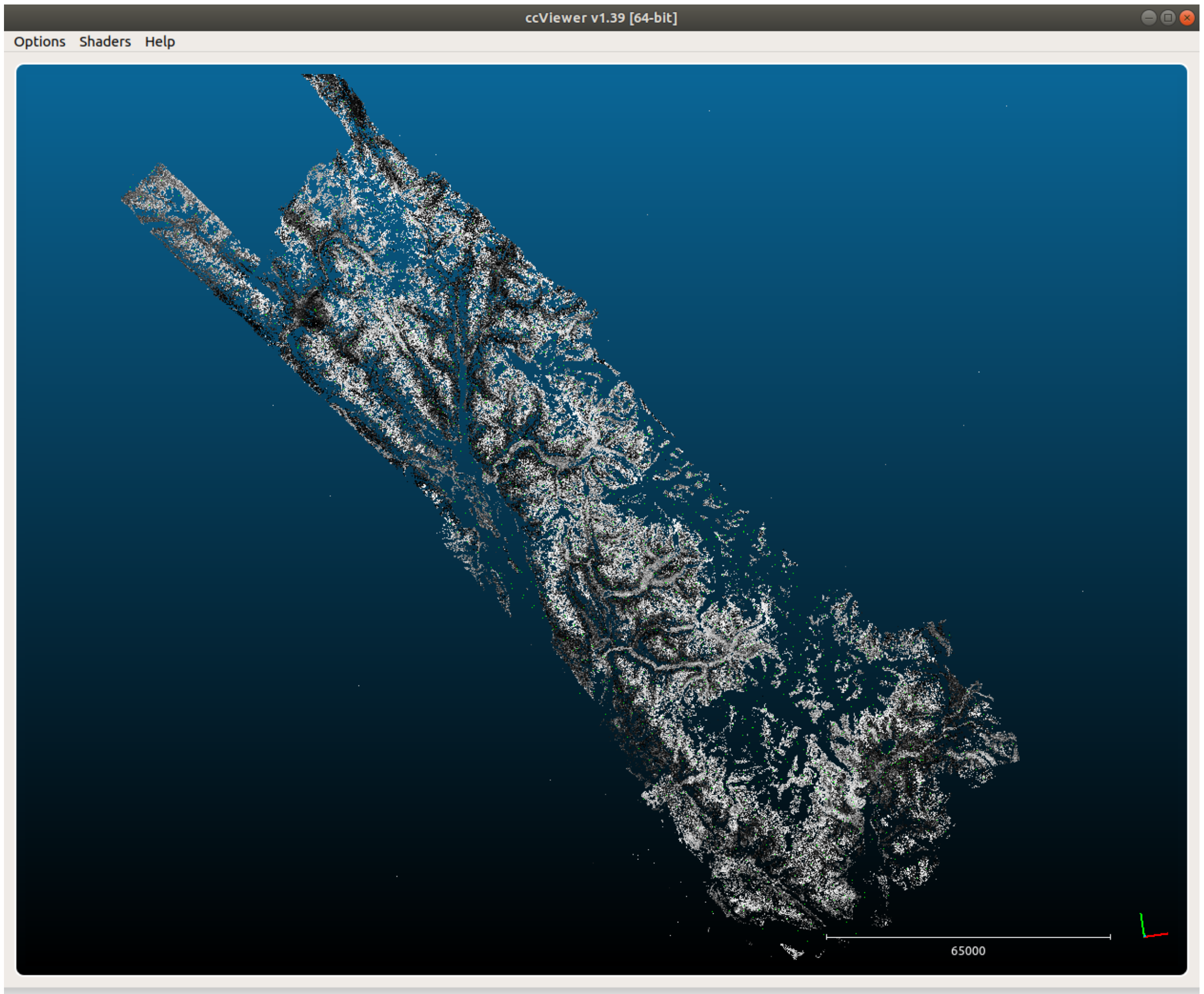


The image below shows camera centers for two different sub-blocks - note the different calculated camera positions in the overlapping area - when the absolute DEMs and Orthophotos are created using `Malt`, this difference will make combining the DEMs and Orthophotos more difficult.

To prevent this issue, *`spymicmac.orientation.combine_block_measures()`* will combine the individual GCP and Measures files for each block, so that `Campari` can be used to create a single absolute orientation for all of the images. You can also run *`spymicmac.micmac.iterate_campari()`* to iteratively run `Campari`, removing any outlier GCPs to further refine/correct the orientation.

Or, you can use *`spymicmac.orientation.block_orientation()`* or the corresponding command-line tool *block_orientation* to combine the steps of both.

The screenshot below shows the point cloud file for a set of 1455 historic images acquired over the Juneau Icefield.



The images were divided into 4 overlapping blocks of roughly equal size for the *computing the relative orthophoto* and *automatically finding control points* steps, before being re-combined into a single orientation used to extract DEMs and Orthophotos. This has resulted in relatively smooth overlaps between the different DEM blocks, as illustrated in the image below:

3.3.8 computing the absolute orthophoto and DEM

Once you have successfully run *spymicmac.register.register_relative()*, you will have an orientation folder, `Ori-TerrainFirstPass`[1]. This is the folder to pass to `Malt`:

```
mm3d Malt Ortho "OIS.*tif" TerrainFirstPass DirMEC=MEC-Malt NbVI=2 MasqImGlob=filtre.tif␣
↪ZoomF=1 DefCor=0 CostTrans=1 EZA=1
```

Just like with the *computing the relative orthophoto* step, once this command finishes, you will have two new directories: `MEC-Malt` and `Ortho-MEC-Malt`. The DEM and associated correlation masks are found in `MEC-Malt`, while the orthophotos are found in `Ortho-MEC-Malt`.

Depending on the scale of your images and how many you are using, you will probably need to mosaic the tiles of the DEM, and possibly the correlation masks[2]:

```
cd MEC-Malt
mosaic_micmac_tiles.py -filename Z_Num9_DeZoom1_STD-Malt
mosaic_micmac_tiles.py -filename Correl_STD-MALT_Num_8
```

Once the tiles have been mosaicked, you can also run **post_process_micmac.sh** to apply the AutoMask and add a CRS to the DEM.

creating the orthomosaic using Tawny

Just like with the *computing the relative orthophoto* step, the orthoimages are not mosaicked - they are just the individual images orthorectified using the extracted DEM - you'll need to run `Tawny` again to mosaic the images:

```
mm3d Tawny Ortho-MEC-Malt Out=Orthophotomosaic.tif RadiomEgal=0
```

Again, we are using `RadiomEgal=0` to use the images as-is, rather than attempting to balance the radiometry (as this can lead to undesirable results). Finally, you might need to re-combine the image tiles, depending on how large they are:

```
cd Ortho-MEC-Malt
mosaic_micmac_tiles.py -filename Orthophotomosaic
```

At this point, you should have a finished DEM and orthomosaic. You may want to check the accuracy of your DEM by co-registering it to a DEM of known quality. You may also wish to remove residual doming effects using `mm3d PostProc Banana`.

You can also run *post_process_micmac* to apply the AutoMask to the DEM and georeference the correlation mask:

```
cd MEC-Malt
post_process_micmac.sh -z "8 +north" -n Block1
```

[1] If you are running *spymicmac.register.register_relative()* with `block` set to a value (e.g., `-b 1`), this will be `Ori-TerrainFirstPass_block1`. You should also change `DirMEC` to a different name, (e.g., `MEC-Malt_block1`), otherwise it will be overwritten with each new block that you run.

[2] Note that in some cases with very large image blocks, the final steps will actually be `Z_Num10_DeZoom1_STD-MALT` and `Correl_STD-MALT_Num_9` - it's probably a good idea to check this!

CHAPTER

FOUR

SPYMICMAC API

spymicmac modules and shell scripts

4.1 python modules

4.1.1 spymicmac.data

spymicmac.data is a collection of tools for handling external datasets

`spymicmac.data.`**`download_arcticdem_mosaic`**(*imlist=None*, *footprints=None*, *imgsource='DECLASSII'*, *globstr='OIS*.tif'*, *res='2m'*, *write_urls=False*)

Download the ArcticDEM v3.0 Mosaic tiles that intersect image footprints. Downloads .tar.gz files to arctic_dem, extracts each DEM, and creates ArcticDEM.vrt in the current directory.

Parameters

- **`imlist`** (*`list`*) – a list of image filenames. If None, uses globstr to search for images in the current directory.
- **`footprints`** (*`GeoDataFrame`*) – a GeoDataFrame of image footprints. If None, uses spymicmac.usgs.get_usgs_footprints to download footprints based on imlist.
- **`imgsource`** (*`str`*) – the EE Dataset name for the images (default: DECLASSII)
- **`globstr`** (*`str`*) – the search string to use to find images in the current directory.
- **`res`** (*`str`*) – the DEM resolution to download. Options are 2m, 10m, or 32m (default: 2m)
- **`write_urls`** (*`bool`*) – write a text file with the urls for each tile, for downloading using curl, wget, etc., instead of via python (default: False)

`spymicmac.data.`**`download_cop30_vrt`**(*imlist=None*, *footprints=None*, *imgsource='DECLASSII'*, *globstr='OIS*.tif'*)

Create a VRT using Copernicus 30m DSM tiles that intersect image footprints. Creates Copernicus_DSM.vrt using files downloaded to cop30_dem/ within the current directory.

Parameters

- **`imlist`** (*`list`*) – a list of image filenames. If None, uses globstr to search for images in the current directory.
- **`footprints`** (*`GeoDataFrame`*) – a GeoDataFrame of image footprints. If None, uses spymicmac.usgs.get_usgs_footprints to download footprints based on imlist.
- **`imgsource`** (*`str`*) – the EE Dataset name for the images (default: DECLASSII)

- **globstr** (*str*) – the search string to use to find images in the current directory.

spymicmac.data.**get_usgs_footprints**(*imlist*, *dataset='DECLASSII'*)

Search for a list of images on USGS Earth Explorer. Note that the image names should be the USGS entity ID (e.g., AR5840034159994 rather than AR5840034159994.tif).

Parameters

- **imlist** (*list*) – a list of image names.
- **dataset** (*str*) – the USGS dataset name to search (default: DECLASSII).

Returns

footprints (*GeoDataFrame*) – a GeoDataFrame of image footprints.

spymicmac.data.**landsat_to_gdf**(*results*)

Convert USGS Landsat search results to a GeoDataFrame

Parameters

results (*list*) – a list of results (i.e., the 'data' value returned by api.scene_metadata)

Returns

meta_gdf (*geopandas.GeoDataFrame*) - a GeoDataFrame of search results

spymicmac.data.**to_wgs84_ellipsoid**(*fn_dem*)

Convert a DEM to WGS84 ellipsoid heights, using the EGM08 Geoid.

Parameters

fn_dem (*str*) – the filename of the DEM to convert.

4.1.2 spymicmac.ee_tools

Note: Before using spymicmac.ee_tools, you will need to run ee.Authenticate(), in order to authorize the access needed by EarthEngine. To do this from the python terminal, run the following:

```
import ee
ee.Authenticate()
```

This will open a browser window and prompt you to select your Google account for authentication. It will generate an authorization code which you should then copy and paste into the prompt in the python terminal.

After you have successfully authenticated, you should be able to use spymicmac.ee_tools normally (note that you will need to explicitly import `ee_tools` - it is not loaded by `spymicmac.__init__.py`, in order to prevent authorization errors):

```
from spymicmac import ee_tools
```

You won't need to do this every time you use *spymicmac.ee_tools()*, but you may need to re-authorize from time to time.

spymicmac.ee_tools is a collection of tools for getting GEE data for processing KH-9 Hexagon imagery.

spymicmac.ee_tools.**get_alos_dem**(*filt_geom=None*)

Return the ALOS DEM, clipped to an optional geometry.

Parameters

filt_geom – A geometry object to clip images to

Returns

spymicmac.ee_tools.**get_landsat_collections**(*filt_geom=None*, *sensors='all'*, *tier='all'*)

Return a collection of raw Landsat images filtered by an (optional) geometry, given a list of sensor names and tier levels. Default behavior is to select all sensors and all tiers.

Valid sensor names: LC08, LE07, LT05, LT04, LM05, LM04, LM03, LM02, LM01

Valid tier levels: T1, T2. Check Landsat docs for details.

Parameters

- **filt_geom** – A geometry object to filter images by. See EE docs for more details.
- **sensors** – Landsat sensor(s) to select images from.
- **tier** – Processing tier(s) to select.

Returns

spymicmac.ee_tools.**get_srtm**(*ver='usgs'*, *filt_geom=None*)

Return the SRTM DEM, clipped to an optional geometry.

Parameters

- **ver** – SRTM version to select, one of 'usgs', 'cgiar'
- **filt_geom** – A geometry object to clip images to

Returns

4.1.3 spymicmac.image

spymicmac.image is a collection of tools for working with images.

spymicmac.image.**balance_image**(*img*)

Apply contrast-limited adaptive histogram equalization (CLAHE) on an image, then apply a de-noising filter.

Parameters
img (*array-like*) – the image to balance.

Returns
img_filt (*array-like*) – the balanced, filtered image.

spymicmac.image.**contrast_enhance**(*fn_img*, *mask_value=None*, *qmin=0.02*, *qmax=0.98*, *gamma=1.25*, *disksize=3*, *imgmin=0*)

Enhance image contrast in a three-step process. First, the image is processed with a median filter to reduce noise. Next, a linear contrast stretch is applied, and finally, a gamma adjustment is applied.

Parameters

- **fn_img** (*str*) – the image filename.
- **mask_value** (*int | float*) – a mask value to use when filtering the image.
- **qmin** (*float*) – the minimum quantile to use for the linear contrast stretch (default: 0.02)
- **qmax** (*float*) – the maximum quantile to use for the linear contrast stretch (default: 0.98)
- **gamma** (*float*) – the value to use for the gamma adjustment
- **disksize** (*int*) – the filter disk size (input to skimage.morphology.disk; default: 3)
- **imgmin** (*int | float*) – the minimum value in the output image (default: 0)

Returns
enhanced (*array-like*) – the contrast-enhanced image.

spymicmac.image.**get_parts_list**(*im_pattern*)

Find all of the parts of a scanned image that match a given filename pattern

Parameters
im_pattern (*str*) – the image pattern to match

Returns
parts_list (*list*) – a list of all parts of the image that match the pattern.

spymicmac.image.**get_rough_frame**(*img*)

Find the rough location of an image frame/border.

Parameters
img (*array-like*) – the image to find a border for

Returns
xmin, **xmax**, **ymin**, **ymax** (*float*) – the left, right, top, and bottom indices for the rough border.

spymicmac.image.**highpass_filter**(*img*)

Subtract a low-pass from an image, to return a highpass filter.

Parameters
img (*array-like*) – the image to filter.

Returns
highpass (*array-like*) – the highpass-filtered image

spymicmac.image.**join_halves**(*left*, *right*, *overlap*, *block_size=None*, *blend=True*, *trim=None*)

Join two halves of a scanned image together.

Parameters

- **left** (*array-like*) – the left half of the image
- **right** (*array-like*) – the right half of the image
- **overlap** (*int*) – the amount of overlap, in pixels, between the two halves.
- **block_size** (*int*) – the number of rows each sub-block should cover. Defaults to overlap.
- **blend** (*bool*) – apply a linear blend between the two scanned halves (default: True).
- **trim** (*int*) – the amount to trim the right side of the image by. (default: None).

spymicmac.image.**join_hexagon**(*im_pattern*, *overlap=2000*, *block_size=None*, *blend=True*, *is_reversed=False*)

Join multiple parts of a scanned image.

Parameters

- **im_pattern** (*str*) – the base name of the image to use (e.g., DZB1216-500280L002001).
- **overlap** (*int*) – the overlap, in pixels, between the image parts.
- **block_size** (*int*) – the number of rows each sub-block should cover. Defaults to overlap.
- **blend** (*bool*) – apply a linear blend between the two scanned halves (default: True).
- **is_reversed** (*bool*) – parts are in reversed order (i.e., part b is the left part, part a is the right part)

spymicmac.image.**make_binary_mask**(*img*, *mult_value=255*, *erode=0*, *mask_value=0*)

Create a binary mask for an image based on a given mask value. Values equal to mask_value will be given a value of 0 in the mask, while all other values will be set equal to mult_value.

Parameters

- **img** (*array-like*) – the image to create a mask for.
- **mult_value** (*int | float*) – the value indicating a non-masked value (default: 255).
- **erode** (*int*) – the size of the erosion operation to apply (default: 0).
- **mask_value** (*int*) – the value to mask in the image (default: 0).

Returns

mask (*array-like*) the binary mask.

spymicmac.image.**remove_scanner_stripes**(*img*, *dtype=<class 'numpy.uint8'>*, *scan_axis=1*)

Remove horizontal (or vertical) stripes from an image.

Parameters

- **img** (*array-like*) – the image to remove stripes from.
- **dtype** (*numpy.dtype*) – the original datatype of the image.
- **scan_axis** (*int*) – the axis corresponding to the direction of the stripes. A scan_axis of 1 corresponds to horizontal stripes (the default), while a scan_axis of 0 corresponds to vertical stripes.

Returns

destriped: the original image with the stripes (mostly) removed.

spymicmac.image.**splitter**(*img*, *nblocks*, *overlap=0*)

Split an image into (m, n) blocks with a given overlap.

Parameters

- **img** (*array-like*) – the image to split
- **nblocks** (*tuple*) – the number of blocks to create along each axis (m, n)
- **overlap** (*int*) – the number of pixels to overlap each block. (default: 0)

Returns

- **blocks** (*list*) – a list of the image blocks created
- **top_inds** (*list*) – a list of the original row index of the top edge of each block
- **left_inds** (*list*) – a list of the original column index of the left edge of each block

spymicmac.image.**stretch_image**(*img*, *scale=(0.0, 1.0)*, *mult=255*, *imgmin=0*, *outtype=<class 'numpy.uint8'>*, *mask=None*)

Apply a linear stretch to an image by clipping and stretching to quantiles.

Parameters

- **img** (*array-like*) – the image to stretch.
- **scale** (*tuple*) – the minimum and maximum quantile to stretch to. (default: (0, 1) - the minimum/maximum values of the image)
- **mult** (*int | float*) – a multiplier to scale the result to. (default: 255)
- **imgmin** (*int | float*) – the minimum value in the output image (default: 0)

- **outtype** (*numpy.dtype*) – the numpy datatype to return the stretched image as. (default: np.uint8)
- **mask** (*array-like*) – a mask of pixels to ignore when calculating quantiles.

Returns
stretched (*array-like*) – the stretched image.

4.1.4 spymicmac.matching

spymicmac.matching is a collection of tools for matching templates in images

`spymicmac.matching.`**`cross_template`**(*shape*, *width=3*)

Create a cross-shaped template for matching reseau or fiducial marks.

Parameters

- **shape** (*int*) – the output shape of the template
- **width** (*int*) – the width of the cross at the center of the template (default: 3 pixels).

Returns
cross (*array-like*) – the cross template

`spymicmac.matching.`**`do_match`**(*dest_img*, *ref_img*, *mask*, *pt*, *srcwin*, *dstwin*)

Find a match between two images using normalized cross-correlation template matching.

Parameters

- **dest_img** (*array-like*) – the image to search for the matching point in.
- **ref_img** (*array-like*) – the reference image to use for matching.
- **mask** (*array-like*) – a mask indicating areas that should be used for matching.
- **pt** (*array-like*) – the index (i, j) to search for a match for.
- **srcwin** (*int*) – the half-size of the template window.
- **dstwin** (*int*) – the half-size of the search window.

Returns

- **match_pt** (*tuple*) – the matching point (j, i) found in dest_img
- **z_corr** (*float*) – number of standard deviations (z-score) above other potential matches
- **peak_corr** (*float*) – the correlation value of the matched point

`spymicmac.matching.`**`find_crosses`**(*img*, *cross*)

Find all cross markers in an image.

Parameters

- **img** (*array-like*) – the image
- **cross** (*array-like*) – the cross template to use

Returns
grid_df (*pandas.DataFrame*) – a dataframe of marker locations and offsets

`spymicmac.matching.`**`find_fiducials`**(*fn_img*, *templates*, *fn_cam=None*)

Match the location of fiducial markers for a scanned aerial photo.

Parameters

- **fn_img** (*str*) – the filename of the image to find fiducial markers in.
- **templates** (*dict*) – a dict of (name, template) pairs corresponding to each fiducial marker.
- **fn_cam** (*str*) – the filename of the MeasuresCamera.xml file for the image. defaults to Ori-InterneScan/MeasuresCamera.xml

`spymicmac.matching.`**`find_grid_matches`**(*tfm_img*, *refgeo*, *mask*, *initM=None*, *spacing=200*, *srcwin=60*, *dstwin=600*)

Find matches between two images on a grid using normalized cross-correlation template matching.

Parameters

- **tfm_img** (*array-like*) – the image to use for matching.
- **refgeo** (*GeoImg*) – the reference image to use for matching.
- **mask** (*array-like*) – a mask indicating areas that should be used for matching.
- **initM** – the model used for transforming the initial, non-georeferenced image.
- **spacing** (*int*) – the grid spacing, in pixels (default: 200 pixels)
- **srcwin** (*int*) – the half-size of the template window.
- **dstwin** (*int*) – the half-size of the search window.

Returns

gcps (*pandas.DataFrame*) – a DataFrame with GCP locations, match strength, and other information.

`spymicmac.matching.`**`find_kh4_notches`**(*img*, *size=101*)

Find all 4 notches along the top of a KH-4 style image.

Parameters

- **img** (*array-like*) – the image.
- **size** (*int*) – the size of the notch template to use.

Returns

coords (*array-like*) – a 4x2 array of notch locations

`spymicmac.matching.`**`find_match`**(*img*, *template*, *how='min'*, *eq=True*)

Given an image and a template, find a match using openCV's normed cross-correlation.

Parameters

- **img** (*array-like*) – the image to find a match in
- **template** (*array-like*) – the template to use for matching
- **how** (*str*) – determines whether the match is the minimum or maximum correlation (default: min)
- **eq** (*bool*) – use a rank equalization filter before matching the templates (default: True)

Returns

- **res** (*array-like*) – the correlation image
- **match_i** (*float*) – the row location of the match

- **match_j** (*float*) – the column location of the match

`spymicmac.matching.`**`find_rail_marks`**(*img*)

Find all rail marks along the bottom edge of a KH-4 style image.

Parameters
`img` (`array-like`) – the image to find the rail marks in.

Returns
coords (*array-like*) – an Nx2 array of the location of the detected markers.

`spymicmac.matching.`**`find_reseau_grid`**(*fn_img*, *csize=361*, *return_val=False*)

Find the locations of the Reseau marks in a scanned KH-9 image. Locations are saved to Ori-InterneScan/MeasuresIm-:fn_img:.xml.

Parameters

- `fn_img` (`str`) – the image filename.
- `csize` (`int`) – the size of the cross template (default: 361 -> 361x361)
- `return_val` (`bool`) – return a pandas DataFrame of the Reseau mark locations (default: False).

Returns
gcps_df (*pandas.DataFrame*) – a DataFrame of the Reseau mark locations (if return_val=True).

`spymicmac.matching.`**`get_dense_keypoints`**(*img*, *mask*, *npix=100*, *nblocks=None*, *return_des=False*)

Find ORB keypoints by dividing an image into smaller parts.

Parameters

- `img` (`array-like`) – the image to use.
- `mask` (`array-like`) – a mask to use for keypoint generation.
- `npix` (`int`) – the block size (in pixels) to divide the image into.
- `nblocks` (`int`) – the number of blocks to divide the image into. If set, overrides value given by npix.
- `return_des` (`bool`) – return the keypoint descriptors, as well

Returns

- **keypoints** (*list*) – a list of keypoint locations
- **descriptors** (*list*) – if requested, a list of keypoint descriptors.

`spymicmac.matching.`**`get_matches`**(*img1*, *img2*, *mask1=None*, *mask2=None*, *dense=False*, *npix=100*, *nblocks=None*)

Return keypoint matches found using openCV's ORB implementation.

Parameters

- `img1` (`array-like`) – the first image to match
- `img2` (`array-like`) – the second image to match
- `mask1` (`array-like`) – a mask to use for the first image. (default: no mask)
- `mask2` (`array-like`) – a mask to use for the second image. (default: no mask)
- `dense` (`bool`) – compute matches over sub-blocks (True) or the entire image (False). (default: False)

Returns

- **keypoints** (*tuple*) – the keypoint locations for the first and second image.
- **descriptors** (*tuple*) – the descriptors for the first and second image.
- **matches** (*list*) – a list of matching keypoints between the first and second image

`spymicmac.matching.`**`make_template`**`(`*img*, *pt*, *half_size*`)`

Return a sub-section of an image to use for matching.

Parameters

- **`img`** (*`array-like`*) – the image from which to create the template
- **`pt`** (*`tuple`*) – the (row, column) center of the template
- **`half_size`** (*`int`*) – the half-size of the template; template size will be 2 * half_size + 1

Returns

- **template** (*array-like*) – the template
- **row_inds** (*list*) – the number of rows above/below the center of the template
- **col_inds** (*list*) – the number of columns left/right of the center of the template

`spymicmac.matching.`**`match_fairchild_k17`**`(`*fn_img*, *size=101*, *fn_cam=None*`)`

Match the "fiducial" locations for a Fairchild K17B-style camera (4 "wing" style fiducial markers in the middle of each side of the image).

Parameters

- **`fn_img`** (*`str`*) – the filename of the image to find fiducial markers in.
- **`size`** (*`int`*) – the size of the template to use (default: 101 pixels)
- **`fn_cam`** (*`str`*) – the filename of the MeasuresCamera.xml file for the image. defaults to Ori-InterneScan/MeasuresCamera.xml

`spymicmac.matching.`**`match_halves`**`(`*left*, *right*, *overlap*, *block_size=None*`)`

Find a transformation to join the left and right halves of an image scan.

Parameters

- **`left`** (*`array-like`*) – the left-hand image scan.
- **`right`** (*`array-like`*) – the right-hand image scan.
- **`overlap`** (*`int`*) – the estimated overlap between the two images, in pixels.
- **`block_size`** (*`int`*) – the number of rows each sub-block should cover. Defaults to overlap.

Returns

model (*EuclideanTransform*) – the estimated Euclidean transformation between the two image halves.

`spymicmac.matching.`**`match_reseau_grid`**`(`*img*, *coords*, *cross*`)`

Find the best match for each KH-9 mapping camera reseau grid point, given a list of potential matches.

Parameters

- **`img`** (*`array-like`*) – the image to use
- **`coords`** (*`array-like`*) – the coordinates of the potential matches
- **`cross`** (*`array-like`*) – the cross template to use.

Returns
grid_df (*pandas.DataFrame*) – a DataFrame of grid locations and match points

`spymicmac.matching.`**`notch_template`**`(`*size*`)`

Create a notch-shaped ("^") template.

Parameters
`size` (*`int`*) – the size of the template, in pixels

Returns
template (*array-like*) – the notch template

`spymicmac.matching.`**`ocm_show_wagon_wheels`**`(`*img*, *size*, *width=3*, *img_border=None*`)`

Find all "wagon wheel" markers in an image.

Parameters

- **`img`** (*`array-like`*) – the image
- **`size`** (*`int`*) – the size of the marker (in pixels)
- **`width`** (*`int`*) – the width/thickness of the cross, in pixels (default: 3)
- **`img_border`** – the approximate top and bottom rows of the image frame. If not set, calls get_rough_frame() on the image.

Returns
coords an Nx2 array of the location of the detected markers.

`spymicmac.matching.`**`remove_crosses`**`(`*fn_img*, *nproc=1*`)`

Remove the Reseau marks from a KH-9 image before re-sampling.

Parameters

- **`fn_img`** (*`str`*) – the image filename.
- **`nproc`** (*`int`*) – the number of subprocesses to use (default: 1).

`spymicmac.matching.`**`templates_from_meas`**`(`*fn_img*, *half_size=100*`)`

Create fiducial templates from points in a MeasuresIm file.

Parameters

- **`fn_img`** (*`str`*) – the filename of the image to use. Points for templates will be taken from Ori-InterneScan-Measuresim{fn-img}.xml.
- **`half_size`** (*`int`*) – the half-size of the template to create, in pixels (default: 100)

Returns
templates (*dict*) – a *dict* of (name, template) pairs for each fiducial marker.

`spymicmac.matching.`**`wagon_wheel`**`(`*size*, *width=3*, *mult=255*`)`

Creates a template in the shape of a "wagon wheel" (a cross inscribed in a ring).

Parameters

- **`size`** (*`int`*) – the width (and height) of the template, in pixels
- **`width`** (*`int`*) – the width/thickness of the cross, in pixels
- **`mult`** – a multiplier to use for the template [default: 255]

Returns
template (*array-like*) the wagon wheel template

4.1.5 spymicmac.micmac

spymicmac.micmac is a collection of tools for interfacing with MicMac

`spymicmac.micmac.`**`apericloud`**(*ori*, *img_pattern='OIS.*tif'*)

Run mm3d AperiCloud to create a point cloud layer

Parameters

- **ori** (*str*) – the input orientation to use
- **img_pattern** (*str*) – the image pattern to pass to AperiCloud (default: OIS.*tif)

`spymicmac.micmac.`**`bascule`**(*in_gcps*, *outdir*, *img_pattern*, *sub*, *ori*, *outori='TerrainRelAuto'*, *fn_gcp='AutoGCPs'*, *fn_meas='AutoMeasures'*)

Interface for running mm3d GCPBascule and reading the residuals from the resulting xml file.

Parameters

- **in_gcps** (*pandas.DataFrame*) – a DataFrame with the GCPs that are being input to Campari.
- **outdir** (*str*) – the output directory where the AutoGCPs.xml file is saved.
- **img_pattern** (*str*) – the match pattern for the images being input to Campari (e.g., “OIS.*tif”)
- **sub** (*str*) – the name of the block, if multiple blocks are being used (e.g., ‘_block1’). If not, use ‘’.
- **ori** (*str*) – the name of the orientation directory (e.g., Ori-Relative).
- **outori** (*str*) – the name of the output orientation directory (default: TerrainRelAuto).
- **fn_gcp** (*str*) – the filename pattern for the GCP file. The file that will be loaded will be fn_gcp + sub + ‘.xml’ (e.g., default: AutoGCPs -> AutoGCPs_block0.xml)
- **fn_meas** (*str*) – the filename pattern for the measures file. The file that will be loaded will be fn_meas + sub + ‘-S2D.xml’ (e.g., default: AutoMeasures -> AutoMeasures_block0-S2D.xml)

Returns

out_gcps (*pandas.DataFrame*) – the input gcps with the updated Bascule residuals.

`spymicmac.micmac.`**`block_malt`**(*imlist*, *nimg=3*, *ori='Relative'*, *zoomf=8*)

Run mm3d Malt Ortho and mm3d Tawny on successive blocks of images.

Parameters

- **imlist** (*iterable*) – an iterable object of image filenames, or an iterable object of lists of image filenames
- **nimg** (*int*) – the number of images to use in a block (default: 3)
- **ori** (*str*) – the name of the orientation directory (e.g., Ori-Relative). (default: Relative)
- **zoomf** (*int*) – the final Zoom level to use (default: 8)

`spymicmac.micmac.`**`campari`**(*in_gcps*, *outdir*, *img_pattern*, *sub*, *dx*, *ortho_res*, *allfree=True*, *fn_gcp='AutoGCPs'*, *fn_meas='AutoMeasures'*, *inori='TerrainRelAuto'*, *outori='TerrainFinal'*, *homol='Homol'*)

Interface for running mm3d Campari and reading the residuals from the residual xml file.

Parameters

- **in_gcps** (*pandas.DataFrame*) – a DataFrame with the GCPs that are being input to Campari.
- **outdir** (*str*) – the output directory where the AutoGCPs.xml file is saved.
- **img_pattern** (*str*) – the match pattern for the images being input to Campari (e.g., "OIS.*tif")
- **sub** (*str*) – the name of the block, if multiple blocks are being used (e.g., '_block1'). If not, use ''.
- **dx** (*int | float*) – the pixel resolution of the reference image.
- **ortho_res** (*int | float*) – the pixel resolution of the orthoimage being used.
- **allfree** (*bool*) – run Campari with AllFree=1 (True), or AllFree=0 (False). (default: True)
- **fn_gcp** (*str*) – the filename pattern for the GCP file. The file that will be loaded will be fn_gcp + sub + '.xml' (e.g., default: AutoGCPs -> AutoGCPs_block0.xml)
- **fn_meas** (*str*) – the filename pattern for the measures file. The file that will be loaded will be fn_meas + sub + '-S2D.xml' (e.g., default: AutoMeasures -> AutoMeasures_block0-S2D.xml)
- **inori** (*str*) – the input orientation to Campari (default: Ori-TerrainRelAuto -> TerrainRelAuto)
- **outori** (*str*) – the output orientation from Campari (default: Ori-TerrainFinal -> TerrainFinal)
- **homol** (*str*) – the Homologue directory to use (default: Homol)

Returns

out_gcps (*pandas.DataFrame*) – the input gcps with the updated Campari residuals.

spymicmac.micmac.**create_localchantier_xml**(*name='KH9MC'*, *short_name='KH-9 Hexagon Mapping Camera'*, *film_size=(460, 220)*, *pattern='.*'*, *focal=304.8*, *add_sfs=False*)

Create a MicMac-LocalChantierDescripteur.xml file for a given camera. Default is the KH-9 Hexagon Mapping Camera.

Parameters

- **name** (*str*) – The name to use for the camera [KH9MC]
- **short_name** (*str*) – A short description of the camera [KH-9 Hexagon Mapping Camera]
- **film_size** (*array-like*) – the film size (width, height) in mm [460, 220]
- **pattern** (*str*) – the matching pattern to use for the images [.*]
- **focal** (*float*) – the nominal focal length, in mm [304.8]
- **add_sfs** (*bool*) – use SFS to help find tie points in low-contrast images [False]

spymicmac.micmac.**create_measurescamera_xml**(*fn_csv*, *ori='InterneScan'*, *translate=False*, *name='gcp'*, *x='im_col'*, *y='im_row'*)

Create a MeasuresCamera.xml file from a csv of fiducial marker locations.

Parameters

- **fn_csv** (*str*) – the filename of the CSV file.
- **ori** (*str*) – the Ori directory to write the MeasuresCamera.xml file to. Defaults to (Ori-)InterneScan.

- **translate** (*bool*) – translate coordinates so that the origin is the upper left corner, rather than the principal point
- **name** (*str*) – the column name in the csv file corresponding to the point name [gcp]
- **x** (*str*) – the column name in the csv file corresponding to the image x location [im_col]
- **y** (*str*) – the column name in the csv file corresponding to the image y location [im_row]

spymicmac.micmac.**dem_to_text**(*fn_dem*, *fn_out='dem_pts.txt'*, *spacing=100*)

Write elevations from a DEM raster to a text file for use in mm3d PostProc Banana.

Parameters

- **fn_dem** (*str*) – the filename of the DEM to read.
- **fn_out** (*str*) – the name of the text file to write out (default: dem_pts.txt)
- **spacing** (*int*) – the pixel spacing of the DEM to write (default: every 100 pixels)

spymicmac.micmac.**generate_measures_files**(*joined=False*)

Create id_fiducial.txt, MeasuresCamera.xml, and Tmp-SL-Glob.xml files for KH-9 Hexagon images.

Parameters

joined (*bool*) – generate files for joined scene (220x460 mm) instead of half (220x230mm)

spymicmac.micmac.**get_autogcp_locations**(*ori*, *meas_file*, *imlist*)

Find location of automatically-detected control points in individual images using mm3d XYZ2Im.

Parameters

- **ori** (*str*) – The orientation directory name (e.g., Ori-Relative)
- **meas_file** (*str*) – The Measures file to find image locations for
- **imlist** (*list*) – a list of image names

spymicmac.micmac.**get_bascule_residuals**(*fn_basc*, *gcp_df*)

Read a given GCPBascule residual file, and add the residuals to a DataFrame with GCP information.

Parameters

- **fn_basc** (*str*) – the GCPBascule xml file to read the residuals from.
- **gcp_df** (*pandas.DataFrame*) – a DataFrame with the GCPs to read the residuals for.

Returns

gcp_df (*pandas.DataFrame*) – the input GCPs with the Bascule residuals added.

spymicmac.micmac.**get_campari_residuals**(*fn_resids*, *gcp_df*)

Read a given Campari residual file, and add the residuals to a DataFrame with GCP information.

Parameters

- **fn_resids** – the Campari residual xml file to read.
- **gcp_df** (*pandas.DataFrame*) – a DataFrame with the GCPs to read the residuals for.

Returns

gcp_df (*pandas.DataFrame*) – the input GCPs with the Campari residuals added.

spymicmac.micmac.**get_gcp_meas**(*im_name*, *meas_name*, *in_dir*, *E*, *nodist=None*, *gcp_name='GCP'*)

Create an lxml.builder.ElementMaker object with a GCP name and the image (row, pixel) location.

Parameters

- **im_name** (*str*) – the image name to write the GCP location for.

- **meas_name** (*str*) – the name of the file to read the point locations from.
- **in_dir** (*str*) – the name of the directory where the images and measures files are located.
- **E** (*lxml.builder.ElementMaker*) – an ElementMaker object for writing to the xml file.
- **nodist** (*str*) – the name of the directory
- **gcp_name** (*str*) – the prefix (e.g., GCP0, GCP1, etc.) for the GCP name (default: GCP).

Returns
this_im_meas (*lxml.builder.ElementMaker*) – an ElementMaker object with the GCP location in the image.

spymicmac.micmac.**get_im_meas**(*points_df*, *E*, *name='gcp'*, *x='im_col'*, *y='im_row'*)

Populate an lxml.builder.ElementMaker object with GCP image locations, for writing to xml files.

Parameters

- **points_df** (*pandas.DataFrame*) – a DataFrame with the points to find image locations for.
- **E** (*lxml.builder.ElementMaker*) – an ElementMaker object for writing to the xml file.
- **name** (*str*) – the column name in points_df corresponding to the point name [gcp]
- **x** (*str*) – the column name in points_df corresponding to the image x location [im_col]
- **y** (*str*) – the column name in points_df corresponding to the image y location [im_row]

Returns
pt_els (*list*) – a list of ElementMaker objects corresponding to each GCP image location.

spymicmac.micmac.**get_match_pattern**(*imlist*)

Given a list of image names, return a match pattern that can be passed to MicMac command line functions.

Parameters
imlist (*list*) – a list of image names.

Returns
pattern (*str*) – a match pattern (e.g., "OIS.*tif") that can be passed to MicMac functions.

spymicmac.micmac.**get_valid_image_points**(*shape*, *pts*, *pts_nodist*)

Find which image points are located within an image based on the size of the image.

Parameters

- **shape** – the shape of the image (rows, columns) to determine valid points for.
- **pts** (*pandas.DataFrame*) – a DataFrame containing point locations (i, j)
- **pts_nodist** (*pandas.DataFrame*) – a DataFrame containing point locations (i, j) calculated using no camera distortion.

Returns
valid_pts (*array-like*) – an array of the points that are located within the image shape.

spymicmac.micmac.**init_autocal**(*imsize=(32200, 15400)*, *framesize=(460, 220)*, *foc=304.8*, *camname='KH9MC'*)

Create an AutoCal xml file for use in the Tapas step. Default values are for KH-9 Hexagon Mapping Camera.

When calling mm3d Tapas, be sure to use "InCal=Init":

mm3d Tapas RadialBasic "OIS.*tif" InCal=Init Out=Relative LibFoc=0

The name of the file changes based on the focal length and camera name. Using the default values of foc=304.8 and camname='KH9MC' creates the following file in Ori-Init:

AutoCal_Foc-KH9MC_304800.xml

Parameters

- **imsize** (*array-like*) – the size of the image (width, height) in pixels (default: (32200, 15400))
- **framesize** (*array-like*) – the size of the image (width, height) in mm (default: (460, 220))
- **foc** (*float*) – nominal focal length, in mm (default: 304.8)
- **camname** (*int*) – the camera short name to use (default: KH9MC)

`spymicmac.micmac.`**`init_git`**`()`

Initialize a git repository in the current working directory.

`spymicmac.micmac.`**`iterate_campari`**`(`*gcps*, *out_dir*, *match_pattern*, *subscript*, *dx*, *ortho_res*, *fn_gcp='AutoGCPs'*, *fn_meas='AutoMeasures'*, *rel_ori='Relative'*, *inori='TerrainRelAuto'*, *outori='TerrainFinal'*, *homol='Homol'*, *allfree=True*, *max_iter=5*`)`

Run Campari iteratively, refining the orientation by removing outlier GCPs and Measures, based on their fit to the estimated camera model.

Parameters

- **gcps** (*pandas.DataFrame*) – a DataFrame with the GCPs that are being input to Campari.
- **out_dir** (*str*) – the output directory where the GCP and Measures files are located.
- **match_pattern** (*str*) – the match pattern for the images being input to Campari (e.g., “OIS.*tif”)
- **subscript** (*str*) – the name of the block, if multiple blocks are being used (e.g., '_block1'). If not, use ''.
- **dx** (*int | float*) – the pixel resolution of the reference image.
- **ortho_res** (*int | float*) – the pixel resolution of the orthoimage being used.
- **fn_gcp** (*str*) – the filename pattern for the GCP file. The file that will be loaded will be fn_gcp + sub + '.xml' (e.g., default: AutoGCPs -> AutoGCPs_block0.xml)
- **fn_meas** (*str*) – the filename pattern for the measures file. The file that will be loaded will be fn_meas + sub + '-S2D.xml' (e.g., default: AutoMeasures -> AutoMeasures_block0-S2D.xml)
- **rel_ori** (*str*) – the name of the relative orientation to input to GCPBascule (default: Relative -> Ori-Relative + sub)
- **inori** (*str*) – the input orientation to Campari (default: Ori-TerrainRelAuto -> TerrainRelAuto)
- **outori** (*str*) – the output orientation from Campari (default: Ori-TerrainFinal -> TerrainFinal)
- **homol** (*str*) – the Homologue directory to use (default: Homol)
- **allfree** (*bool*) – run Campari with AllFree=1 (True), or AllFree=0 (False). (default: True)
- **max_iter** (*int*) – the maximum number of iterations to run. (default: 5)

Returns
gcps (*pandas.DataFrame*) – the gcps with updated residuals after the iterative process.

`spymicmac.micmac.`**`malt`**`(`*imlist*, *ori*, *zoomf=1*, *zoomi=None*, *dirmec='MEC-Malt'*, *seed_img=None*, *seed_xml=None*`)`

Run mm3d Malt Ortho.

Parameters

- **imlist** (*str | iterable*) – either a match pattern (e.g., OIS.*tif) or an iterable object of image filenames.
- **ori** (*str*) – the orientation directory to use for Malt.
- **zoomf** (*int*) – the final Zoom level to use (default: 1)
- **zoomi** (*int*) – the initial Zoom level to use (default: not set)
- **dirmec** (*str*) – the output MEC directory to create (default: MEC-Malt)
- **seed_img** (*str*) – a DEM to pass to Malt as DEMInitImg. Note that if seed_img is set, seed_xml must also be set. (default: not used)
- **seed_xml** (*str*) – an XML file corresponding to the seed_img (default: not used)

`spymicmac.micmac.`**`mask_invalid_els`**`(`*dir_mec*, *fn_dem*, *fn_mask*, *ori*, *match_pattern='OIS.*tif'*, *zoomf=1*`)`

Mask invalid elevations (e.g., water) in a DEM, then re-run the final step of mm3d Malt Ortho to make nicer orthophotos.

Parameters

- **dir_mec** (*str*) – the MEC directory (e.g., MEC-Malt) to use
- **fn_dem** (*str*) – the filename of the reference DEM
- **fn_mask** (*str*) – filename for the mask vector file
- **ori** (*str*) – the orientation directory used to run Malt
- **match_pattern** (*str*) – the match pattern used to
- **zoomf** (*int*) – the final zoom level to run Malt at (default: ZoomF=1)

`spymicmac.micmac.`**`mosaic_micmac_tiles`**`(`*filename*, *dirname='.'*`)`

Re-stitch images tiled by MicMac.

Parameters

- **filename** (*str*) – MicMac filename to mosaic together
- **dirname** (*str*) – Directory containing images to Mosaic (default: .)

`spymicmac.micmac.`**`move_bad_tapas`**`(`*ori*`)`

Read residual files output from Tapas (or Campari, GCPBascule), and move images with a NaN residual.

Parameters
ori (*str*) – the orientation directory to read the residuals file from (e.g., ‘Ori-Relative’).

`spymicmac.micmac.`**`parse_im_meas`**`(`*fn_meas*`)`

Read an xml file with GCP image locations into a pandas DataFrame.

Parameters
fn_meas – the name of the measures file to read.

Returns
gcp_df (*pandas.DataFrame*) – a DataFrame with gcp names and image locations.

`spymicmac.micmac.`**`post_process`**`(`*projstr*, *out_name*, *dirmec*, *do_ortho=True*`)`

Apply georeferencing and masking to the final DEM and Correlation images (optionally, the orthomosaic as well).

Output files are written as follows:

- DEM: post_processed/{out_name}_Z.tif
- Hillshade: post_processed/{out_name}_HS.tif
- Correlation: post_processed/{out_name}_CORR.tif
- Orthomosaic: post_processed/{out_name}_Ortho.tif

Parameters

- **projstr** (*str*) – A string corresponding to the DEM's CRS that GDAL can use to georeference the rasters.
- **out_name** (*str*) – The name that the output files should have.
- **dirmec** (*str*) – The MEC directory to process files from (e.g., MEC-Malt)
- **do_ortho** (*bool*) – Post-process the orthomosaic in Ortho-{dirmec}, as well. Assumes that you have run mm3d Tawny with Out=Orthophotomosaic first.

`spymicmac.micmac.`**`remove_measure`**`(`*fn_meas*, *name*`)`

Remove all instances of a given measure from an xml file.

Parameters

- **fn_meas** (*str*) – the xml file (e.g., AutoMeasures-S2D.xml)
- **name** (*str*) – the measurement name (e.g., GCP0)

`spymicmac.micmac.`**`remove_worst_mesures`**`(`*fn_meas*, *ori*`)`

Remove outlier measures from an xml file, given the output from Campari.

Parameters

- **fn_meas** (*str*) – the filename for the measures file.
- **ori** (*str*) – the orientation directory output from Campari (e.g., Ori-TerrainFinal -> TerrainFinal)

`spymicmac.micmac.`**`rename_gcps`**`(`*root*, *ngcp=0*`)`

Rename all GCPs in order of their appearance in an (opened) xml file.

Parameters

- **root** (*xml.etree.ElementTree.Element*) – the root element of an xml tree
- **ngcp** (*int*) – the number to start counting from (defaults to 0)

Returns

- **mes_dict** (*dict*) – a dict containing image, gcp key/value pairs
- **gcp_dict** (*dict*) – a dict containing old/new gcp name key/value pairs

`spymicmac.micmac.`**`save_gcps`**`(`*in_gcps*, *outdir*, *utmstr*, *sub*, *fn_gcp='AutoGCPs'*, *fn_meas='AutoMeasures'*`)`

Save a GeoDataFrame of GCP information to shapefile, txt, and xml formats.

After running, the following new files will be created:

- outdir/fn_gcp.shp (+ associated files)

- outdir/fn_gcp.txt
- outdir/fn_gcp.xml (output from mm3d GCPConvert)
- outdir/fn_meas.xml (a file with image locations for each GCP)

Parameters

- **in_gcps** (*GeoDataFrame*) – the gcps GeoDataFrame to save
- **outdir** (*str*) – the output directory to save the files to
- **utmstr** (*str*) – a UTM string generated by register.get_utm_str()
- **sub** (*str*) – the name of the block, if multiple blocks are being used (e.g., '_block1'). If not, use ''.
- **fn_gcp** (*str*) – the filename pattern for the GCP file. The file that will be loaded will be fn_gcp + sub + '.xml' (e.g., default: AutoGCPs -> AutoGCPs_block0.xml)
- **fn_meas** (*str*) – the filename pattern for the measures file. The file that will be loaded will be fn_meas + sub + '-S2D.xml' (e.g., default: AutoMeasures -> AutoMeasures_block0-S2D.xml)

spymicmac.micmac.**tapas**(*cam_model*, *ori_out*, *img_pattern='OIS.*tif'*, *in_cal=None*, *lib_foc=True*, *lib_pp=True*, *lib_cd=True*)

Run mm3d Tapas with a given camera calibration model.

Some basic camera calibration models for air photos:

- RadialBasic
- RadialStd
- RadialExtended
- FraserBasic
- Fraser

See MicMac docs for a full list/explanation of the camera models.

Parameters

- **cam_model** (*str*) – the camera calibration model to use.
- **ori_out** (*str*) – the output orientation. Will create a directory, Ori-{ori_out}, with camera parameter files.
- **img_pattern** (*str*) – the image pattern to pass to Tapas (default: OIS.*tif)
- **in_cal** (*str*) – an input calibration model to refine (default: None)
- **lib_foc** (*bool*) – allow the focal length to be calibrated (default: True)
- **lib_pp** (*bool*) – allow the principal point to be calibrated (default: True)
- **lib_cd** (*bool*) – allow the center of distortion to be calibrated (default: True)

spymicmac.micmac.**tapioca**(*img_pattern='OIS.*tif'*, *res_low=400*, *res_high=1200*)

Run mm3d Tapioca MulScale

Parameters

- **img_pattern** (*str*) – The image pattern to pass to Tapioca (default: OIS.*tif)

- **res_low** (*int*) – the size of the largest image axis, in pixels, for low-resolution matching (default: 400)
- **res_high** (*int*) – the size of the largest image axis, in pixels, for high-resolution matching (default: 1200)

`spymicmac.micmac.`**`tawny`**`(`*dirmec*, *radiomegal=False*`)`

Run mm3d Tawny to create an orthomosaic.

Parameters

- **dirmec** (*str*) – the MEC directory to use
- **radiomegal** (*bool*) – run Tawny with RadiomEgal=1 (default: False)

`spymicmac.micmac.`**`write_auto_gcps`**`(`*gcp_df*, *sub*, *outdir*, *utm_zone*, *outname='AutoGCPs'*`)`

Write GCP name, x, y, and z information to a text file to use with mm3d GCPConvert.

Parameters

- **gcp_df** (*pandas.DataFrame*) – a DataFrame with the GCPs to save.
- **sub** (*str*) – the name of the block, if multiple blocks are being used (e.g., '_block1'). If not, use ''.
- **outdir** (*str*) – the output directory to save the files to.
- **utm_zone** (*str*) – the UTM zone name (e.g., 8N).
- **outname** (*str*) – the name to use for the GCPs file (default: AutoGCPs.txt)

`spymicmac.micmac.`**`write_auto_mesures`**`(`*gcps*, *sub*, *outdir*, *outname='AutoMeasures'*`)`

Write a file with GCP locations in relaive space (x, y, z) to use with get_autogcp_locations.sh

Parameters

- **gcps** (*pandas.DataFrame*) – a DataFrame with the GCPs to save.
- **sub** (*str*) – the name of the block, if multiple blocks are being used (e.g., '_block1'). If not, use ''.
- **outdir** (*str*) – the output directory to save the files to.
- **outname** (*str*) – the base name of the file to create (default: AutoMeasures).

`spymicmac.micmac.`**`write_image_mesures`**`(`*imlist*, *gcps*, *outdir='.'*, *sub=''*, *ort_dir='Ortho-MEC-Relative'*`)`

Create a Measures-S2D.xml file (row, pixel) for each GCP in each image from a list of image names.

Parameters

- **imlist** (*list*) – a list of image names.
- **gcps** (*pandas.DataFrame*) – a DataFrame of GCPs.
- **outdir** (*str*) – the output directory to save the files to.
- **sub** (*str*) – the name of the block, if multiple blocks are being used (e.g., '_block1').
- **ort_dir** (*str*) – the Ortho-MEC directory where the images are located.

`spymicmac.micmac.`**`write_neighbour_images`**`(`*imlist*, *fprints=None*, *nameField='ID'*, *prefix='OIS-Reech_'*, *fileExt='.tif'*, *dataset='AERIAL_COMBIN'*`)`

Using a list of images and a collection of image footprints, return a list of potential image pairs for processing with Tapioca.

Parameters

- **imlist** (*list*) – a list of (original) image names to use (e.g., without 'OIS-Reech_')
- **fprints** (*GeoDataFrame*) – a vector dataset of footprint polygons. If not provided, will attempt to download metadata from USGS for the images.
- **nameField** (*str*) – the field in fprints table that contains the image name
- **prefix** (*str*) – the prefix attached to the image name read by Tapioca (default: 'OIS-Reech_')
- **fileExt** (*str*) – the file extension for the images read by Tapioca (default: .tif)
- **dataset** – the USGS dataset name to search if no footprints are provided (default: AERIAL_COMBIN)

spymicmac.micmac.**write_xml**(*fn_img*, *fn_mask='./MEC-Malt/Masq_STD-MALT_DeZoom1.tif'*, *fn_xml=None*, *geomname='eGeomMNTEuclid'*)

Given a GDAL dataset, create a MicMac xml worldfile.

Parameters

- **fn_img** (*str*) – the filename of the image.
- **fn_mask** (*str*) – the filename of the mask file (default: ./MEC-Malt/Masq_STD-MALT_DeZoom1.tif)
- **fn_xml** (*str*) – the filename of the xml file to create (default: fn_img + '.xml')
- **geomname** (*str*) – the MicMac Geometry name to use (default: eGeomMNTEuclid)

4.1.6 spymicmac.orientation

spymicmac.orientation is a collection of tools for working with image orientation using micmac.

spymicmac.orientation.**block_orientation**(*blocks*, *meas_out='AutoMeasures'*, *gcp_out='AutoGCPs'*, *fn_mes='AutoMeasures_block'*, *fn_gcp='AutoGCPs_block'*, *dirname='auto_gcps'*, *rel_ori='Relative'*, *outori='TerrainFinal'*, *homol='Homol'*, *ref_dx=15*, *ortho_res=8*, *allfree=True*, *max_iter=1*)

Combine GCPs, Measures files, and Ori directories from multiple sub-blocks into a single file and orientation.

Parameters

- **blocks** (*list*) – a list of the sub-block numbers to combine
- **meas_out** (*str*) – the output filename for the Measures file (no extension). (default: AutoMeasures)
- **gcp_out** (*str*) – the output filename for the GCP file (no extension). (default: AutoGCPs)
- **fn_mes** (*str*) – the name pattern of the measures files to combine (default: AutoMeasures_block)
- **fn_gcp** (*str*) – the name pattern of the GCP files to combine (default: AutoGCPs_block)
- **dirname** (*str*) – the output directory where the files are saved (default: auto_gcps)
- **rel_ori** (*str*) – the name of the relative orientation to input to GCPBascule (default: Relative -> Ori-Relative)
- **outori** (*str*) – the output orientation from Campari (default: TerrainFinal -> Ori-TerrainFinal)

- **homol** (*str*) – the Homologue directory to use (default: Homol)
- **ref_dx** (*int | float*) – the pixel resolution of the reference image, in meters. (default: 15)
- **ortho_res** (*int | float*) – the pixel resolution of the orthoimage being used, in meters. (default: 8)
- **allfree** (*bool*) – run Campari with AllFree=1 (True), or AllFree=0 (False). (default: True)
- **max_iter** (*int*) – the maximum number of iterations to run. (default: 1)

Returns

gcps (*GeoDataFrame*) – the combined GCPs output from spymicmac.micmac.iterate_campari

spymicmac.orientation.**combine_block_measures**(*blocks*, *meas_out='AutoMeasures'*, *gcp_out='AutoGCPs'*, *fn_mes='AutoMeasures_block'*, *fn_gcp='AutoGCPs_block'*, *dirname='auto_gcps'*)

Combine GCPs and Measures files from multiple sub-blocks into a single file.

Parameters

- **blocks** (*list*) – a list of the sub-block numbers to combine
- **meas_out** (*str*) – the output filename for the Measures file (no extension). (default: AutoMeasures)
- **gcp_out** (*str*) – the output filename for the GCP file (no extension). (default: AutoGCPs)
- **fn_mes** (*str*) – the name pattern of the measures files to combine (default: AutoMeasures_block)
- **fn_gcp** (*str*) – the name pattern of the GCP files to combine (default: AutoGCPs_block)
- **dirname** (*str*) – the output directory where the files are saved (default: auto_gcps)

spymicmac.orientation.**extend_line**(*df*, *first*, *last*)

Extend a flightline using existing camera positions.

Parameters

- **df** (*GeoDataFrame*) – a GeoDataFrame containing the camera positions and image names
- **first** (*str*) – the name of the image to start interpolating from.
- **last** (*str*) – the name of the image to end interpolating at.

Returns

outpt (*shapely.Point*) – the new point along the flightline.

spymicmac.orientation.**fix_orientation**(*cameras*, *ori_df*, *ori*, *nsig=4*)

Correct erroneous Tapas camera positions using an estimated affine transformation between the absolute camera locations and the relative locations read from the orientation directory.

Once the positions have been updated, you should re-run Tapas using the InOri set to the directory; e.g., if you have updated Ori-Relative, you should run:

mm3d Tapas RadialBasic “OIS.*tif” InOri=Relative Out=Relative LibFoc=0

Parameters

- **cameras** (*pandas.DataFrame*) – A DataFrame containing camera positions (x, y, z) and a ‘name’ column that contains the image names.

- **ori_df** (*pandas.DataFrame*) – A DataFrame output from sPyMicMac.micmac.load_all_orientations, or that contains a 'name' column and camera positions in relative space (x, y, z)
- **ori** (*str*) – the Orientation directory to update (e.g., Ori-Relative)
- **nsig** (*int | float*) – the number of normalized absolute deviations from the median residual value to consider a camera an outlier (default: 4)

spymicmac.orientation.**interp_line**(*df*, *first*, *last*, *nimgs=None*, *pos=None*)

Interpolate camera positions along a flightline.

Parameters

- **df** (*GeoDataFrame*) – a GeoDataFrame containing the camera positions and image names
- **first** (*str*) – the name of the image to start interpolating from.
- **last** (*str*) – the name of the image to end interpolating at.
- **nimgs** (*int*) – the number of images to interpolate (default: calculated based on the image numbers)
- **pos** (*int*) – which image position to return (default: all images between first and last)

Returns

ptList (*list*) – a list containing the interpolated camera positions (or, a tuple of the requested position).

spymicmac.orientation.**load_all_orientation**(*ori*, *imlist=None*)

Load all of the orientation parameters for a set of images from a given directory.

Parameters

- **ori** (*str*) – the orientation directory to read
- **imlist** (*list*) – the images to load. If not set, loads all orientation files from the given directory.

Returns

ori_df (*pandas.DataFrame*) – a DataFrame containing the orientation parameters for each image

spymicmac.orientation.**load_orientation**(*fn_img*, *ori*)

Read camera position and rotation information from an Orientation xml file.

Parameters

- **fn_img** (*str*) – the name of the image to read the orientation file for.
- **ori** (*str*) – the name of the orientation directory (e.g., Ori-Relative).

Returns

- **centre** (*list*) – the camera position (x, y, z)
- **l1** (*list*) – the L1 orientation parameters
- **l2** (*list*) – the L2 orientation parameters
- **l3** (*list*) – the L3 orientation parameters
- **prof** (*float*) – the 'Profondeur' value from the xml file.
- **altisol** (*float*) – the 'AltiSol' value from the xml file.

spymicmac.orientation.**transform_centers**(*img_gt*, *ref*, *imlist*, *footprints*, *ori*, *imgeom=True*)

Use the camera centers in relative space provided by MicMac Orientation files, along with camera footprints, to estimate a transformation between the relative coordinate system and the absolute coordinate system.

Parameters

- **img_gt** (*array-like*) – the image GeoTransform (as read from a TFW file)
- **ref** (*GeoImg*) – the reference image to use to determine the output image shape
- **imlist** (*list*) – a list of the images that were used for the relative orthophoto
- **footprints** (*GeoDataFrame*) – the (approximate) image footprints - the centroid will be used for the absolute camera positions.
- **ori** (*str*) – name of orientation directory
- **imgeom** (*bool*) – calculate a transformation between image ij locations (True)

Returns

- **model** (*AffineTransform*) – the estimated Affine Transformation between relative and absolute space
- **inliers** (*array-like*) – a list of the inliers returned by skimage.measure.ransac
- **join** (*GeoDataFrame*) – the joined image footprints and relative orientation files

spymicmac.orientation.**transform_points**(*ref*, *ref_pts*, *rel_gt*, *rel_pts*)

Given x,y points and two "geo"-referenced images, finds an affine transformation between the two images.

Parameters

- **ref** (*GeoImg*) – the reference image
- **ref_pts** (*np.array*) – an Mx2 array of the x,y points in the reference image
- **rel_gt** (*array-like*) – the "geo" transform for the second image, as read from a .tfw file.
- **rel_pts** (*np.array*) – an Mx2 array of the x,y points in the second image.

Returns

- **model** (*AffineTransform*) – the estimated Affine Transformation between relative and absolute space
- **inliers** (*array-like*) – a list of the inliers returned by skimage.measure.ransac

spymicmac.orientation.**update_center**(*fn_img*, *ori*, *new_center*)

Update the camera position in an Orientation file.

Parameters

- **fn_img** (*str*) – the name of the image to update the orientation for (e.g., 'OIS-Reech_ARCSEA000590122.tif')
- **ori** (*str*) – the name of the orientation directory (e.g., 'Ori-Relative')
- **new_center** (*list*) – a list of the new camera position [x, y, z]

spymicmac.orientation.**update_params**(*fn_img*, *ori*, *profondeur*, *altisol*)

Update the profondeur and altisol parameters in an orientation file.

Parameters

- **fn_img** (*str*) – the name of the image to update the orientation for (e.g., 'OIS-Reech_ARCSEA000590122.tif')

- **ori** (*str*) – the name of the orientation directory (e.g., 'Ori-Relative')
- **profondeur** (*float*) – the new profondeur value
- **altisol** (*float*) – the new altisol value

spymicmac.orientation.**update_pose**(*fn_img*, *ori*, *new_rot*)

Update the camera pose (rotation matrix) in an Orientation file.

Parameters

- **fn_img** (*str*) – the name of the image to update the orientation for (e.g., 'OIS-Reech_ARCSEA000590122.tif')
- **ori** (*str*) – the name of the orientation directory (e.g., 'Ori-Relative')
- **new_rot** (*array-like*) – the new 3x3 rotation matrix

4.1.7 spymicmac.register

spymicmac.register is a collection of tools for registering images and finding GCPs.

spymicmac.register.**register_individual**(*dir_ortho*, *fn_ref*, *fn_reldem*, *fn_dem*, *glacmask=None*, *landmask=None*, *footprints=None*, *im_subset=None*, *block_num=None*, *ori='Relative'*, *ortho_res=8.0*, *imgsource='DECLASSII'*, *density=200*, *out_dir='auto_gcps'*, *allfree=True*, *is_geo=False*)

IN DEVELOPMENT: Register individual orthoimages to a reference orthorectified image and DEM.

Parameters

- **dir_ortho** (*str*) – directory containing orthoimages (e.g., Ortho-MEC-Relative)
- **fn_ref** (*str*) – path to reference orthorectified image
- **fn_reldem** (*str*) – path to relative DEM
- **fn_dem** (*str*) – path to reference DEM
- **glacmask** (*str*) – path to file of glacier outlines (i.e., an exclusion mask)
- **landmask** (*str*) – path to file of land outlines (i.e., an inclusion mask)
- **footprints** (*str*) – path to shapefile of image outlines. If not set, will download from USGS.
- **im_subset** (*str*) – subset of raw images to work with
- **block_num** (*str*) – block number to use if processing multiple image blocks
- **ori** (*str*) – name of orientation directory (after Ori-) (default: Relative)
- **ortho_res** (*float*) – approx. ground sampling distance (pixel resolution) of ortho image (default: 8 m)
- **imgsource** (*str*) – USGS dataset name for images (default: DECLASSII)
- **density** (*int*) – pixel spacing to look for GCPs (default: 200)
- **out_dir** (*str*) – output directory to save auto GCP files to (default: auto_gcps)
- **allfree** (*bool*) – run Campari setting all parameters free (default: True)
- **is_geo** (*bool*) – True if orthoimages are already in absolute coordinates (default: False)

`spymicmac.register.`**`register_relative`**(*dirmec*, *fn_dem*, *fn_ref=None*, *fn_ortho=None*, *glacmask=None*, *landmask=None*, *footprints=None*, *im_subset=None*, *block_num=None*, *subscript=None*, *ori='Relative'*, *ortho_res=8.0*, *imgsource='DECLASSII'*, *density=200*, *out_dir=None*, *allfree=True*, *useortho=False*, *max_iter=5*)

Register a relative DEM or orthoimage to a reference DEM and/or orthorectified image.

Parameters

- **`dirmec`** (*`str`*) – the name of the MEC directory to read the relative DEM from (e.g., MEC-Relative)
- **`fn_dem`** (*`str`*) – path to reference DEM
- **`fn_ref`** (*`str`*) – path to reference orthorectified image (optional)
- **`fn_ortho`** (*`str`*) – path to relative orthoimage (optional)
- **`glacmask`** (*`str`*) – path to file of glacier outlines (i.e., an exclusion mask)
- **`landmask`** (*`str`*) – path to file of land outlines (i.e., an inclusion mask)
- **`footprints`** (*`str`*) – path to shapefile of image outlines. If not set, will download from USGS.
- **`im_subset`** (*`str`*) – subset of raw images to work with
- **`block_num`** (*`str`*) – block number to use if processing multiple image blocks
- **`subscript`** (*`str`*) – optional subscript to use for output filenames (default: None)
- **`ori`** (*`str`*) – name of orientation directory (after Ori-) (default: Relative)
- **`ortho_res`** (*`float`*) – approx. ground sampling distance (pixel resolution) of ortho image (default: 8 m)
- **`imgsource`** (*`str`*) – USGS dataset name for images (default: DECLASSII)
- **`density`** (*`int`*) – pixel spacing to look for GCPs (default: 200)
- **`out_dir`** (*`str`*) – output directory to save auto GCP files to (default: auto_gcps)
- **`allfree`** (*`bool`*) – run Campari setting all parameters free (default: True)
- **`useortho`** (*`bool`*) – use the orthomosaic in Ortho-{dirmec} rather than the DEM (default: False). If fn_ortho is set, uses that file instead.
- **`max_iter`** (*`int`*) – the maximum number of Campari iterations to run. (default: 5)

`spymicmac.register.`**`warp_image`**(*model*, *ref*, *img*)

Given a transformation model between two coordinate systems, warp an image to a reference image

Parameters

- **`model`** (*`GeometricTransform`*) – the transformation model between the coordinate systems
- **`ref`** (*`GeoImg`*) – the reference GeoImg
- **`img`** (*`GeoImg`*) – the GeoImg to be transformed

Returns

- **tfm_img** (*np.array*) – the input image transformed to the same extent as the reference image
- **this_model** (*AffineTransform*) – the estimated Affine Transformation between the two images
- **inliers** (*array-like*) – a list of the inliers returned by skimage.measure.ransac

4.1.8 spymicmac.resample

spymicmac.resample is a collection of tools for resampling images

`spymicmac.resample.`**`downsample`**`(`*img*, *fact=4*`)`

Rescale an image using Lanczos resampling

Parameters

- **`img`** (*`array-like`*) – the image to rescale
- **`fact`** (*`numeric`*) – the number by which to divide the image width and height (default: 4)

Returns

rescaled (*array-like*) – the rescaled image

`spymicmac.resample.`**`resample_hex`**`(`*fn_img*, *scale*, *ori='InterneScan'*`)`

Resample a KH-9 Mapping Camera image based on the reseau grid, using gdal.Warp

Parameters

- **`fn_img`** (*`str`*) – the filename of the image to resample
- **`scale`** (*`int`*) – the number of pixels per mm of the scanned image
- **`ori`** (*`str`*) – the Ori directory that contains both MeasuresCamera.xml and MeasuresIm (default: InterneScan)

`spymicmac.resample.`**`resample_kh4`**`(`*img*`)`

INCOMPLETE

Parameters

`img` (*`array-like`*) – the image to resample

Returns

`spymicmac.resample.`**`rotate_kh4`**`(`*img*`)`

Use the rail marks in a KH-4 image to rotate the image.

Parameters

`img` (*`array-like`*) – the image to rotate.

Returns

rotated (*array-like*) – the rotated image

4.2 shell scripts

4.2.1 balance_images

Apply Contrast-limited Adaptive Histogram Equalization (CLAHE) to all re-sampled images in current directory.

```
usage: balance_images [-h]
```

4.2.2 block_orientation

Combine GCPs, Measures files, and Ori directories from multiple sub-blocks into a single file and orientation.

```
usage: block_orientation [-h] [-meas_out MEAS_OUT] [-gcp_out GCP_OUT]
                         [-fn_mes FN_MES] [-fn_gcp FN_GCP] [-d DIRNAME]
                         [-r REL_ORI] [-o OUTORI] [-homol HOMOL]
                         [-ref_dx REF_DX] [-ortho_res ORTHO_RES] [-allfree]
                         [-max_iter MAX_ITER]
                         blocks [blocks ...]
```

Positional Arguments

blocks	the block numbers to combine

Named Arguments

-meas_out	the output filename for the Measures file (no extension). (default: AutoMeasures) Default: “AutoMeasures”
-gcp_out	the output filename for the GCP file (no extension). (default: AutoGCPs) Default: “AutoGCPs”
-fn_mes	the name pattern of the measures files to combine (default: AutoMeasures_block) Default: “AutoMeasures_block”
-fn_gcp	the name pattern of the GCP files to combine (default: AutoGCPs_block) Default: “AutoGCPs_block”
-d, --dirname	the output directory where the files are saved (default: auto_gcps) Default: “auto_gcps”
-r, --rel_ori	the relative orientation to input to GCPBascule (default: Relative -> Ori-Relative) Default: “Relative”
-o, --outori	the output orientation from Campari (default: TerrainFinal -> Ori-TerrainFinal) Default: “TerrainFinal”
-homol	the Homologue directory to use (default: Homol) Default: “Homol”
-ref_dx	the pixel resolution of the reference image, in meters. (default: 15) Default: 15
-ortho_res	the pixel resolution of the orthoimage being used, in meters. (default: 8) Default: 8
-allfree	run Campari with AllFree set to False Default: True

-max_iter	the maximum number of iterations to run. (default: 1)
	Default: 1

4.2.3 combine_auto_measures

Combine outputs of XYZ2Im into single xml file for further processing

```
usage: combine_auto_measures [-h] [-o OUT_FILE] [-n] ij_files [ij_files ...]
```

Positional Arguments

ij_files	txt files containing image i,j points

Named Arguments

-o, --out_file	output filename [AutoMeasures.xml]
	Default: “MeasuresAuto.xml”
-n, --no_distortion	Use gcp locations computed assuming no distortion to filter GCPs from images.
	Default: False

4.2.4 create_localchantier_xml

Create a MicMac-LocalChantierDescripteur.xml file

```
usage: create_localchantier_xml [-h] [-n NAME] [-s SHORT_NAME]
                                [--film_size FILM_SIZE FILM_SIZE] [-p PATTERN]
                                [-f FOCAL] [--add_sfs]
```

Named Arguments

-n, --name	Camera name to use. [Default: KH9MC]
	Default: “KH9MC”
-s, --short_name	Short description of camera. [Default: KH-9 Hexagon Mapping Camera]
	Default: “KH-9 Hexagon Mapping Camera”
--film_size	Film size (width, height), in mm. [Default: 460, 220]
	Default: (460, 220)
-p, --pattern	Camera name to use. [Default: .*]
	Default: “.*”
-f, --focal	Camera focal length (in mm). [Default: 304.8]
	Default: 304.8

--add_sfs	Use SFS for tie point matching [Default: False]. Default: False

4.2.5 download_cop30_vrt

Create a VRT using Copernicus 30m DSM tiles that intersect image footprints. Creates Copernicus_DSM.vrt using files downloaded to cop30_dem/ within the current directory.

```
usage: download_cop30_vrt [-h] [-imlist IMLIST [IMLIST ...]]
                          [-footprints FOOTPRINTS] [-imgsource IMGSOURCE]
```

Named Arguments

-imlist	image(s) to use for geographic extent. If not set, will search for images of form OIS*.tif
-footprints	filename for image footprints. By default, downloads footprints from USGS Earth Explorer.
-imgsource	the EE Dataset name for the images (default: DECLASSII) Default: “DECLASSII”

4.2.6 find_reseau_grid

Find Reseau marks in a scanned KH-9 Hexagon image.

```
usage: find_reseau_grid [-h] [-csize CSIZE] [-n NPROC] img [img ...]
```

Positional Arguments

img	Image(s) to find Reseau marks in.

Named Arguments

-csize	Reseau mark template size [361 pixels] Default: 361
-n, --nproc	number of sub-processes to use [Default: 1]. Default: 1

4.2.7 generate_micmac_measures

`generate_micmac_measures` calls `spymicmac.micmac.generate_micmac_measures()` to create `id_fiducial.txt`, `MeasuresCamera.xml`, and `Tmp-SL-Glob.xml` files for KH-9 Hexagon images.

Warning: Once created, you should move `MeasuresCamera.xml` into the `Ori-InterneScan` directory so that it can be found by `mm3d ReSampFid` or `spymicmac.image.resample_hex()`.

Note: To use the `Tmp-SL-Glob.xml` file, copy it into the `Tmp-SaisieAppuis` directory with the name of the image appended; e.g.,:

```
cp Tmp-SL-Glob.xml Tmp-SaisieAppuis/Tmp-SL-Glob-MeasuresIm-DZB1215-500425L002001.tif.xml
```

You will also need MeasuresIm-DZB1215-500425L002001.tif-S2D.xml (or whatever the image name is) to exist in the directory:

```
cp Ori-InterneScan/MeasuresIm-DZB1215-500425L002001.tif.xml MeasuresIm-DZB1215-
↪500425L002001.tif-S2D.xml
```

and, you should remove any temporary files from `Tmp-SaisieAppuis`:

```
rm Tmp-SaisieAppuis/Tmp-SL-Im-MeasuresIm-MeasuresIm-DZB1215-500425L002001.tif.xml.*
```

Create MicMac-LocalChantierDescripteur.xml, AutoCal_Foc-304800_KH9MC.xml, id_fiducial.txt, MeasuresCamera.xml, and Tmp-SL-Glob.xml files for KH-9 Hexagon Mapping Camera images.

```
usage: generate_micmac_measures [-h] [-joined]
```

Named Arguments

-joined	generate files for joined scene (220x460 mm) instead of half (220x230mm) Default: False

4.2.8 join_hexagon

Join parts of a scanned image

```
usage: join_hexagon [-h] [-p PATTERN] [-o OVERLAP] [-k BLOCK_SIZE] [-b] [-r]
```

Named Arguments

-p, --pattern	Match pattern for images [DZB] Default: "DZB"
-o, --overlap	overlap search width between two images [2000] Default: 2000
-k, --block_size	the number of rows each sub-block should cover. Defaults to overlap value.
-b, --blend	Blend across image halves to prevent a sharp line at edge. Default: False
-r, --reversed	parts are in reversed order (i.e., part b is the left part, part a is the right part) Default: False

4.2.9 mosaic_micmac_tiles

Re-stitch images tiled by MicMac.

```
usage: mosaic_micmac_tiles [-h] [-imgdir IMGDIR] filename
```

Positional Arguments

filename	MicMac filename to mosaic together

Named Arguments

-imgdir	Directory containing images to Mosaic (default: .) Default: "."

4.2.10 post_process_micmac

```
usage: post_process_micmac [-h] [--do_ortho] projstr out_name dirmec
```

Positional Arguments

projstr	A string corresponding to the DEM's CRS that GDAL can use to georeference the rasters.
out_name	The name that the output files should have.
dirmec	The MEC directory to process files from (e.g., MEC-Malt)

Named Arguments

--do_ortho	Post-process the orthomosaic in Ortho-{dirmec}, as well. Assumes that you have runmm3d Tawny with Out=Orthophotomosaic first. Default: False

4.2.11 preprocess_kh9

Run pre-processing steps for KH-9 Hexagon Mapping Camera images. By default, runs all steps (equivalent to "–steps all"):

- extract: extracts images from tar files (skips if no tar files are found)
- join: joins scanned image halves
- reseau: finds reseau marker locations in the joined image
- erase: erases reseau markers from image
- filter: use a 1-sigma gaussian filter to smooth the images before resampling
- resample: resamples images to common size using the reseau marker locations
- tapioca: calls mm3d Tapioca MulScale to find tie points
- tapas: calls mm3d Tapas to calibrate camera model, find relative image orientation
- aperi: calls mm3d AperiCloud to create point cloud using calibrated camera model

To run steps individually, use the –steps flag with the corresponding step name(s). For example, to only run the 'reseau' and 'erase' steps:

preprocess_kh9 –steps reseau erase <additional arguments>

```
usage: preprocess_kh9 [-h] [--steps STEPS [STEPS ...]] [--tar_ext TAR_EXT]
                      [-s SCALE] [-b] [--add_sfs] [--res_low RES_LOW]
                      [--res_high RES_HIGH] [--camera_model CAMERA_MODEL]
                      [--ori ORI] [--init_cal INIT_CAL] [--lib_foc] [--lib_pp]
                      [--lib_cd] [-n NPROC]
```

Named Arguments

--steps	The pre-processing steps to run. Default: "all"
--tar_ext	Extension for tar files (default: .tgz) Default: ".tgz"
-s, --scale	The scale of the resampled images, in pixels per mm. (default: 70) Default: 70
-b, --blend	Blend across image halves to prevent a sharp line at edge. Default: False

--add_sfs	use SFS to help find tie points in low-contrast images [False] Default: False
--res_low	the size of the largest image axis, in pixels, for low-resolution matching with Tapioca (default: 400) Default: 400
--res_high	the size of the largest image axis, in pixels, for low-resolution matching with Tapioca (default: 1200) Default: 1200
--camera_model	The camera calibration model to use for Tapas (default: RadialExtended) Default: "RadialExtended"
--ori	The output Ori directory to create using Tapas (default: Relative) Default: "Relative"
--init_cal	The initial calibration Ori to use for Tapas (default: Init) Default: "Init"
--lib_foc	Use LibFoc=1 for mm3d Tapas (default: LibFoc=0) Default: False
--lib_pp	Use LibPP=1 for mm3d Tapas (default: LibPP=0) Default: False
--lib_cd	Use LibCD=1 for mm3d Tapas (default: LibCD=0) Default: False
-n, --nproc	number of sub-processes to use (default: 1). Default: 1

4.2.12 register_relative

Register a relative DEM or orthoimage to a reference DEM and/or orthorectified image.

```
usage: register_relative [-h] [-ort FN_ORTHO] [-ref FN_REF]
                         [-glacmask GLACMASK] [-landmask LANDMASK]
                         [-footprints FOOTPRINTS]
                         [-im_subset IM_SUBSET [IM_SUBSET ...]] [-b BLOCK_NUM]
                         [--subscript SUBSCRIPT] [-ori ORI]
                         [-ortho_res ORTHO_RES] [-imgsource IMGSOURCE]
                         [-density DENSITY] [-no_allfree] [-useortho]
                         [-max_iter MAX_ITER]
                         dirmec fn_dem
```

Positional Arguments

dirmec	the name of the MEC directory to read the relative DEM from (e.g., MEC-Relative)
fn_dem	path to reference DEM

Named Arguments

-ort, --fn_ortho	path to relative orthoimage (optional)
-ref, --fn_ref	path to reference orthorectified image (optional)
-glacmask	path to shapefile of glacier outlines (i.e., an exclusion mask)
-landmask	path to shapefile of land outlines (i.e., an inclusion mask)
-footprints	path to shapefile of image outlines. If not set, will attempt to download from USGS.
-im_subset	subset of raw images to work with (default all)
-b, --block_num	Block number to use if multiple image blocks exist in directory.
--subscript	Optional subscript to add to filenames.
-ori	name of orientation directory (after Ori-) [Relative] Default: “Relative”
-ortho_res	approx. ground sampling distance (pixel resolution) of ortho image. [8 m] Default: 8
-imgsource	USGS dataset name for images [DECLASSII] Default: “DECLASSII”
-density	pixel spacing to look for GCPs [200] Default: 200
-no_allfree	run Campari with AllFree set to False Default: True
-useortho	use the orthomosaic in Ortho-{dirmec} rather than the DEM [False]. If fn_ortho is set, uses that file instead. Default: False
-max_iter	the maximum number of Campari iterations to run [5] Default: 5

4.2.13 remove_crosses

Remove Reseau marks from KH-9 image(s).

```
usage: remove_crosses [-h] [-n NPROC] img [img ...]
```

Positional Arguments

img	Image(s) to remove crosses from.

Named Arguments

-n, --nproc	number of sub-processes to use (default: 1). Default: 1

4.2.14 remove_measures

Remove GCP(s) from a Measures xml file.

```
usage: remove_measures [-h] fn_meas gcp [gcp ...]
```

Positional Arguments

fn_meas	xml file to remove GCP(s) from.
gcp	GCP name(s) to remove from <fn_meas>.

4.2.15 resample_hexagon

Use a piecewise affine transformation to resample KH-9 images using the reseau grid

```
usage: resample_hexagon [-h] [-s SCALE] [-o ORI] [-n NPROC] img [img ...]
```

Positional Arguments

img	Image(s) to resample from.

Named Arguments

-s, --scale	The scale of the resampled image, in pixels per mm. [Default: 70] Default: 70
-o, --ori	The Ori directory that contains both MeasuresCamera.xml and MeasuresIm for each image. [Default: InterneScan] Default: "InterneScan"
-n, --nproc	number of sub-processes to use [Default: 1]. Default: 1

4.2.16 write_micmac_xml

Given a GDAL dataset, create a MicMac xml worldfile.

```
usage: write_micmac_xml [-h] [-m MASK] [-g GEOM] filename
```

Positional Arguments

filename	the filename of the image.

Named Arguments

-m, --mask	Path to mask file [./MEC-Malt/Masq_STD-MALT_DeZoom1.tif] Default: "./MEC-Malt/Masq_STD-MALT_DeZoom1.tif"
-g, --geom	MicMac Geometry name [eGeomMNTEuclid] Default: "eGeomMNTEuclid"

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Crippen1989] Crippen, R. E. (1989) “A simple spatial filtering routine for the cosmetic removal of scan-line noise from Landsat TM P-tape imagery.” *Photogrammetric Engineering & Remote Sensing*, 55(3):327–31

PYTHON MODULE INDEX

INDEX