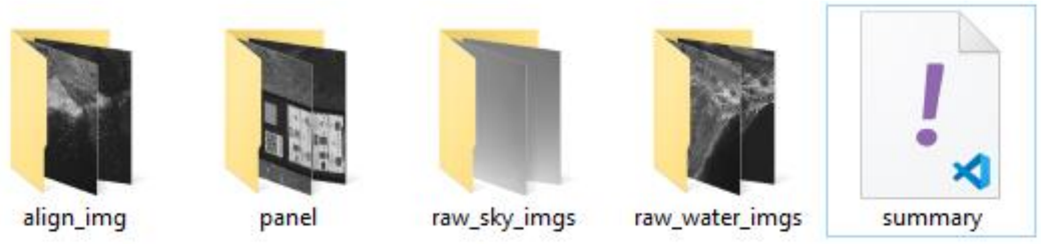
MOSAICKING CODE: HOW-TO-USE

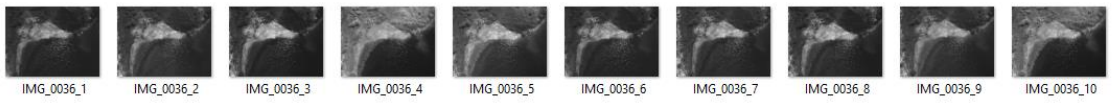
Alejandro Román & Sergio Heredia

1. DATASET STRUCTURE.

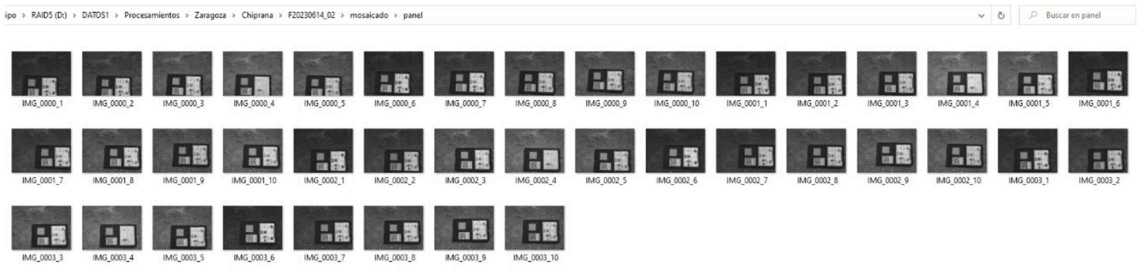
The first thing we need to do is structure the data into the following subfolders:



- align_img: Contains a reference image used to align the rest of the dataset captures. Ideally, it should be an intermediate capture from the dataset.

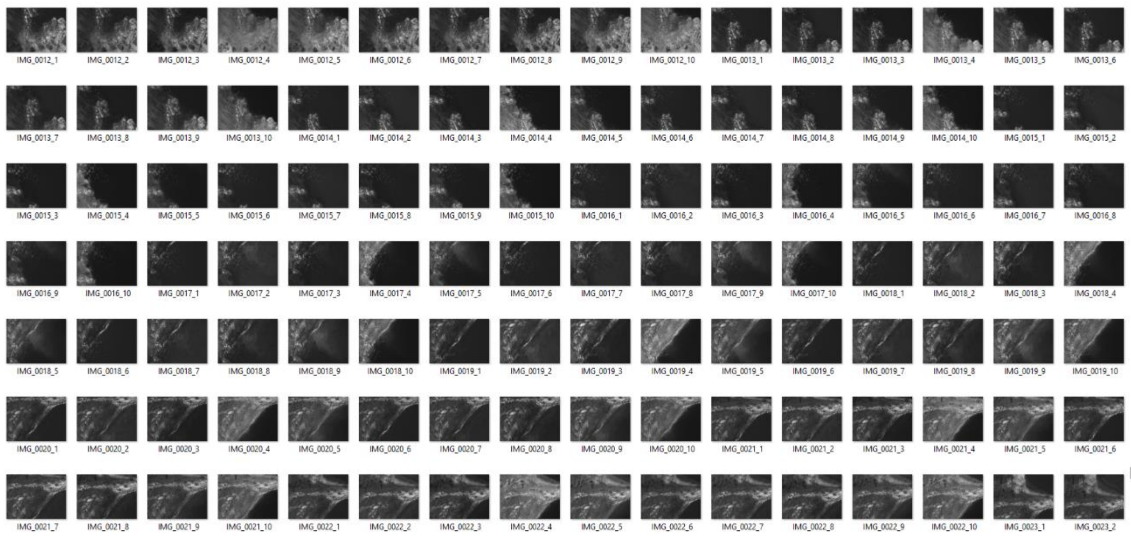


- panel: Contains captures taken on the MicaSense sensor calibration panel, in case it may be necessary.


ipo › RAID5 (D:) › DATOS1 › Procesamientos › Zaragoza › Chiprana › F20230614_02 › mosaicado › panel
Buscar en panel

- raw_sky_images: Contains the directory with all the images taken of the sky at a 40-degree angle from zenith and an approximate 135-degree azimuthal viewing direction.

- raw_water_images: Contains all the captures taken over the water surface, i.e., the complete flight dataset.



- summary.yml: It is a file used to define flight metadata, primarily serving to determine which captures to georeference and stitch together.

1.1. SUMMARY.YML FILE

The summary.yml file needs to be configured before starting the processing. This step might be considered the most "tedious" in the entire mosaic code because it requires manually reviewing all the flight lines and configuring the file accordingly, depending on whether the flight has a high or low overlap.

COMMON TO ALL CASES (WITH AND WITHOUT OVERLAP)

The "**Description**" field, which includes general flight information, should be filled as follows:

- name: The name we give to the flight (any desired name).

- bands: The number of bands in the multispectral sensor (5 or 10, depending on whether it is dual or not).

- sensor: In this case, "micasense".

- path: We define this in the code itself, so we can leave it as an empty string (' ').

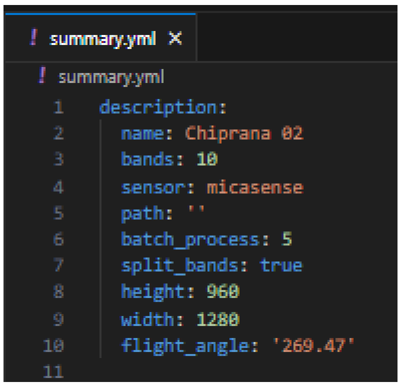
- height: The height of the image in pixels.

- weight: The width of the image in pixels.

- batch_process: The number of captures processed simultaneously (not currently used).

- split_bands: If set to "true", the bands will be exported separately.

- flight_angle: Here, we select the flight angle.



summary.yml

```
description:
  name: Chiprana 02
  bands: 10
  sensor: micasense
  path: ''
  batch_process: 5
  split_bands: true
  height: 960
  width: 1280
  flight_angle: '269.47'
```

The "**Flight**" field will include the information for each flight line. The process needs to be repeated for each flight line contained in the dataset. It is important to note that we do not consider the captures during turns, as they introduce aberrations in the final result. The field should be filled as follows:

- start: A numeric index indicating the capture that starts a flight line (starting from 0 in Python).

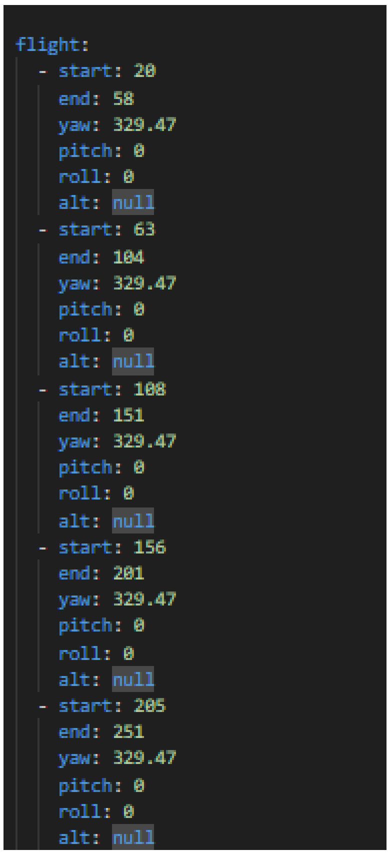
- end: A numeric index indicating the capture that ends a flight line.

- yaw: The angle at which the drone is facing (flight angle). This value will be constant for all flight lines.

- pitch: The pitch angle. We will always set it to 0.

- roll: The roll angle. We will always set it to 0.

- alt: The altitude of the captures within the same flight line. We will always set it to "null".



```
flight:
  - start: 20
    end: 58
    yaw: 329.47
    pitch: 0
    roll: 0
    alt: null
  - start: 63
    end: 104
    yaw: 329.47
    pitch: 0
    roll: 0
    alt: null
  - start: 108
    end: 151
    yaw: 329.47
    pitch: 0
    roll: 0
    alt: null
  - start: 156
    end: 201
    yaw: 329.47
    pitch: 0
    roll: 0
    alt: null
  - start: 205
    end: 251
    yaw: 329.47
    pitch: 0
    roll: 0
    alt: null
```

<u>*SPECIFIC CASE OF FLIGHTS WITH OVERLAP*</u>

Ideally, the drone flight should be without overlap because it covers a larger area and results in fewer artifacts in the final orthomosaic. However, in flights with overlap, we introduce a third field to separate even-numbered flight lines from odd-numbered flight lines. This field is called "**merge_partitions**":

- name: The name of the merge type ("even" for even-numbered lines and "odd" for odd-numbered lines).

- start: A numeric index indicating the first even or odd flight line to be included in the processing (remember that the first line is line 0 in Python).

- end: A numeric index indicating the last even or odd flight line to be included in the processing.

- steps: Indicates the number of lines we skip. It will always be 2.

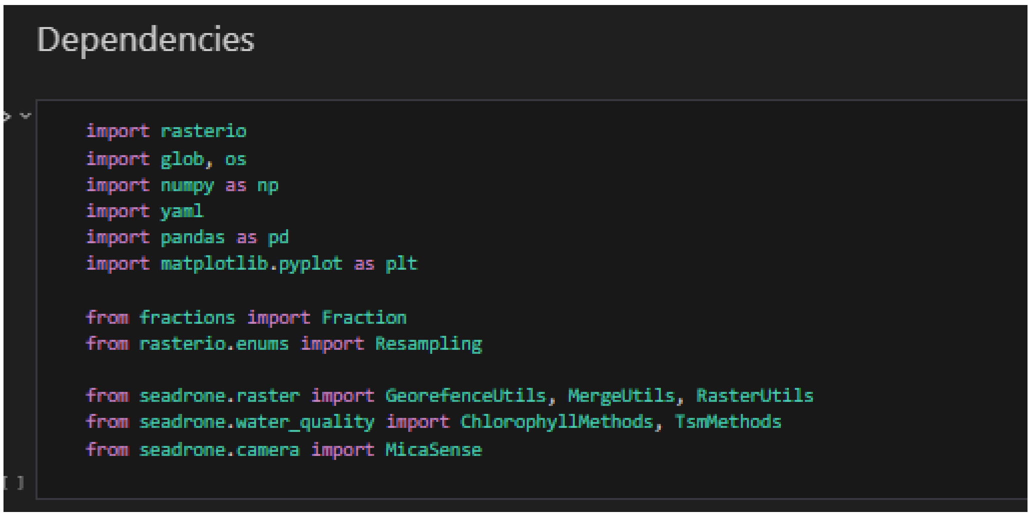
Note: The above information assumes that the flight lines are numbered consecutively, with even and odd lines alternating.

2. START OF "MONO_PROCESSING_JUPYTER" PROCESSING

Here, we would begin modifying the code to carry out the combined processing of DroneWQ and the mosaic code. The first step is to access the code structure, specifically the "**mono_processing_jupyter**" folder. In this guide, we will explain this option, which is a step-by-step process for a specific dataset. However, we will also explain a way to do it in batch later on.

Open the "**direct_georeference_products**" file, and below is a line-by-line description:

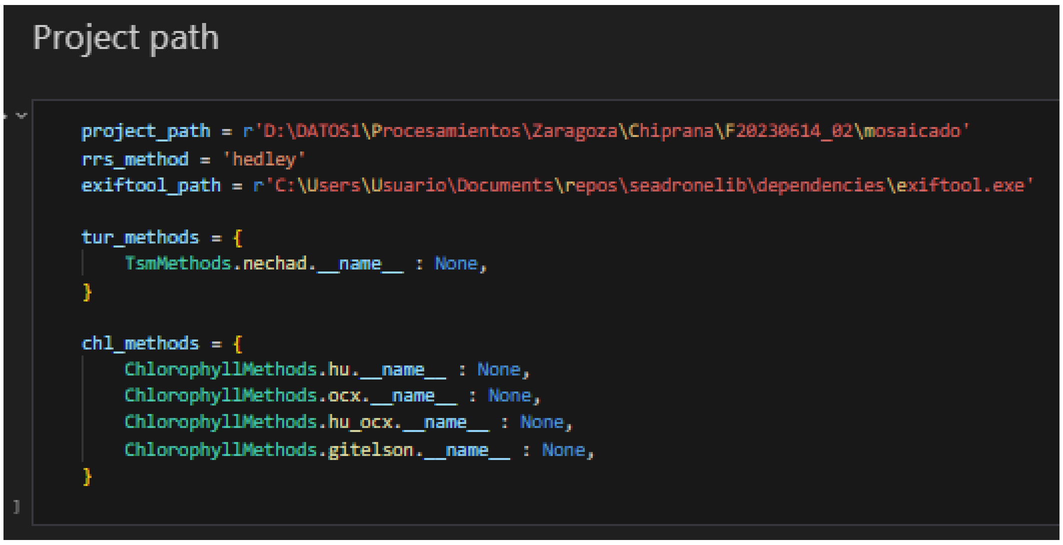
- Dependencies: This includes all the necessary libraries to run the code.



Dependencies

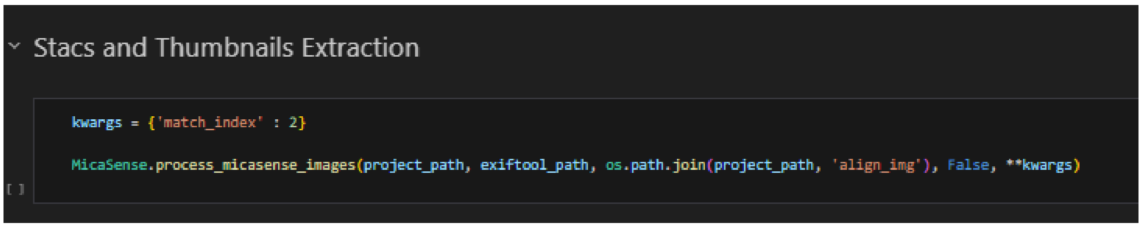
```
import rasterio
import glob, os
import numpy as np
import yaml
import pandas as pd
import matplotlib.pyplot as plt

from fractions import Fraction
from rasterio.enums import Resampling

from seadrone.raster import GeorefenceUtils, MergeUtils, RasterUtils
from seadrone.water_quality import ChlorophyllMethods, TsmMethods
from seadrone.camera import MicaSense
```

- Project path: This section contains the dataset configuration. In this case, we need to specify the "project_path" with the full path of the dataset, the "rrs_method" which determines how we derive rrs ("hedley", "mobley_rho", or "blackpixel"), and the "exiftool_path" with the full path of the exiftool.exe file.


Project path

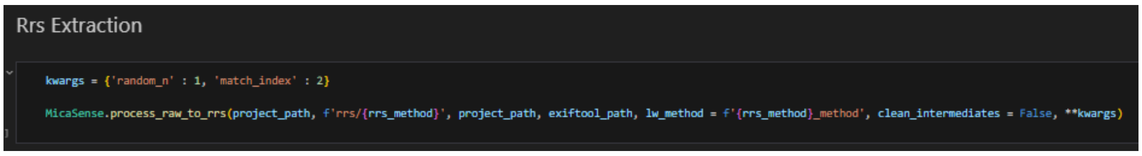
```
project_path = r'D:\DATOS1\Procesamientos\Zaragoza\Chiprana\F20230614_02\mosaicado'
rrs_method = 'hedley'
exiftool_path = r'C:\Users\Usuario\Documents\repos\seadronelib\dependencies\exiftool.exe'

tur_methods = {
    TsmMethods.nechad.__name__ : None,
}

chl_methods = {
    ChlorophyllMethods.hu.__name__ : None,
    ChlorophyllMethods.ocx.__name__ : None,
    ChlorophyllMethods.hu_ocx.__name__ : None,
    ChlorophyllMethods.gitelson.__name__ : None,
}
```

- Stacks and Thumbnails Extraction: This extracts the thumbnails (pseudo-RGB) along with the metadata file. No modifications are required in this line.


Stacs and Thumbnails Extraction

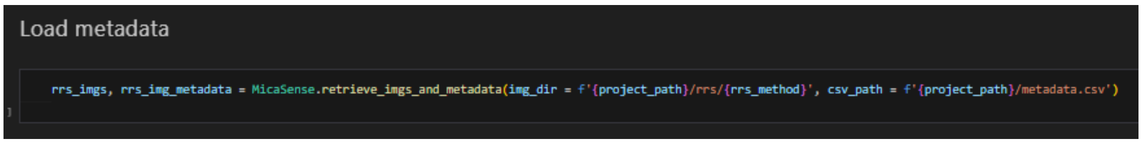
```
kwargs = {'match_index' : 2}

MicaSense.process_micasense_images(project_path, exiftool_path, os.path.join(project_path, 'align_img'), False, **kwargs)
```

- Rrs Extraction: This extracts the rrs values for each capture. No modifications are required in this line.


Rrs Extraction

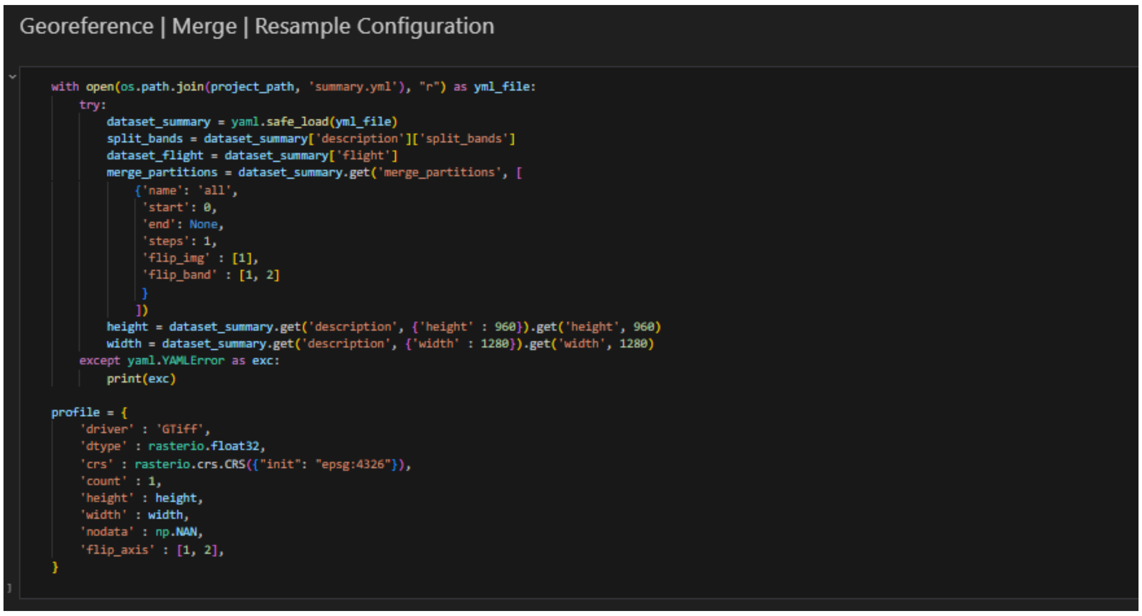
```
kwargs = {'random_n' : 1, 'match_index' : 2}

MicaSense.process_raw_to_rrs(project_path, f'rrs/{rrs_method}', project_path, exiftool_path, lw_method = f'{rrs_method}_method', clean_intermediates = False, **kwargs)
```

- Load metadata: This line opens the metadata file and loads all the images with the extracted rrs values. No modifications are required in this line.


Load metadata

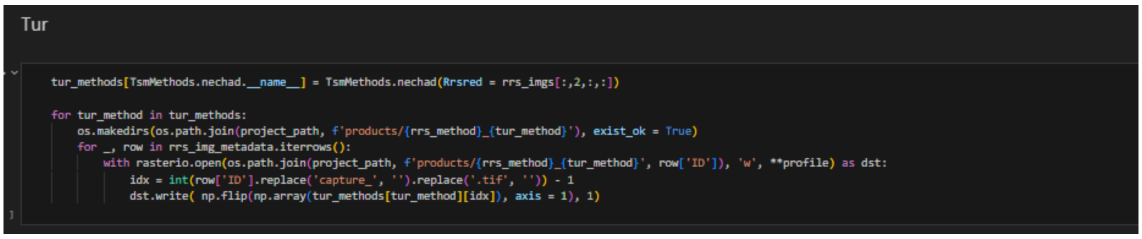
```
rrs_imgs, rrs_img_metadata = MicaSense.retrieve_imgs_and_metadata(img_dir = f'{project_path}/rrs/{rrs_method}', csv_path = f'{project_path}/metadata.csv')
```

- Georeference | Merge | Resample Configuration: This section opens the summary.yml file to use information for mosaic and georeference, as well as to save the product information in a pre-georeferenced tif file. No modifications are required in this line.


Georeference | Merge | Resample Configuration

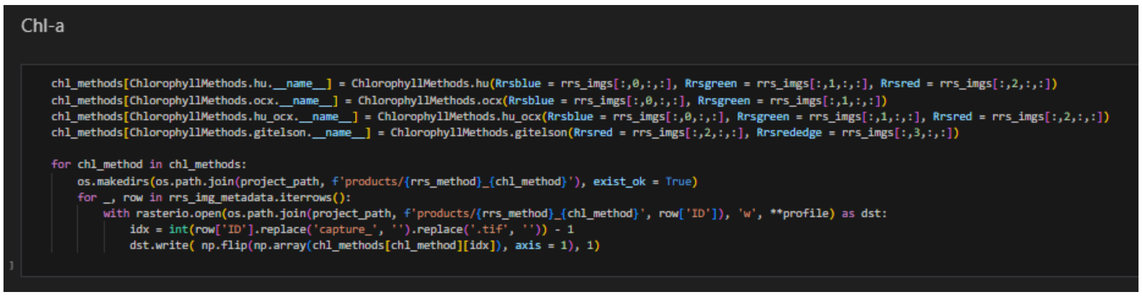
```python
with open(os.path.join(project_path, 'summary.yml'), "r") as yml_file:
    try:
        dataset_summary = yaml.safe_load(yml_file)
        split_bands = dataset_summary['description']['split_bands']
        dataset_flight = dataset_summary['flight']
        merge_partitions = dataset_summary.get('merge_partitions', [
            {'name': 'all',
             'start': 0,
             'end': None,
             'steps': 1,
             'flip_img' : [1],
             'flip_band' : [1, 2]
             }
            ])
        height = dataset_summary.get('description', {'height' : 960}).get('height', 960)
        width = dataset_summary.get('description', {'width' : 1280}).get('width', 1280)
    except yaml.YAMLError as exc:
        print(exc)

profile = {
    'driver' : 'GTiff',
    'dtype' : rasterio.float32,
    'crs' : rasterio.crs.CRS({"init": "epsg:4326"}),
    'count' : 1,
    'height' : height,
    'width' : width,
    'nodata' : np.NAN,
    'flip_axis' : [1, 2],
}
```

- Products | Tur: This calculates turbidity and saves it in a non-georeferenced tif file. No modifications are required in this line.


Tur

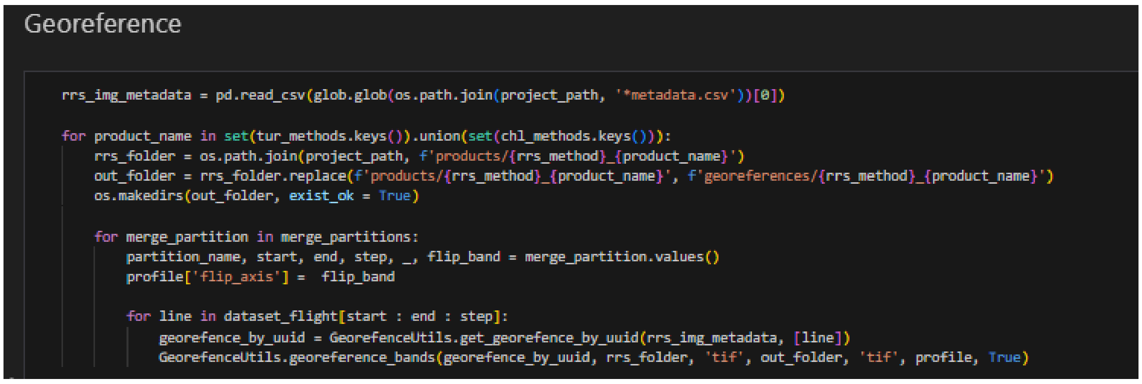
```python
tur_methods[TsmMethods.nechad.__name__] = TsmMethods.nechad(Rrsred = rrs_imgs[:,2,:,:])

for tur_method in tur_methods:
    os.makedirs(os.path.join(project_path, f'products/{rrs_method}_{tur_method}'), exist_ok = True)
    for _, row in rrs_img_metadata.iterrows():
        with rasterio.open(os.path.join(project_path, f'products/{rrs_method}_{tur_method}', row['ID']), 'w', **profile) as dst:
            idx = int(row['ID'].replace('capture_', '').replace('.tif', '')) - 1
            dst.write( np.flip(np.array(tur_methods[tur_method][idx]), axis = 1), 1)
```

- Products | Chl-a: This calculates Chlorophyll-a and saves it in a non-georeferenced tif file. No modifications are required in this line.


Chl-a

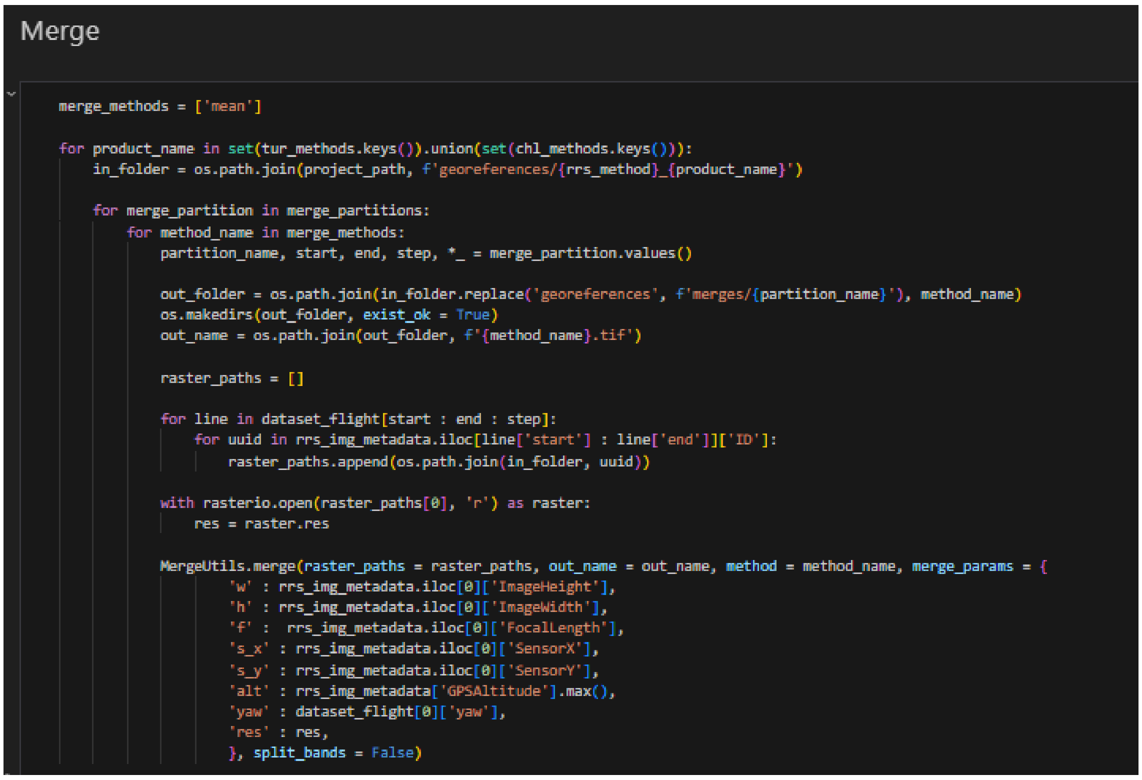
```python
chl_methods[ChlorophyllMethods.hu.__name__] = ChlorophyllMethods.hu(Rrsblue = rrs_imgs[:,0,:,:], Rrsgreen = rrs_imgs[:,1,:,:], Rrsred = rrs_imgs[:,2,:,:])
chl_methods[ChlorophyllMethods.ocx.__name__] = ChlorophyllMethods.ocx(Rrsblue = rrs_imgs[:,0,:,:], Rrsgreen = rrs_imgs[:,1,:,:])
chl_methods[ChlorophyllMethods.hu_ocx.__name__] = ChlorophyllMethods.hu_ocx(Rrsblue = rrs_imgs[:,0,:,:], Rrsgreen = rrs_imgs[:,1,:,:], Rrsred = rrs_imgs[:,2,:,:])
chl_methods[ChlorophyllMethods.gitelson.__name__] = ChlorophyllMethods.gitelson(Rrsred = rrs_imgs[:,2,:,:], Rrsrededge = rrs_imgs[:,3,:,:])

for chl_method in chl_methods:
    os.makedirs(os.path.join(project_path, f'products/{rrs_method}_{chl_method}'), exist_ok = True)
    for _, row in rrs_img_metadata.iterrows():
        with rasterio.open(os.path.join(project_path, f'products/{rrs_method}_{chl_method}', row['ID']), 'w', **profile) as dst:
            idx = int(row['ID'].replace('capture_', '').replace('.tif', '')) - 1
            dst.write( np.flip(np.array(chl_methods[chl_method][idx]), axis = 1), 1)
```

- Products | Georeference: This applies georeferencing to all the products. No modifications are required in this line.


Georeference

```python
rrs_img_metadata = pd.read_csv(glob.glob(os.path.join(project_path, '*metadata.csv'))[0])

for product_name in set(tur_methods.keys()).union(set(chl_methods.keys())):
    rrs_folder = os.path.join(project_path, f'products/{rrs_method}_{product_name}')
    out_folder = rrs_folder.replace(f'products/{rrs_method}_{product_name}', f'georeferences/{rrs_method}_{product_name}')
    os.makedirs(out_folder, exist_ok = True)

    for merge_partition in merge_partitions:
        partition_name, start, end, step, _, flip_band = merge_partition.values()
        profile['flip_axis'] =  flip_band

        for line in dataset_flight[start : end : step]:
            georefence_by_uuid = GeorefenceUtils.get_georefence_by_uuid(rrs_img_metadata, [line])
            GeorefenceUtils.georeference_bands(georefence_by_uuid, rrs_folder, 'tif', out_folder, 'tif', profile, True)
```

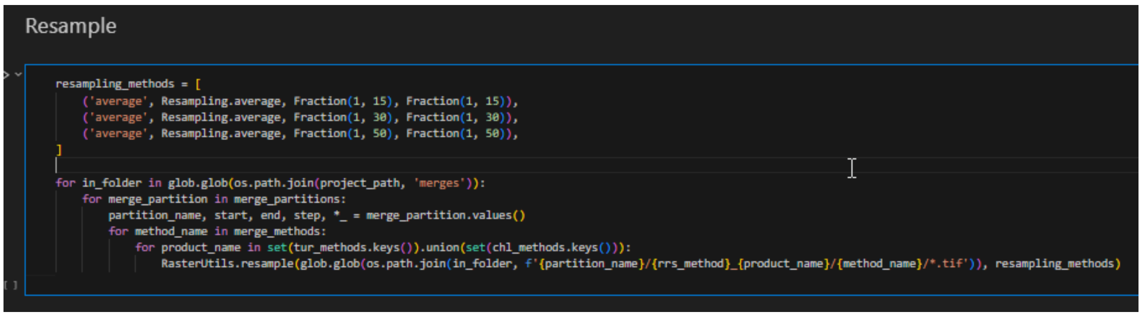
- Products | Merge: This generates the orthomosaic and exports it to its corresponding folder. It's important to note that if we set "split = true" in the summary.yml file, it will save an orthomosaic for each band according to the specified method. No modifications are required in this line, although there are four different merge methods indicated in "merge_methods". These methods are: "mean", which calculates the mean value of all pixels at the same geolocation without considering NaN values; "first", which selects the first data stored in the matrix; "max", which selects the maximum values for each pixel; and "min", which selects the minimum values for each pixel.



Merge

```python
merge_methods = ['mean']

for product_name in set(tur_methods.keys()).union(set(chl_methods.keys())):
    in_folder = os.path.join(project_path, f'georeferences/{rrs_method}_{product_name}')

    for merge_partition in merge_partitions:
        for method_name in merge_methods:
            partition_name, start, end, step, *_ = merge_partition.values()

            out_folder = os.path.join(in_folder.replace('georeferences', f'merges/{partition_name}'), method_name)
            os.makedirs(out_folder, exist_ok = True)
            out_name = os.path.join(out_folder, f'{method_name}.tif')

            raster_paths = []

            for line in dataset_flight[start : end : step]:
                for uuid in rrs_img_metadata.iloc[line['start'] : line['end']]['ID']:
                    raster_paths.append(os.path.join(in_folder, uuid))

            with rasterio.open(raster_paths[0], 'r') as raster:
                res = raster.res

            MergeUtils.merge(raster_paths = raster_paths, out_name = out_name, method = method_name, merge_params = {
                    'w' : rrs_img_metadata.iloc[0]['ImageHeight'],
                    'h' : rrs_img_metadata.iloc[0]['ImageWidth'],
                    'f' :  rrs_img_metadata.iloc[0]['FocalLength'],
                    's_x' : rrs_img_metadata.iloc[0]['SensorX'],
                    's_y' : rrs_img_metadata.iloc[0]['SensorY'],
                    'alt' : rrs_img_metadata['GPSAltitude'].max(),
                    'yaw' : dataset_flight[0]['yaw'],
                    'res' : res,
                }, split_bands = False)
```

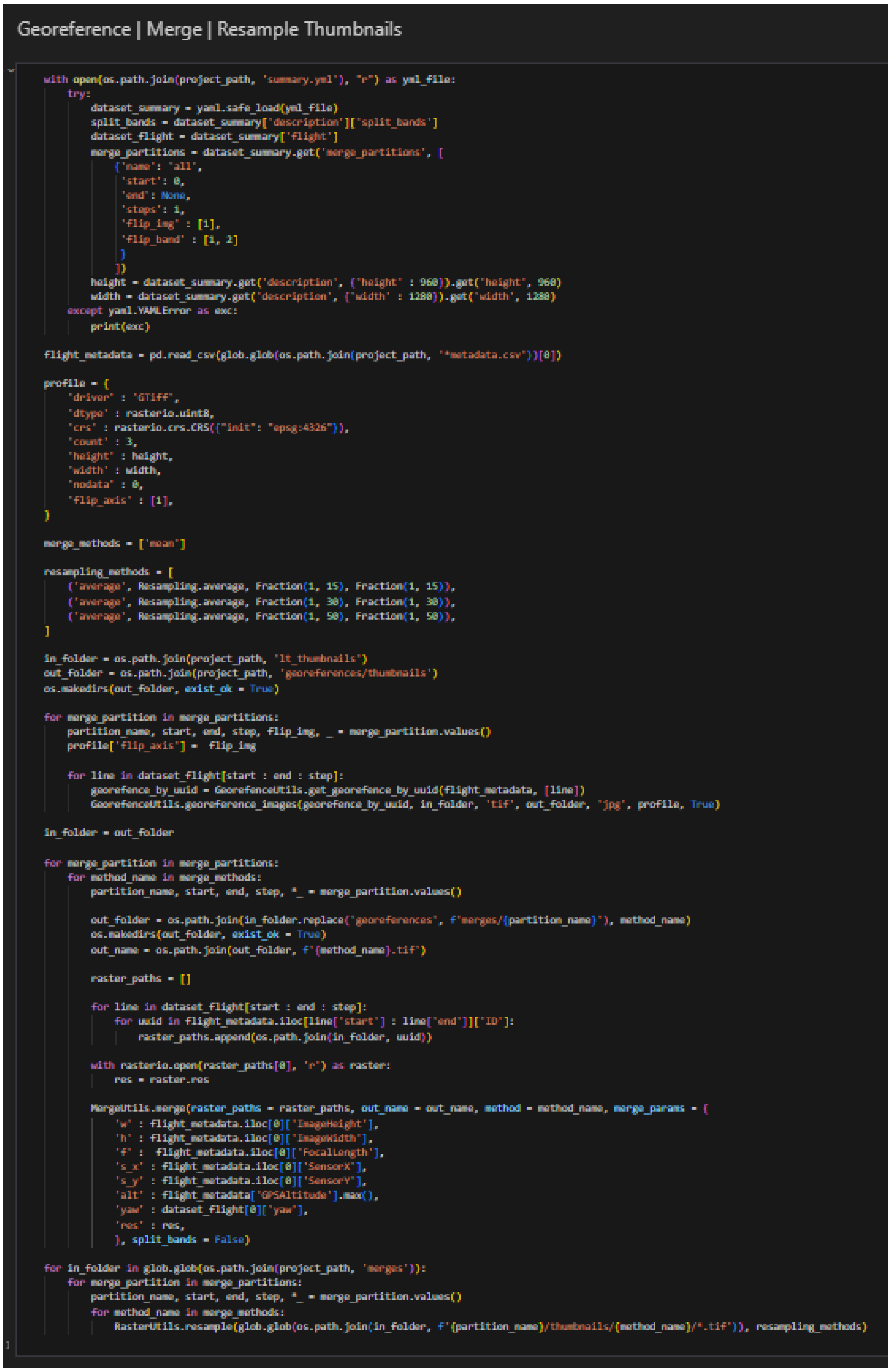
- Products | Resample: This applies downsampling to the final orthomosaic product. No modifications are required in this line, except for the scaling factor in "resampling_methods" if you want to test other values.



Resample

```python
resampling_methods = [
    ('average', Resampling.average, Fraction(1, 15), Fraction(1, 15)),
    ('average', Resampling.average, Fraction(1, 30), Fraction(1, 30)),
    ('average', Resampling.average, Fraction(1, 50), Fraction(1, 50)),
]

for in_folder in glob.glob(os.path.join(project_path, 'merges')):
    for merge_partition in merge_partitions:
        partition_name, start, end, step, *_ = merge_partition.values()
        for method_name in merge_methods:
            for product_name in set(tur_methods.keys()).union(set(chl_methods.keys())):
                RasterUtils.resample(glob.glob(os.path.join(in_folder, f'{partition_name}/{rrs_method}_{product_name}/{method_name}/*.tif')), resampling_methods)
```

- Georeference | Merge | Resample Thumbnails: This applies the steps described above for products but for the thumbnails (pseudo-RGB).


Georeference | Merge | Resample Thumbnails

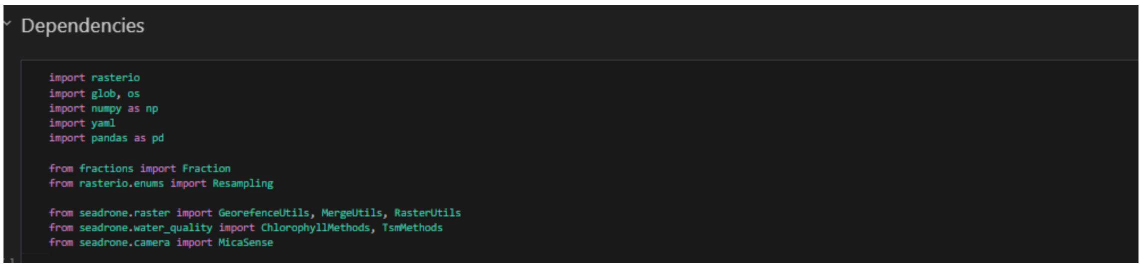
```python
with open(os.path.join(project_path, 'summary.yml'), "r") as yml_file:
    try:
        dataset_summary = yaml.safe_load(yml_file)
        split_bands = dataset_summary['description']['split_bands']
        dataset_flight = dataset_summary['flight']
        merge_partitions = dataset_summary.get('merge_partitions', [
            {'name': 'all',
             'start': 0,
             'end': None,
             'steps': 1,
             'flip_img' : [1],
             'flip_band' : [1, 2]
             }
            ])
        height = dataset_summary.get('description', {'height' : 960}).get('height', 960)
        width = dataset_summary.get('description', {'width' : 1280}).get('width', 1280)
    except yaml.YAMLError as exc:
        print(exc)

flight_metadata = pd.read_csv(glob.glob(os.path.join(project_path, '*metadata.csv'))[0])

profile = {
    'driver' : 'GTiff',
    'dtype' : rasterio.uint8,
    'crs' : rasterio.crs.CRS({"init": "epsg:4326"}),
    'count' : 3,
    'height' : height,
    'width' : width,
    'nodata' : 0,
    'flip_axis' : [1],
}

merge_methods = ['mean']

resampling_methods = [
    ('average', Resampling.average, Fraction(1, 15), Fraction(1, 15)),
    ('average', Resampling.average, Fraction(1, 30), Fraction(1, 30)),
    ('average', Resampling.average, Fraction(1, 50), Fraction(1, 50)),
]

in_folder = os.path.join(project_path, 'lt_thumbnails')
out_folder = os.path.join(project_path, 'georeferences/thumbnails')
os.makedirs(out_folder, exist_ok = True)

for merge_partition in merge_partitions:
    partition_name, start, end, step, flip_img, _ = merge_partition.values()
    profile['flip_axis'] =  flip_img

    for line in dataset_flight[start : end : step]:
        georefence_by_uuid = GeorefenceUtils.get_georefence_by_uuid(flight_metadata, [line])
        GeorefenceUtils.georeference_images(georefence_by_uuid, in_folder, 'tif', out_folder, 'jpg', profile, True)

in_folder = out_folder

for merge_partition in merge_partitions:
    for method_name in merge_methods:
        partition_name, start, end, step, *_ = merge_partition.values()

        out_folder = os.path.join(in_folder.replace('georeferences', f'merges/{partition_name}'), method_name)
        os.makedirs(out_folder, exist_ok = True)
        out_name = os.path.join(out_folder, f'{method_name}.tif')

        raster_paths = []

        for line in dataset_flight[start : end : step]:
            for uuid in flight_metadata.iloc[line['start'] : line['end']]['ID']:
                raster_paths.append(os.path.join(in_folder, uuid))

        with rasterio.open(raster_paths[0], 'r') as raster:
            res = raster.res

        MergeUtils.merge(raster_paths = raster_paths, out_name = out_name, method = method_name, merge_params = {
            'w' : flight_metadata.iloc[0]['ImageHeight'],
            'h' : flight_metadata.iloc[0]['ImageWidth'],
            'f' :  flight_metadata.iloc[0]['FocalLength'],
            's_x' : flight_metadata.iloc[0]['SensorX'],
            's_y' : flight_metadata.iloc[0]['SensorY'],
            'alt' : flight_metadata['GPSAltitude'].max(),
            'yaw' : dataset_flight[0]['yaw'],
            'res' : res,
            }, split_bands = False)

for in_folder in glob.glob(os.path.join(project_path, 'merges')):
    for merge_partition in merge_partitions:
        partition_name, start, end, step, *_ = merge_partition.values()
        for method_name in merge_methods:
            RasterUtils.resample(glob.glob(os.path.join(in_folder, f'{partition_name}/thumbnails/{method_name}/*.tif')), resampling_methods)
```

Note: The instructions provided assume familiarity with the specific code structure and variables in the mentioned file.

3. START OF "BATCH_PROCESSING_JUPYTER" PROCESSING

Another option, once you are familiar with "**mono_processing_jupyter**" is to work in batch using "**batch_processing_jupyter**". In this case, open the "direct_georeference_products" file, and below is a line-by-line description:

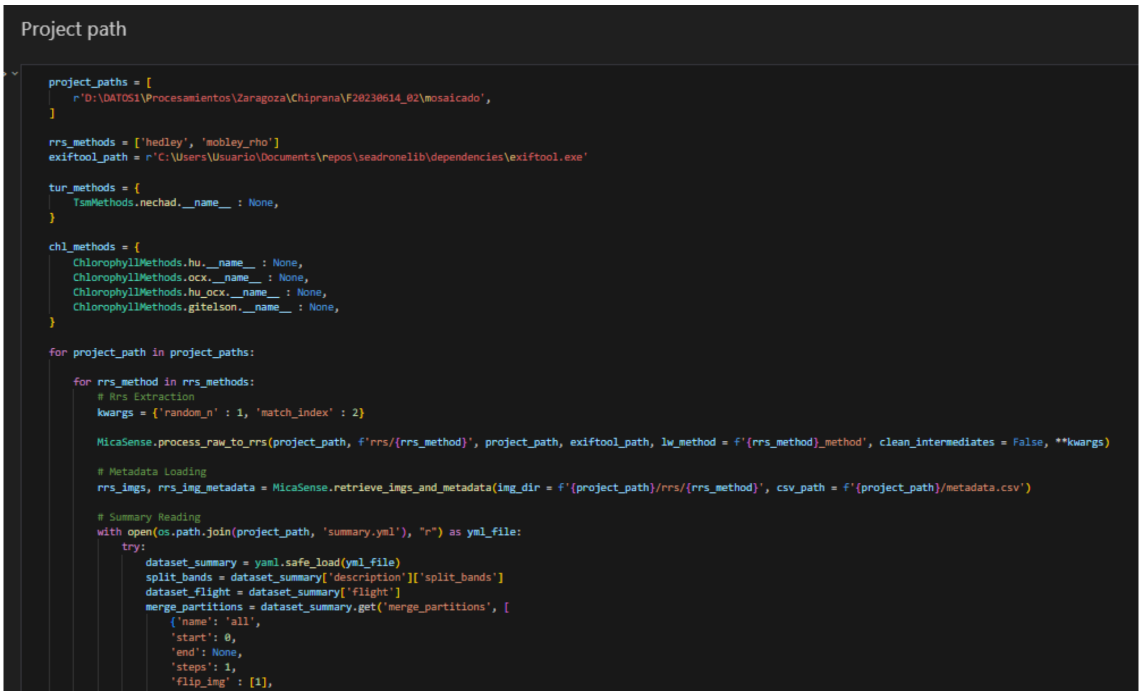
- Dependencies: This includes all the necessary libraries to run the code.



Dependencies

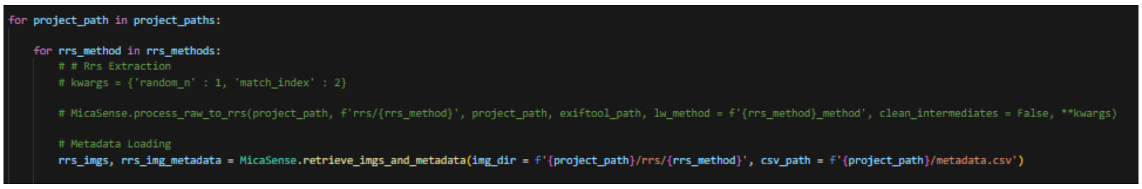
```
import rasterio
import glob, os
import numpy as np
import yaml
import pandas as pd

from fractions import Fraction
from rasterio.enums import Resampling

from seadrone.raster import GeorefenceUtils, MergeUtils, RasterUtils
from seadrone.water_quality import ChlorophyllMethods, TsmMethods
from seadrone.camera import MicaSense
```

- Project path: Here, we have the entire processing from data import to obtaining georeferenced products. You would need to modify the "project_paths" with the full path of the dataset, the "rrs_method" which determines how we derive rrs ("hedley", "mobley_rho", or "blackpixel"). Unlike "mono_processing_jupyter," this option allows for simultaneous processing of multiple methods. Also, modify the "exiftool_path" with the full path of the exiftool.exe file. You don't need to modify anything else unless you want to exclude specific product exports, in which case you can comment out those lines using the "#" symbol in Python. Be careful not to comment out a step that is a prerequisite for a subsequent step, as it would cause an error in the code.



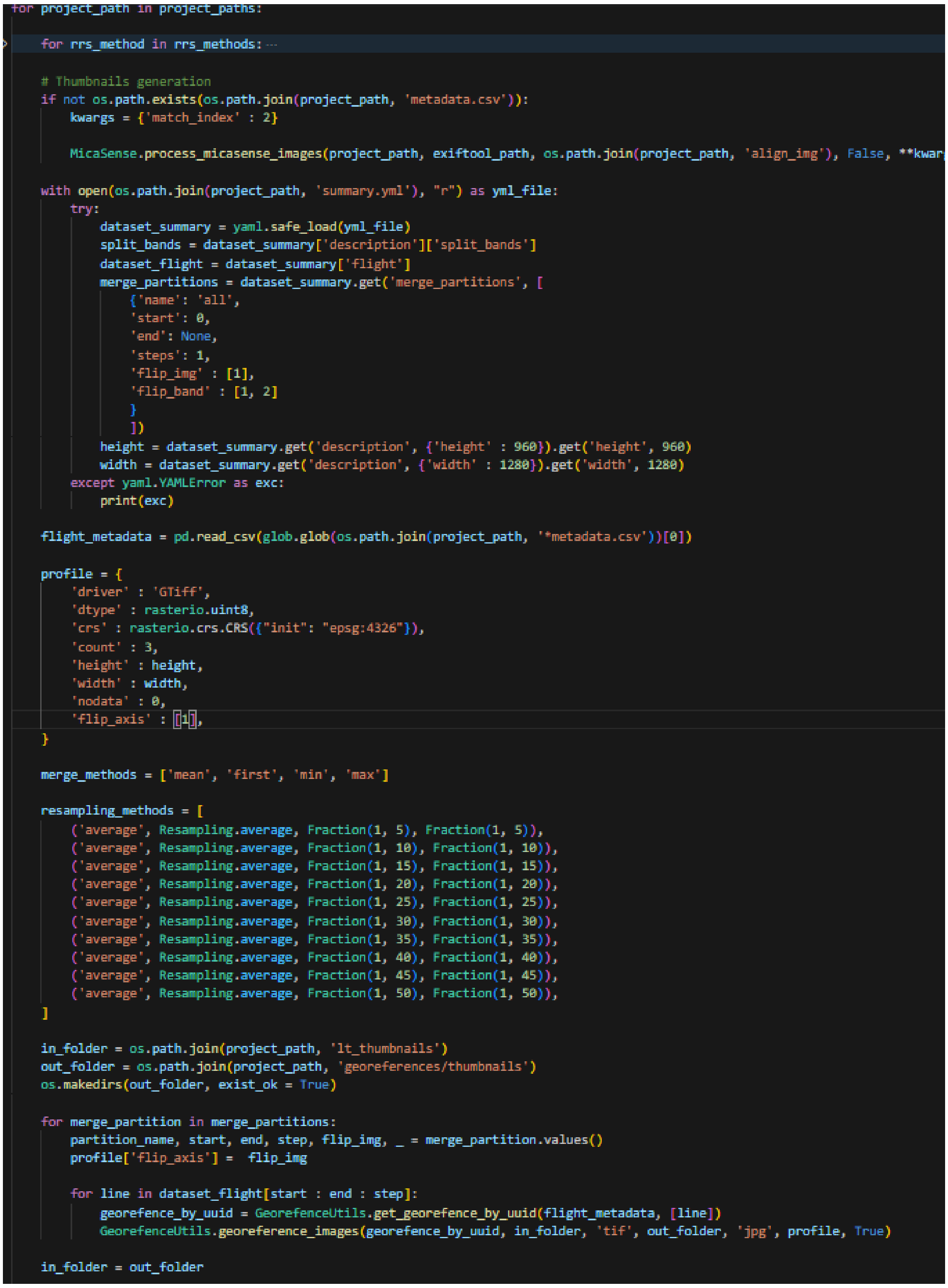
Project path

```
project_paths = [
    r'D:\DATOS1\Procesamientos\Zaragoza\Chiprana\F20230614_02\mosaicado',
]

rrs_methods = ['hedley', 'mobley_rho']
exiftool_path = r'C:\Users\Usuario\Documents\repos\seadronelib\dependencies\exiftool.exe'

tur_methods = {
    TsmMethods.nechad.__name__ : None,
}

chl_methods = {
    ChlorophyllMethods.hu.__name__ : None,
    ChlorophyllMethods.ocx.__name__ : None,
    ChlorophyllMethods.hu_ocx.__name__ : None,
    ChlorophyllMethods.gitelson.__name__ : None,
}

for project_path in project_paths:

    for rrs_method in rrs_methods:
        # Rrs Extraction
        kwargs = {'random_n' : 1, 'match_index' : 2}

        MicaSense.process_raw_to_rrs(project_path, f'rrs/{rrs_method}', project_path, exiftool_path, lw_method = f'{rrs_method}_method', clean_intermediates = False, **kwargs)

        # Metadata Loading
        rrs_imgs, rrs_img_metadata = MicaSense.retrieve_imgs_and_metadata(img_dir = f'{project_path}/rrs/{rrs_method}', csv_path = f'{project_path}/metadata.csv')

        # Summary Reading
        with open(os.path.join(project_path, 'summary.yml'), "r") as yml_file:
            try:
                dataset_summary = yaml.safe_load(yml_file)
                split_bands = dataset_summary['description']['split_bands']
                dataset_flight = dataset_summary['flight']
                merge_partitions = dataset_summary.get('merge_partitions', [
                    {'name': 'all',
                    'start': 0,
                    'end': None,
                    'steps': 1,
                    'flip_img' : [1],
```



```
for project_path in project_paths:

    for rrs_method in rrs_methods:
        # # Rrs Extraction
        # kwargs = {'random_n' : 1, 'match_index' : 2}

        # MicaSense.process_raw_to_rrs(project_path, f'rrs/{rrs_method}', project_path, exiftool_path, lw_method = f'{rrs_method}_method', clean_intermediates = False, **kwargs)

        # Metadata Loading
        rrs_imgs, rrs_img_metadata = MicaSense.retrieve_imgs_and_metadata(img_dir = f'{project_path}/rrs/{rrs_method}', csv_path = f'{project_path}/metadata.csv')
```

Note: It is important to have a good understanding of the code structure and variables to effectively modify and work with the "batch_processing_jupyter" option.



```
for project_path in project_paths:

    for rrs_method in rrs_methods:···

    # Thumbnails generation
    if not os.path.exists(os.path.join(project_path, 'metadata.csv')):
        kwargs = {'match_index' : 2}

        MicaSense.process_micasense_images(project_path, exiftool_path, os.path.join(project_path, 'align_img'), False, **kwar

    with open(os.path.join(project_path, 'summary.yml'), "r") as yml_file:
        try:
            dataset_summary = yaml.safe_load(yml_file)
            split_bands = dataset_summary['description']['split_bands']
            dataset_flight = dataset_summary['flight']
            merge_partitions = dataset_summary.get('merge_partitions', [
                {'name': 'all',
                'start': 0,
                'end': None,
                'steps': 1,
                'flip_img' : [1],
                'flip_band' : [1, 2]
                }
                ])
            height = dataset_summary.get('description', {'height' : 960}).get('height', 960)
            width = dataset_summary.get('description', {'width' : 1280}).get('width', 1280)
        except yaml.YAMLError as exc:
            print(exc)

    flight_metadata = pd.read_csv(glob.glob(os.path.join(project_path, '*metadata.csv'))[0])

    profile = {
        'driver' : 'GTiff',
        'dtype' : rasterio.uint8,
        'crs' : rasterio.crs.CRS({"init": "epsg:4326"}),
        'count' : 3,
        'height' : height,
        'width' : width,
        'nodata' : 0,
        'flip_axis' : [1],
    }

    merge_methods = ['mean', 'first', 'min', 'max']

    resampling_methods = [
        ('average', Resampling.average, Fraction(1, 5), Fraction(1, 5)),
        ('average', Resampling.average, Fraction(1, 10), Fraction(1, 10)),
        ('average', Resampling.average, Fraction(1, 15), Fraction(1, 15)),
        ('average', Resampling.average, Fraction(1, 20), Fraction(1, 20)),
        ('average', Resampling.average, Fraction(1, 25), Fraction(1, 25)),
        ('average', Resampling.average, Fraction(1, 30), Fraction(1, 30)),
        ('average', Resampling.average, Fraction(1, 35), Fraction(1, 35)),
        ('average', Resampling.average, Fraction(1, 40), Fraction(1, 40)),
        ('average', Resampling.average, Fraction(1, 45), Fraction(1, 45)),
        ('average', Resampling.average, Fraction(1, 50), Fraction(1, 50)),
    ]

    in_folder = os.path.join(project_path, 'lt_thumbnails')
    out_folder = os.path.join(project_path, 'georeferences/thumbnails')
    os.makedirs(out_folder, exist_ok = True)

    for merge_partition in merge_partitions:
        partition_name, start, end, step, flip_img, _ = merge_partition.values()
        profile['flip_axis'] =  flip_img

        for line in dataset_flight[start : end : step]:
            georefence_by_uuid = GeorefenceUtils.get_georefence_by_uuid(flight_metadata, [line])
            GeorefenceUtils.georeference_images(georefence_by_uuid, in_folder, 'tif', out_folder, 'jpg', profile, True)

    in_folder = out_folder
```