METEOROLOGICAL POST PROCESSING FOR GEOSTATIONARY 
AND POLAR IMAGERY

M. Raspaud, A. Dybbroe, A. Geidne

SMHI,
Folkborgsvägen 1
SE–60176, Norrköping
satsa@smhi.se

K.R. Larsen, E.S. Nielsen, L.Ø. Rasmussen

DMI,
Lyngbyvej 100
DK–2100 København Ø
satsys@smhi.se

Abstract

We present three new software packages allowing easy meteorological post-processing for geostationary and polar imagery. They have been developed and are used at DMI and SMHI. They are written in Python and are distributed as open-source software. The capabilities of these packages include the reading of several satellite data formats, resampling and re-projection of data, combination of satellite channels to produce RGB composites, and saving these composites in several formats. A guided tour is provided to show parts of the possibilities of the software.

1 INTRODUCTION

In the 1980s and 1990s before the birth of the acronyms MSG, EARS and SAF, many satellite image processing systems in operation at national meteorological institutes around Europe circled around the simple processing of images (perhaps not even rectified or re-projected) from the three channel Meteosat imager and the five channel NOAA AVHRR. Typically the local reception and pre-processing of raw data was handled by commercial software (e.g. provided by VCS or Kongsberg Spacetec) and maybe RGB colour composite images were generated by dedicated (in-house developed or commercially tailored) software which also in some cases (e.g. the SMHI Prosat system) could handle the processing of locally tuned cloud classification algorithms.

With the introduction of the SAF’s, delivering software for local processing or providing products disseminated in near-real time, and of course especially with the MSG, EPS, and EOS satellites, the wealth and variability of meteorological and oceanographic real time data has grown tremendously. This fairly recent reality has already pushed the old processing environments to their limits, even despite a modular design from the beginning. The project presented here aims to overcome the problems encountered with the past satellite processing systems and will cope with the new and emerging demands of a steady growth of input data and demand for new products. We introduce a simple and clean framework which will allow the rapid development of new products and a flexible and modular layout which will make the operational implementation of new products straightforward. Data transparency and open distribution systems are also important components. Here we will focus on three separate modules: the input data layer (ingestion of raw level 0 or level 1 data), the resampling and projection layer, and finally the image generation layer.

The present versions of the software can ingest MSG SEVIRI HRIT data, GOES, MTSAT, METEOSAT LRIT data, NOAA and EPS AVHRR level 1b (AAPP1 format, using AHAMAP2) and Terra & Aqua MODIS level 1b3, resample and re-project the data to user defined geographical areas, and make various RGB composite images (including the standard ones proposed in the MSG interpretation guide (Kerkmann et al., 2006)) and can store them in either GEOTIFF (digestible in the NINJO and DIANA visualisation systems) or PNG. In addition the systems can handle the outputs from the NWCSAF MSG4 and NWCSAF/PPS5 software packages.
The software is open-source and written in Python with C-extensions where needed, and it makes extensive use of Numpy masked arrays. Other third party free and open-source libraries include the Geospatial Data Abstraction Language GDAL\textsuperscript{6}, Python Image Library, and USGS Proj.\textsuperscript{4}\textsuperscript{7}. The software is version controlled under Git, documented with Sphinx and checked with Pylint and Unittest.

This project is the fruit of a collaboration between DMI and SMHI. The software is used in production at SMHI for all the geostationary imagery and parts of the polar imagery, and it is in test production at DMI for geostationary imagery and will be in full production in the near future.

This article is organised as follows: we present first an overall view of the software structure, with descriptions of the different packages of the software chain. Then we give a guided tour of the possibilities of the software.

2 THE SOFTWARE PACKAGES

The software contains three parts: mipp (Rasmussen, 2010) as an input layer for HRIT and LRIT data, pyresample (Nielsen, 2010) for data resampling and re-projection, and mpop (Raspaud, 2010) to glue them all and produce the actual images. As such, in a typical production chain, mpop would be used to handle the data and build images, while it would call mipp for loading the data and pyresample to fit it into given areas.

Even though we present them as a whole, mipp, pyresample, and mpop can be used independently.

2.1 mipp

The Meteorological Ingest-Processing Package (mipp) is a Python package for reading level 1.5 satellite data and convert it to a common format understood by mpop.

Configuration files, defining satellites, its instruments and where data is located, and environment variables are shared with mpop.

For a given satellite, time stamp, channel and slice definition it will return a tuple containing a dictionary of meta-data and a Numpy array containing the image data.

It will:

• decompress XRIT files (if Eumetsat’s xRITDecompress is available).
• decode/strip-off (according to [\textsuperscript{1}lri, 1999], [\textsuperscript{2}mtp, 2000], [\textsuperscript{3}msg, 2006], [\textsuperscript{4}msg, 2007]) XRIT headers and collect meta-data.
• catenate image data into a numpy-array.
  – if needed, it will convert 10 bit data to 16 bit.
  – if a region is defined (by a slice or (center, size)) it will only read what is needed.

Source code and tarballs for mipp can be downloaded from http://github.com/loerum/mipp.

2.2 Pyresample

Pyresample is a Python package for resampling (re-projection) of data from earth observing satellites. The package handles both resampling of gridded data (e.g. geostationary data) and geolocated swath data (polar orbiting satellites).

Pyresample uses a kd-tree approach for finding nearest neighbours when resampling a data set. Types of resampling include nearest neighbour resampling, gaussian weighting of pixel values and weighting with user defined radial functions. Parallel resampling can be performed as an option on a specified number of processor cores.

Pyresample is designed to enable users to resample data with very few steps while still offering advanced users the flexibility of more closely controlling the procedure. For example, OOP (Object
Oriented Programming) is used to define a hierarchy of geometry definitions to describe the geometry type of a dataset (e.g. grid or swath). This allows for equal usage of the Pyresample API regardless of whether a swath is being resampled to a grid definition, a grid is remapped to a swath or any other combination of geometry mapping.

The basic data structure used in Pyresample is Numpy arrays including support for masked array data. This allows for integration with other packages from the numerical Python community.

Source code and tarballs can be downloaded from http://code.google.com/p/pyresample/.

2.3 mpop

The Meteorological Post-Processing package (mpop) is a python library for generating RGB products for meteorological remote sensing. As such it can be used to create RGB composites directly from satellite instrument channels, or take advantage of PGEs precomputed from external software (PPS or NWCSAF/MSG).

It is designed to be easily extensible to support any meteorological satellite by the use of plug-ins. In the base distribution, we provide support for Meteosat 7, 8, 9, MTSAT1R, MTSAT2, GOES 11, GOES 12, GOES 13 through the use of mipp (see section 2.1), and NOAA 15, 16, 17, 18, 19, and Metop-A through the use of AAPP and AHAMAP.

The heart of the mpop package is the generic scene module that hold the data and metadata of the satellite and its channels. Using a hierarchy of classes, we make the same functions (including composite generation) available for all the supported satellites. The secret for the easy maintenance of this scheme is the fact that channels in scenes for any instrument and satellite are retrieved by their centre wavelength. This way, it is exactly the same function call that will be used to run a given composite on AVHRR, SEVIRI, or any other instruments.

Another important part of mpop is the imageo module, which handles geographic image, i.e. images that contain geographical information. This module rests on Numpy and GDAL for processing and saving the data. It holds several functions for image processing such as stretching and gamma correction.

Mpop contains also the saturn module which is a collection of tools for batch processing of satellite data. It includes a task-list manager, a file-polling tool and a runner module that takes as input a task-list and some data and generates the necessary composites.

Source code and tarballs can be downloaded from http://github.com/mraspaud/mpop.

3 GUIDED TOUR TO THE METEOROLOGICAL POST-PROCESSING SOFTWARE

The software uses OOP extensively, to allow higher level meta-object handling.

What we show here is just a fraction of what the software can do.

In the following, we consider that the software packages have been installed properly on the host system and that the raw data is accessible from the prompt. For instructions on how to install the software, the reader is referred to the respective documentation pages.

3.1 First example

We show here how to load a scene and generate our first composite image.

```python
>>> from pp.satellites import get_satellite_class
>>> import datetime
>>> time_slot = datetime.datetime(2009, 10, 8, 14, 30)
>>> global_data = (get_satellite_class("meteosat", "09")
... (area="EuropeCanary", time_slot=time_slot))
```
In this example, we create a meteosat 09 scene object containing SEVIRI data specifying the name of the area we will work on and the time of the snapshot of interest. The time is defined as a `datetime` object. Specifying an area to work on restricts the loaded material to the minimal amount needed, enhancing performance of the software.

The next step is loading the data. This is done using `mipp`, which takes care of reading the HRIT data, and slicing the data so that we read just what is needed. Calibration is also done with mipp from counts to reflectance or brightness temperature.

Here we call the `load()` function with a list of the wavelengths of the channels we are interested in. Each retrieved channel is the closest in terms of central wavelength, provided that the required wavelength is within the bounds of the channel.

The wavelengths are given in micrometres and have to be input as a floating point number (i.e., don’t type ‘1’, but ‘1.0’). Using an integer number instead returns a channel based on resolution, while using a string retrieves a channels based on its name.

Once the channels are loaded, we generate an overview RGB composite image, and save it as a png image. Instead of `save()`, one could also use `show()` if the only purpose is to display the image on screen.

Available composites are listed in the `pp.satellites.visir` module in the mpop documentation (Raspaud, 2010).

### 3.2 Loading data for specific needs

In the last example, the composite generation worked because the channels needed for the overview composite (0.6, 0.8, 10.8 µm) were loaded. If we try to generate a day natural colour composite, which requires also the 1.6 µm channel, it will result in an error:

```python
global_data.load([0.6, 0.8, 10.8])
print(global_data)
```

```
'VIS006: (0.560,0.635,0.710)µm, shape (1200, 3000), resolution 3000m'
'VIS008: (0.740,0.810,0.880)µm, shape (1200, 3000), resolution 3000m'
'IR_016: (1.500,1.640,1.780)µm, resolution 3000m, not loaded'
...'
'HRV: (0.500,0.700,0.900)µm, resolution 1000m, not loaded'
```

```python
>>> img = global_data.overview()
>>> img.save('./myoverview.png')
```

Once the channels are loaded, we generate an overview RGB composite image, and save it as a png image. Instead of `save()`, one could also use `show()` if the only purpose is to display the image on screen.

Available composites are listed in the `pp.satellites.visir` module in the mpop documentation (Raspaud, 2010).

```python
>>> img = global_data.natural()
>>> img.save('./mynaturalcolors.png')
```

If you want to generate several composites and want to load all the needed channels at once, since prerequisites are python sets, you can do:
>>> global_data.load(global_data.overview.prerequisites |
...   global_data.natural.prerequisites)

... and add as many | global_data.mymethod.prerequisites as needed.

3.3 Retrieving channels

Retrieving channels is simple. From the centre wavelength (the closest channel):

```python
>>> print global_data[0.6]
'VIS06: (0.560, 0.635, 0.710)\mu m, shape (1200, 3000), resolution 3000m'
```

or from the channel name (exact match):

```python
>>> print global_data['VIS06']
'VIS06: (0.560, 0.635, 0.710)\mu m, shape (1200, 3000), resolution 3000m'
```

or from the resolution (first hit):

```python
>>> print global_data[3000]
'VIS06: (0.560, 0.635, 0.710)\mu m, shape (1200, 3000), resolution 3000m'
```

or more than one at the time (acts like `and`):

```python
>>> print global_data[3000, 0.8]
'VIS08: (0.740, 0.810, 0.880)\mu m, shape (1200, 3000), resolution 3000m'
```

The channel description consists of the following values:

- First the name of the channel,
- then a triplet gives the min-, centre-, and max-wavelength of the channel,
- follows the size of the loaded data, or nothing if the data is not loaded,
- the theoretical resolution of the channel at nadir is shown,
- “not loaded” if the channel is not loaded.

The data of the channel can be retrieved as an bumpy (masked) array using the `data` property:

```python
>>> print global_data[0.6].data
[[-- -- -- ..., -- -- --]
[-- -- -- ..., -- -- --]
[-- -- -- ..., -- -- --]
..., [7.37684259374 8.65549530999 6.58997938374 ..., 0.29507370375 0.1967158025 0.1967158025]
[7.18012679124 7.86863209999 6.19654777874 ..., 0.29507370375 0.29507370375 0.29507370375]
[5.80311617374 7.57355839624 6.88505308749 ..., 0.29507370375 0.29507370375 0.29507370375]]
```

3.4 PGES

If PGES are available from third party software (PPS or NWCSAF/MSG), they can be loaded in the same fashion exactly as regular channels:

```python
>>> global_data.load(['CTTH'])

... and they can be retrieved as simply as before:

```python
>>> print global_data['CTTH']
'CTTH: shape (1200, 3000), resolution 3000m'
```
3.5 Making a custom composite

Building custom composites makes use of the `imageo` module. For example, building an overview composite can be done manually with:

```python
>>> from imageo.geo_image import GeoImage
>>> img = geo_image.GeoImage((global_data[0.6].data,
...                           global_data[0.8].data,
...                           global_data[10.8].data),
...                           "EuropeCanary",
...                           time_slot,
...                           mode = "RGB")
>>> img.enhance(stretch="crude")
>>> img.enhance(gamma=1.7)
```

3.6 Projections

Until now, we have used the channels directly as provided by the satellite, that is in satellite projection. Generating composites thus produces views in satellite projection, i.e. as viewed by the satellite.

Most often however, we will want to project the data onto a predefined area (see subsection 3.7) so that only the area of interest is depicted in the RGB composites.

Here is how we do that:

```python
>>> local_data = global_data.project("eurol")
```

Now we have projected data onto the “eurol” area in the `local_data` variable and we can operate as before to generate RGB composites:

```python
>>> img = local_data.overview()
>>> img.save("./local_overview.tif")
```

Note that saving an image with the “tif” extension generates a GEOTIFF image.

On projected images, one can also add contour overlay with the `add_overlay()` method.

3.7 Area definitions

Instead of an area name, we can provide an area definition to the `project()` method. Creating an area definition is done with the following:

```python
>>> from pyresample import geometry
>>> area_id = 'ease_nh'
>>> name = 'Antarctic EASE grid'
>>> proj_id = 'ease_nh'
>>> x_size = 425
>>> y_size = 425
>>> area_extent = (-5326849.0625,-5326849.0625,5326849.0625,5326849.0625)
>>> proj_dict = {'a': '6371228.0', 'units': 'm', 'lon_0': '0',
...              'proj': 'laea', 'lat_0': '90'}
>>> area_def = geometry.AreaDefinition(area_id, name, proj_id, proj_dict, x_size,
...                                     y_size, area_extent)
>>> print area_def
Area ID: ease_nh
Name: Antarctic EASE grid
Projection ID: ease_nh
Projection: ('a': 6371228.0, 'units': 'm', 'lon_0': '0', 'proj': 'laea', ...
Number of columns: 425
Number of rows: 425
Area extent: (-5326849.0625, -5326849.0625, 5326849.0625, 5326849.0625)

Area definitions can also be saved in a file for later reuse. The parse_area_file() function can then be used to parse area definitions from the configuration file. Assuming the file /tmp/areas.cfg exists with the following content:

```plaintext
REGION: ease_nh {
    NAME: Arctic EASE grid
    PCS_ID: ease_nh
    PCS_DEF: proj=laea, lat_0=90, lon_0=0, a=6371228.0, units=m
    XSIZE: 425
    YSIZE: 425
    AREA_EXTENT: (-5326849.0625, -5326849.0625, 5326849.0625, 5326849.0625)
};
```

an area definition dict can be read using:

```python
>>> from pyresample import utils
>>> areas = utils.parse_area_file('/tmp/areas.cfg', 'ease_nh')
>>> print areas[0]
Area ID: ease_nh
Name: Arctic EASE grid
Projection ID: ease_nh
Projection: {'a': '6371228.0', 'units': 'm', 'lon_0': '0', 'proj': 'laea', ...}
Number of columns: 425
Number of rows: 425
Area extent: (-5326849.0625, -5326849.0625, 5326849.0625, 5326849.0625)
```

3.8 Configuration

Mipp and mpop are based on the same configuration files.

These configuration files are used to give the properties and information on a satellite and its instruments. It is then used by mipp to retrieve the satellite data, and can be used by mpop to generate satellite scene classes on the fly if needed. As an example, we show in figure 1 such a configuration file for the Meteosat 7 satellite. The same template is used for the other supported satellites (Meteosat 7, 8, 9, MTSAT-1R, MTSAT-2, GOES 11, GOES 12, GOES 13, NOAA 15, 16, 17, 18, 19, Metop-A, EOS Terra and Aqua). The configuration files have to be placed in the directory specified by the PPP_CONFIG_DIR environment variable.

The configuration file must hold a “satellite” section, the list of channels for the needed instruments (here “mviri-n” sections), and how to read the data in mipp (“mviri-level1”) and how to read it in mpop (“mviri-level2”). More detailed specifications of the config files can be found in the mpop documentation (Raspaud, 2010).

4 CONCLUSION

We have presented here three software packages for easy meteorological post processing of geostationary and polar satellite data, and showed some of their capabilities.

In the future, we intend to develop the production chain by coupling data files to a database that would then be queried if some product is to be build. Distribution and task managing modules are also on our to-do list.

5 REFERENCES

(1999) , LRIT/HRIT Global Specification, CGMS 03; Issue 2.6
[satellite]
satname = 'meteosat'
number = '07'
instruments = ('mviri',)
projection = 'geos(57.0)'

[mviri-level2]
format = 'mipp'

[mviri-level1]
format = 'xrit/MTP'
dir = 'space/remote/satellite/meteosat7'
filename = 'L-000-MTP___MET7________-%(channel)s_057E-%(segment)s-%Y%m%d%H%M-__'

[mviri-1]
name = '00_7'
frequency = (0.5, 0.7, 0.9)
resolution = 2248.49
size = (5000, 5000)

[mviri-2]
name = '06_4'
frequency = (5.7, 6.4, 7.1)
resolution = 4496.98
size = (2500, 2500)

[mviri-3]
name = '11_5'
frequency = (10.5, 11.5, 12.5)
resolution = 4496.98
size = (2500, 2500)

Figure 1: Example configuration file for the Meteosat 7 satellite

NOTES
1 http://research.metoffice.gov.uk/research/interproj/nwpsaf/aapp/index.html
2 http://www.nwcsaf.org/
3 Even more formats are supported with the use of third party software like AAPP, KAI, and EUGENE
4 http://www.nwcsaf.org/
5 http://www.nwcsaf.org/
6 http://www.gdal.org/
7 http://trac.osgeo.org/proj/
8 for now only CloudType and CTTH are supported
9 Although a working installation of PPS is required for this to work