pyplis Documentation

Release 1.4.4

Jonas Gliss

Apr 17, 2021

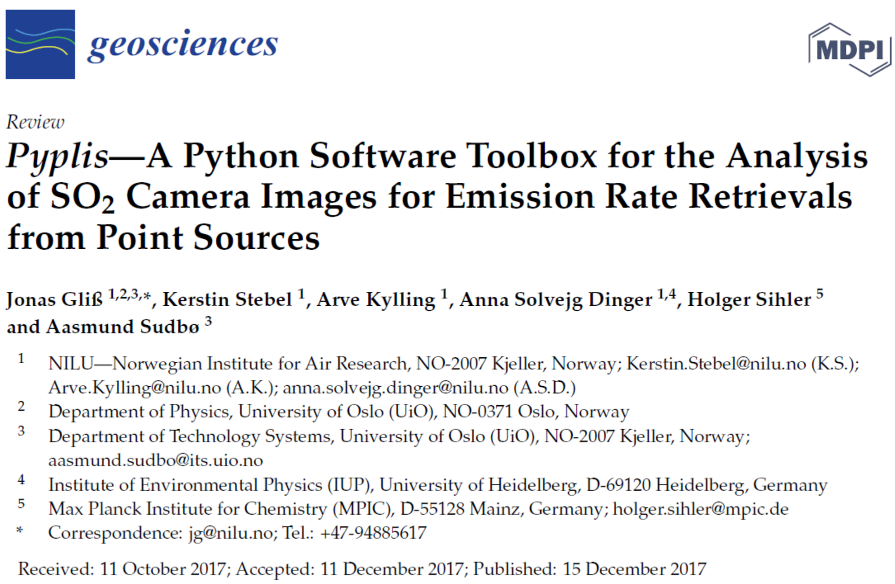
Contents

Pyplis

Python Plume Imaging Software

Official website of *Pyplis*, a Python software containing algorithms and analysis routines for UV SO2 camera data. Apart from the API documentation, this website includes additional relevant information about the software (e.g. installation details, access of example data) as well as useful practical information that helps getting started with *Pyplis*.

A paper introducing the software is published in the Journal *Geosciences* (MDPI). If you find *Pyplis* useful for your analysis, we highly appreciate if you acknowledge our work by citing the paper.



geosciences

MDPI

Review

Pyplis—A Python Software Toolbox for the Analysis of SO_2 Camera Images for Emission Rate Retrievals from Point Sources

Jonas Gliß [1,2,3,*], Kerstin Stebel [1], Arve Kylling [1], Anna Solvejg Dinger [1,4], Holger Sihler [5] and Aasmund Sudbø [3]

[1] NILU—Norwegian Institute for Air Research, NO-2007 Kjeller, Norway; Kerstin.Stebel@nilu.no (K.S.); Arve.Kylling@nilu.no (A.K.); anna.solvejg.dinger@nilu.no (A.S.D.)
[2] Department of Physics, University of Oslo (UiO), NO-0371 Oslo, Norway
[3] Department of Technology Systems, University of Oslo (UiO), NO-2007 Kjeller, Norway; aasmund.sudbo@its.uio.no
[4] Institute of Environmental Physics (IUP), University of Heidelberg, D-69120 Heidelberg, Germany
[5] Max Planck Institute for Chemistry (MPIC), D-55128 Mainz, Germany; holger.sihler@mpic.de
[*] Correspondence: jg@nilu.no; Tel.: +47-94885617

Received: 11 October 2017; Accepted: 11 December 2017; Published: 15 December 2017

Fig. 1: Screenshot of the *Pyplis* paper. Please acknowledge our work by citing the paper.

The software can be downloaded from the Github page or installed via PyPi using:

```
pip install pyplis
```

from the command line.

Click here to see a video showing an animation of Mt. Etna SO2 emissions recorded on 16/09/2015

Contents:

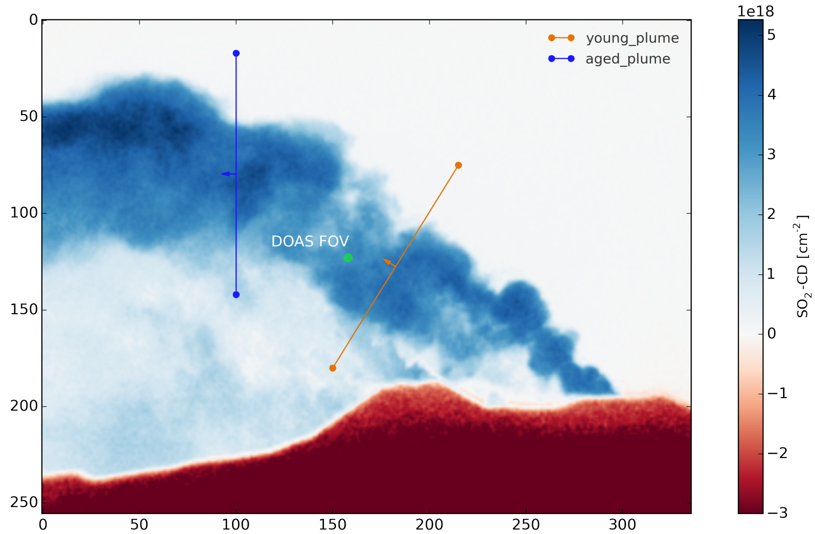


Fig. 2: Calibrated SO2 column density image showing 2 plume cross section lines and the position of the FOV of a co-located DOAS instrument.

CHAPTER 1

Introduction

Pyplis is a Python toolbox originally developed for the analysis of UV SO2 camera data. The software includes a comprehensive and flexible collection of algorithms for the analysis of atmospheric imaging data and is tested for all major operating systems and python 3 as well as python 2.7 (which is not recommended to use anymore).

Contact: Jonas Gliß (jonasgliss@gmail.com)

1.1 Help needed / Contributions more than welcome

Since I have to maintain pyplis during my free time and since I am not working in the field of volcano remote sensing anymore, I would be very grateful for contributions from users / developers closer to the application of UV SO2 cameras. If you are interested, feel free to get in touch with me (jonasgliss@gmail.com) or go ahead and send a PR via a fork, or by creating issues (best if you tag me with @jgliss in issues and PRs).

Cheers, Jonas

1.2 Code documentation and more

The code documentation of pyplis and more information is hosted on Read the Docs.

1.3 Main features

- Detailed analysis of the measurement geometry including automatic retrieval of distances to the emission plume and/or to topographic features in the camera images (at pixel-level).
- Several routines for the retrieval of plume background intensities (either from plume images directly or using an additional sky reference image).
- Automatic analysis of cell calibration data.

- Correction for cross-detector variations in the SO2 sensitivity arising from wavelength shifts in the filter transmission windows.
- DOAS calibration routine including two methods to identify the field of view of a DOAS instrument within the camera images.
- Plume velocity retrieval either using an optical flow analysis or using signal cross correlation.
- Histogram based post analysis of optical flow field for gas velocity analysis in low contrast image regions, where the optical flow fails to detect motion (cf. Gliss et al., 2018, AMT).
- Routine for image based correction of the signal dilution effect based on contrast variations of dark terrain features located at different distances in the images.
- Support of standard image formats including FITS format.
- Easy and flexible setup for data import and camera specifications.

A detailed description of pyplis and its features (including analysis examples) can be found in Gliss et al., 2017, MDPI Geosciences.

1.4 Installation instructions and Requirements

We recommend using the Anaconda Python distribution (or Miniconda, if you want to save disk space) and to use the *conda* package manager. Why? See, e.g. here for some good reasons.

Below it is assumed that you made yourself familiar with the *conda* package manager and that it is installed on your system. It is recommended to have a look at the guidelines related to conda virtual environments.

1.4.1 Comment regarding conda environments

We highly recommend to work in individual conda environments for your different projects and not to install everything into your Anaconda root environment (base), which is usually activated by default. In other words: please do not install pyplis into your root environment but create a new one using:

```
conda create -n my_awesome_conda_environment
```

Why?

1.4.2 Installation using conda

Pyplis is available via the conda-forge channel and can be easily installed via:

```
conda install -c conda-forge pyplis
```

This will install all requirements as well. This is the recommended (and by far easiest) way to get pyplis running on your system.

1.4.3 Requirements

Before installing pyplis, make sure you have all requirements installed (which is done automatically if you install pyplis via conda as described in previous section).

A list of all mandatory requirements can be found in the provided conda environment file pyplis_env.yml, which can also directly be used to install the requirements, as described below.

Optional dependencies (to use extra features)

- Pillow (PIL fork) >= 3.3.0
  - may be used to define custom image read functions, see e.g. this example
  - We recommend using `pip install pillow` rather than `conda install pillow` due to
  - well known installation issues, e.g. here
- pydoas >= 1.0.0 (comes with conda installation and provided environment file)

1.4.4 Installation of the requirements

Before installing *Pyplis*, you need to install all requirements. To do so, you may either use the provided conda environment file or install all requirements manually, as described in the following two sections. All instructions below assume that you use Anaconda as package manager.

Installation of requirements using provided conda environment file

You can install all mandatory requirements using the provided environment file *pyplis_env.yml* (or *pyplis_env_py27.yml* if you still use python 2.7). You can install the environment file into a new environment (here, named *pyplis*) using:

```
conda env create -n pyplis_env_test -f pyplis_env.yml
```

Or you may install it into an existing environment by activating the environment and then:

```
conda env update -f=pyplis_env.yml
```

Manual installation of requirements

You may also install all requirements from scratch as described in the following step-by-step guide:

```
conda create --name pyplis # creates new conda environment with name pyplis (optional)
conda activate pyplis # activates new environment (optional)
conda install -c conda-forge scipy pandas astropy basemap opencv geonum pydoas
```

1.4.5 Installation of pyplis

Here, you have 3 options.

Via conda

From the command line, call:

```
conda install -c conda-forge pyplis
```

This option installs pyplis and all requirements automatically.

Via pip

From the command line, call:

```
pip install pyplis
```

This option only installs pyplis, you have to install all requirements yourself (for details, see previous sections).

From Source

In order to install from source, please download or clone the repo (or one of the pyplis releases) into a local directory of your choice. Then, unzip and from the project root directory (the one that contains setup.py file) call:

```
python setup.py install
```

This option only installs pyplis, you have to install all requirements yourself (for details, see previous sections).

Note

Use Option 2 if you want to run the tests and / or example scripts (since these are not shipped with the PyPi installation that uses a binary wheel of Pyplis).

1.4.6 Installation remarks and known issues

- If you work on a Windows machine and run into problems with installation of one of the requirements (e.g. if you already had Python 2.7 installed and want to upgrade dependencies such as numpy or scipy), check out the pre-compiled binary wheels on Christoph Gohlke's webpage
- Sometimes it is helpful, to reinstall your whole Python environment (or, if you use Anaconda, create a new one) rather than trying to upgrade all dependencies to the required version
- If you find a bug or detect a specific problem with one of the requirements (e.g. due to future releases) please let us know or raise an issue.

Do not hesitate to contact us (or raise an issue), if you have problems installing pyplis.

1.5 Getting started

The Pyplis example scripts (see previous point) are a good starting point to get familiar with the features of Pyplis and for writing customised analysis scripts. The scripts require downloading the Etna example dataset (see following section for instructions). If you require more thorough testing, refer to this wiki entry

1.6 Example and test data

The pyplis example data (required to run example scripts) is not part of the installation. It can be downloaded from here or automatically downloaded in a Python shell (after installation) using:

```python
import pyplis
pyplis.inout.download_test_data(<desired_location>)
```

which downloads the data into the *my_pyplis* directory if <desired_location> is unspecified. Else, (and if <desired_location> is a valid location) it will be downloaded into <desired_location> which will then be added to the supplementary file *_paths.txt* located in the installation *data* directory. It can then be found by the test data search method:

```
pyplis.inout.find_test_data()
```

The latter searches all paths provided in the file *_paths.txt* whenever access to the test data is required. It raises an Exception, if the data cannot be found.

1.6.1 Note

If the data is downloaded manually (e.g. using the link provided above), please make sure to unzip it into a local directory *<desired_location>* and let pyplis know about it, using:

```
import pyplis
pyplis.inout.set_test_data_path(<desired_location>)
```

1.7 Scientific background

The article:

Pyplis - A Python Software Toolbox for the Analysis of SO2 Camera Images for Emission Rate Retrievals from Point Sources, Gliß, J., Stebel, K., Kylling, A., Dinger, A. S., Sihler, H., and Sudbø, A., Geosciences, 2017

introduces *Pyplis* and implementation details. Furthermore, the article provides a comprehensive review of the technique of SO2 cameras with a focus on the required image analysis. The paper was published in December 2017 as part of a special issue on Volcanic plumes of the Journal *Geosciences* (MDPI). Download paper.

1.7.1 Citation

If you find *Pyplis* useful for your data analysis, we would highly appreciate if you acknowledge our work by citing the paper. Citing details can be found here.

1.8 Copyright

Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License a published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see here.

1.9 Visualisation of API architecture

The following two flowcharts illustrate details about the Pyplis architecture, for details, please see section article.

1.9.1 Flowchart emission-rate retrieval (scientific)

The following flowchart illustrates the main analysis steps for emission-rate retrievals using UV SO2 cameras. The colours indicate geometrical calculations (yellow), background modelling (light grey), camera calibration (light blue), plume speed retrieval (light purple) and the central processing steps for the emission-rate retrieval (light green). Shaded and dashed symbols indicate optional or alternative analysis methods.

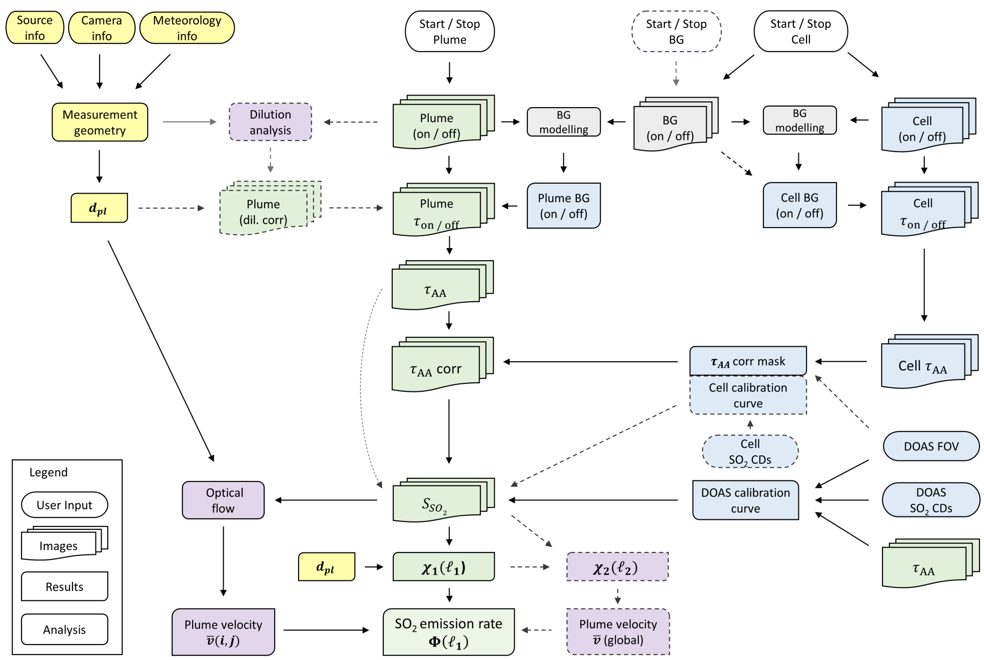


Fig. 1: Flowchart showing the main analysis steps for emission rate retrievals

1.9.2 Flowchart API (code architecture)

The following flowchart illustrates the most relevant classes / methods of the *Pyplis* API with a focus on the required routines for SO2 emission-rate retrievals. Italic denotations correspond to class names in Pyplis. Optional / alternative analysis procedures are indicated by dashed boxes. Setup classes (red) include relevant meta information and can be used to create Dataset objects (blue). The latter perform file separation by image type and create ImgList objects (green) for each type (e.g. on, off, dark). Further analysis classes are indicated in yellow. Note that the routine for signal dilution correction is not shown here.

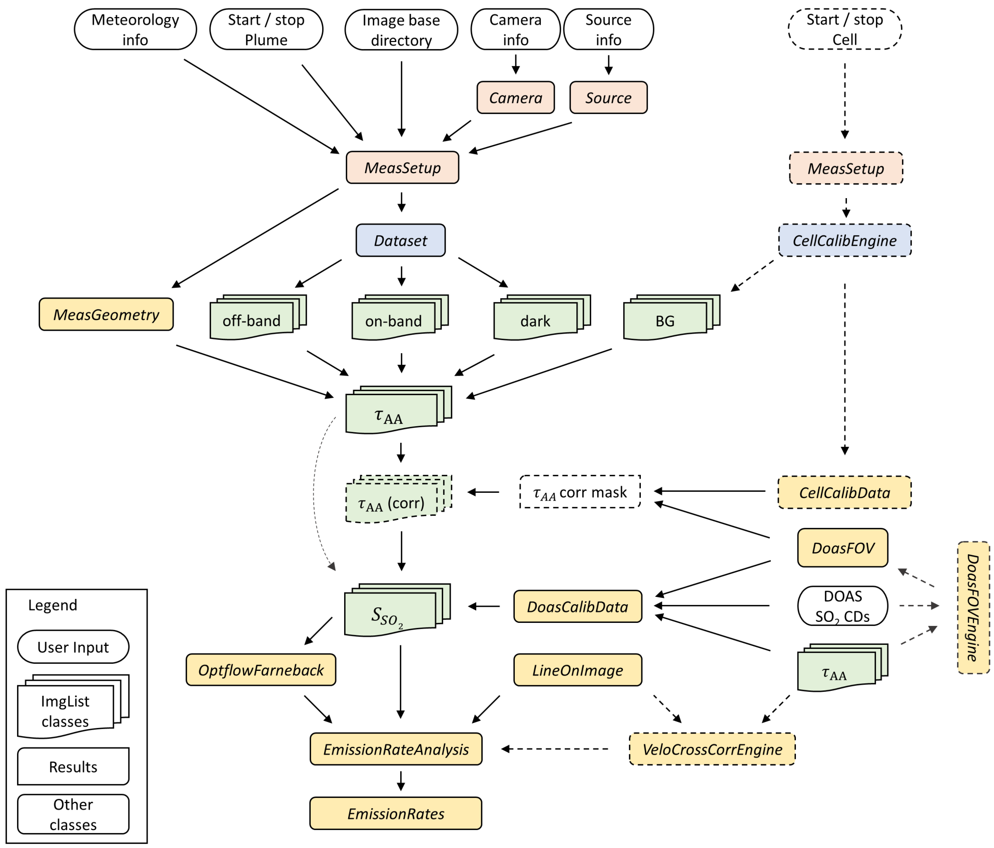


Fig. 2: Flowchart illustrating the basic architecture of pyplis (note: the engine for signal dilution correction is not included here).

CHAPTER 2

Plot gallery

This page contains a collection of plots from the Etna example data.

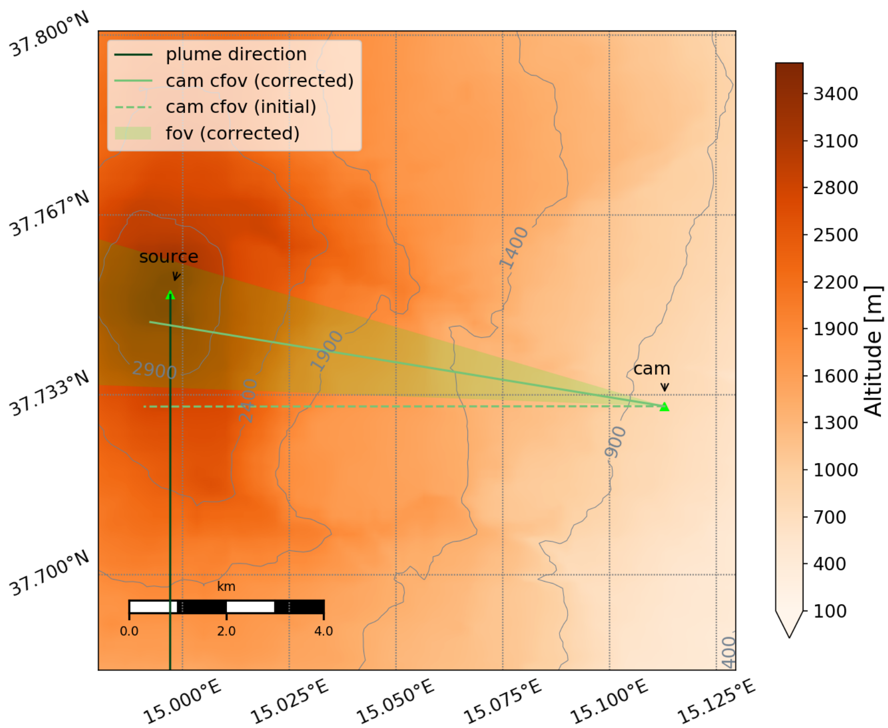


Fig. 1: 2D map showing a measurement setup (automatically created using class `MeasGeometry`)

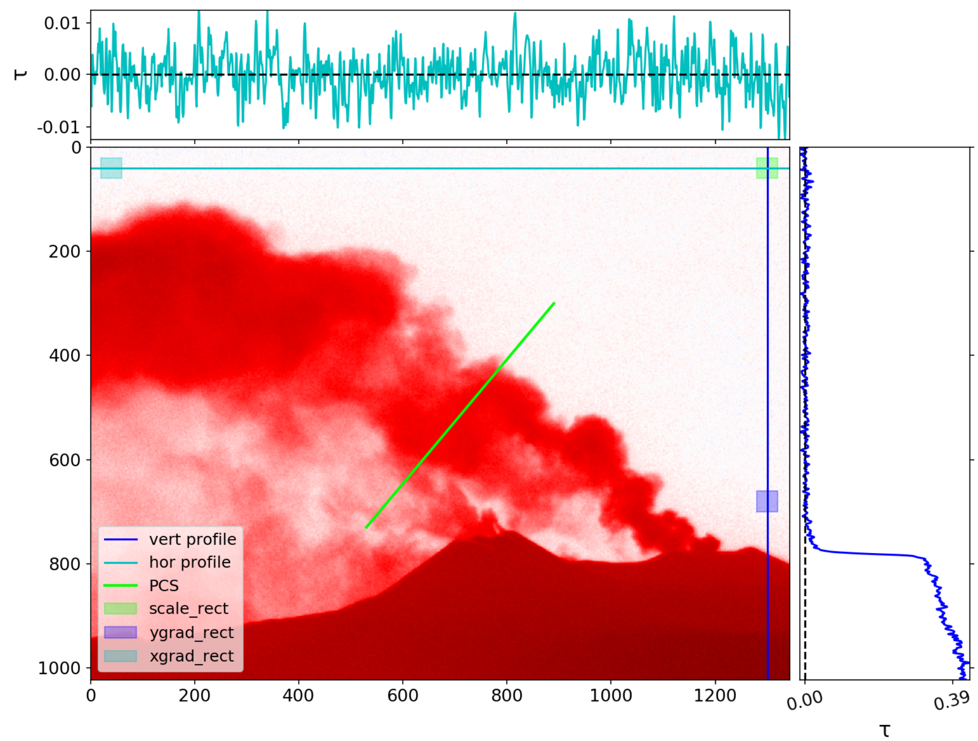


Fig. 2: On-band optical density image determined using plume background modelling mode 6 in class `PlumeBackgroundModel`

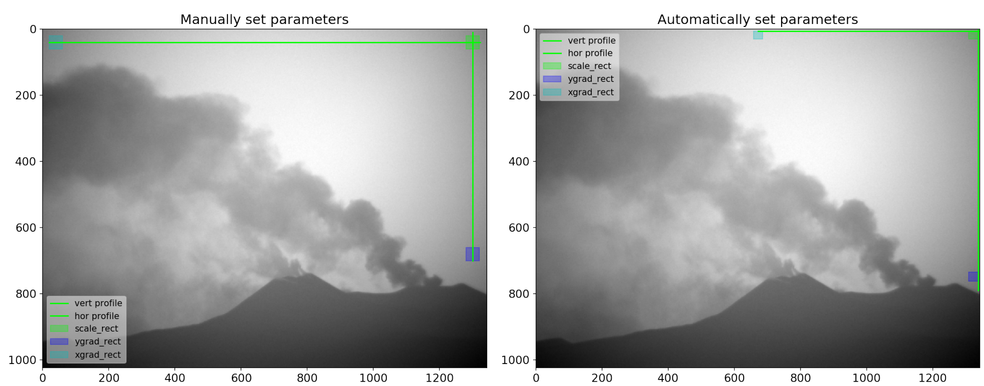


Fig. 3: Exemplary sky reference areas for plume background modelling, left: set manually, right: set automatically (cf. example script 3)

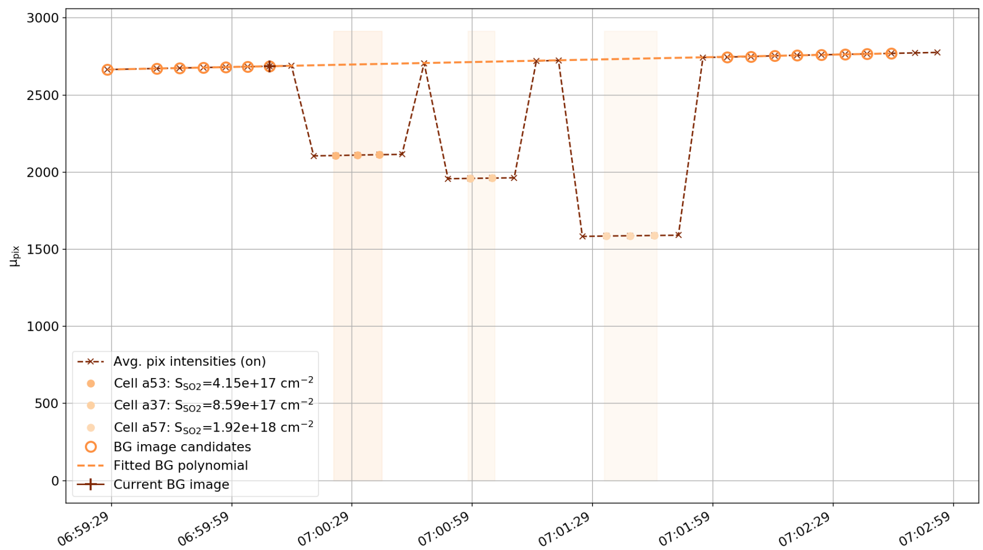


Fig. 4: Result of routine for automatic detection of SO2 cell time windows (from time series of on-band images, cf. example script 5)

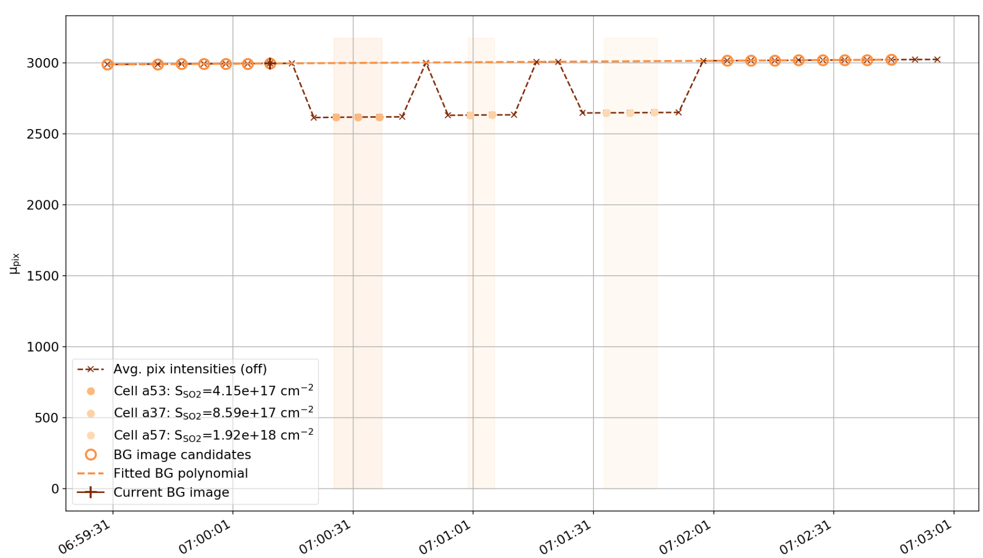


Fig. 5: Result of routine for automatic detection of SO2 cell time windows (from time series of off-band images, cf. example script 5)

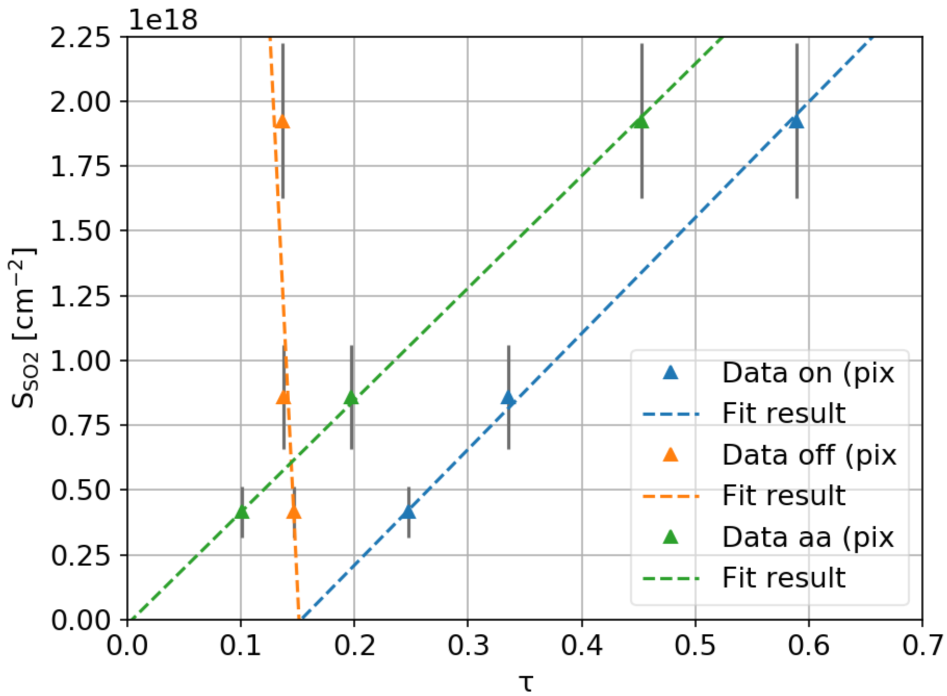


Fig. 6: Exemplary SO2 cell calibration curves (for center image pixel, cf. example script 5)

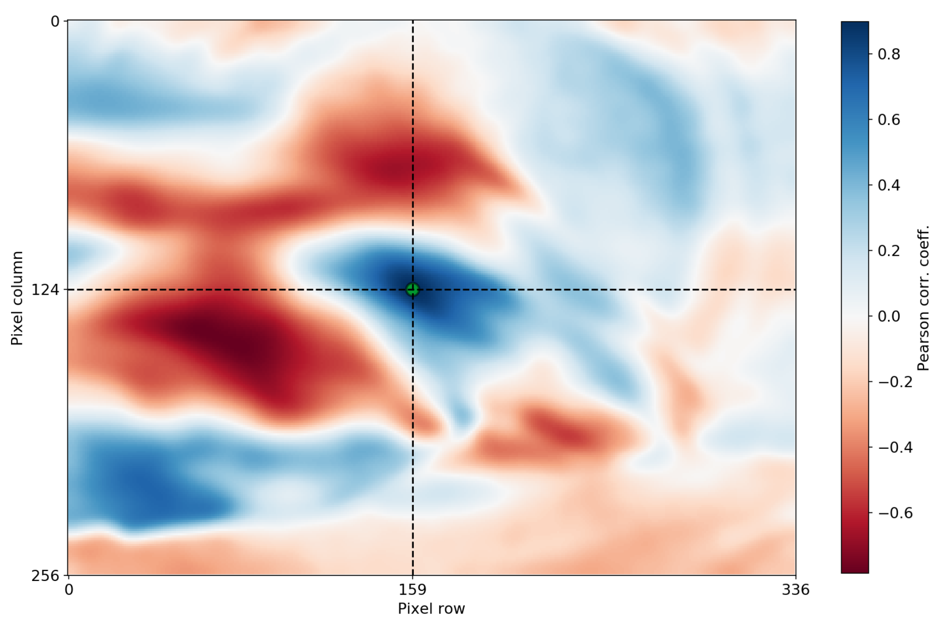


Fig. 7: Result of DOAS FOV search using Pearson correlation method (cf. example script 6)

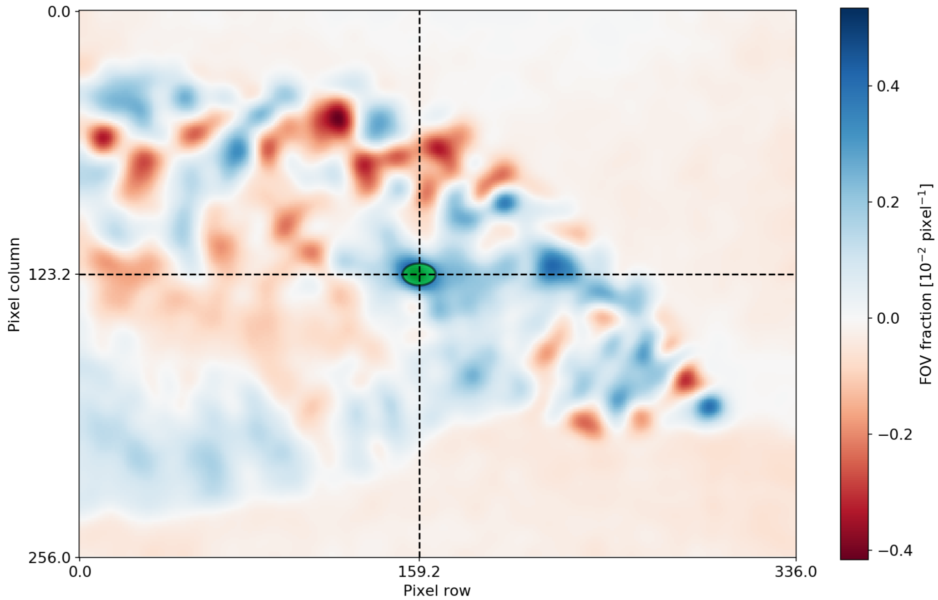


Fig. 8: Result of DOAS FOV search using IFR method (cf. example script 6)

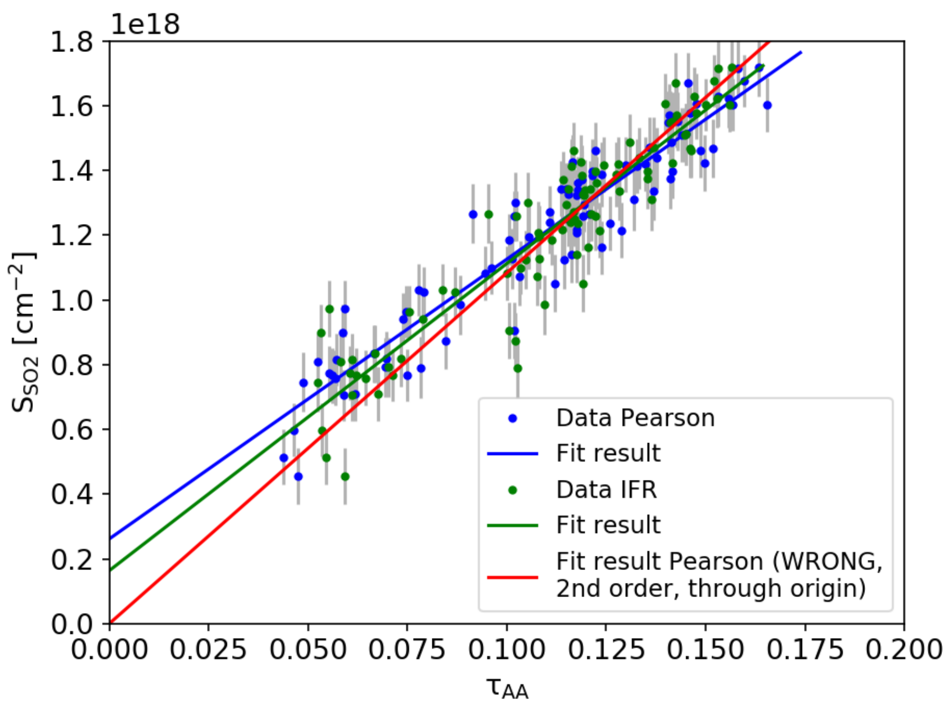


Fig. 9: Exemplary DOAS calibration curves determined using the FOV results shown in the prev. 2 Figs. (cf. example script 6)

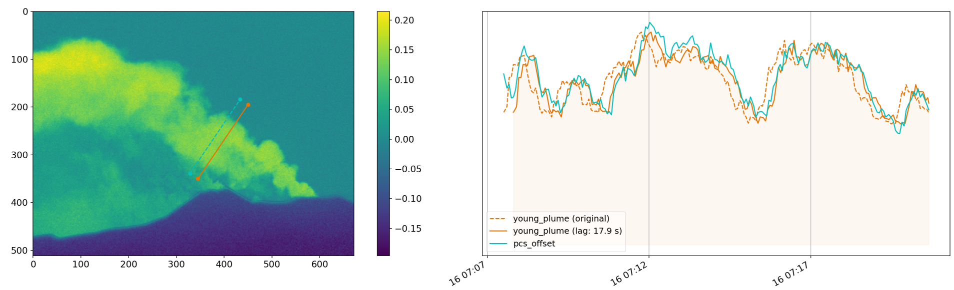


Fig. 10: Left: plume AA image including two plume cross section lines used for cross correlation based plume velocity retrieval. Right: Result of cross correlation analysis using the two PCS lines shown left resulting in a velocity of 4.29 m/s (cf. example script 8)

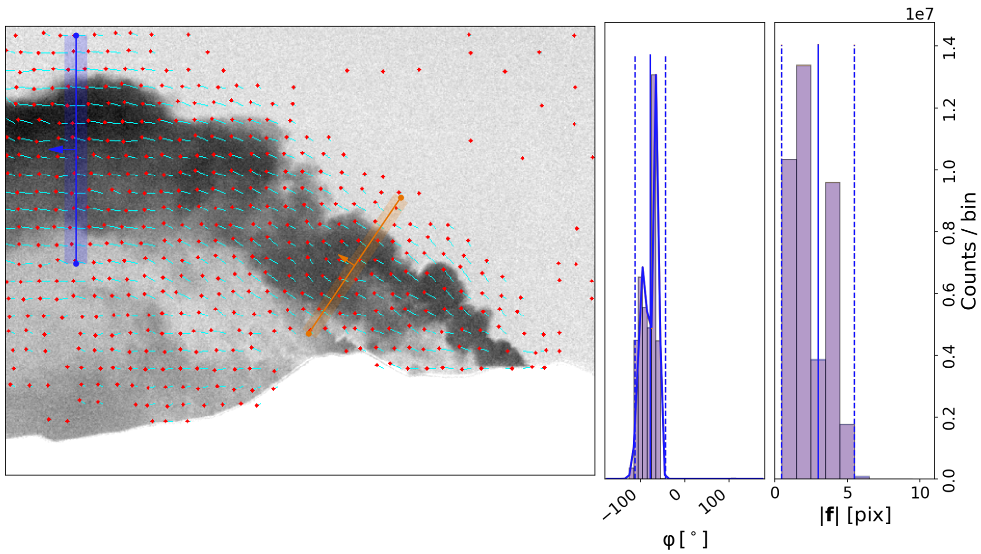


Fig. 11: Example output of optical flow Farneback algorithm (left) including histograms of orientation angles (middle) and flow vector magnitudes (right) retrieved within ROIs around both lines. Retrieved expectation values and intervals, derived from 1. and 2. moments of the histograms are indicated by solid and dashed lines, respectively (cf. ex. script 9).

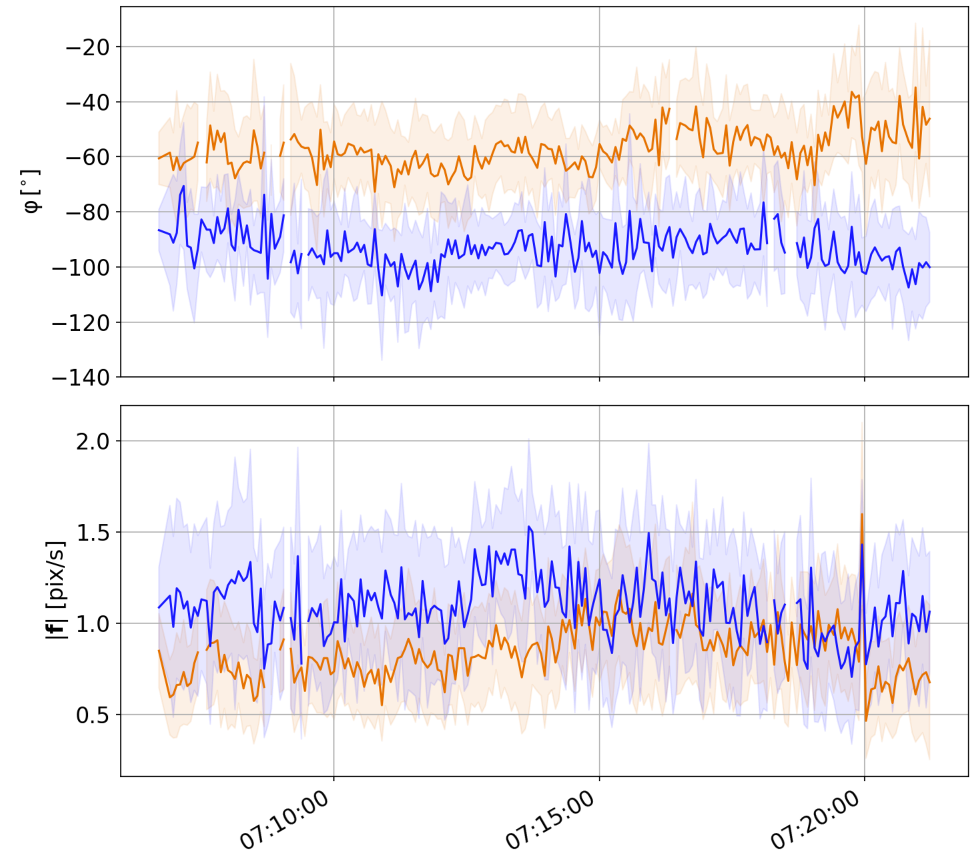


Fig. 12: Time series of plume velocity parameters (direction, top; displacement length, bottom) retrieved using histogram based post analysis of optical flow field for the two retrieval lines shown in prev. Fig. (cf. ex. script 9)

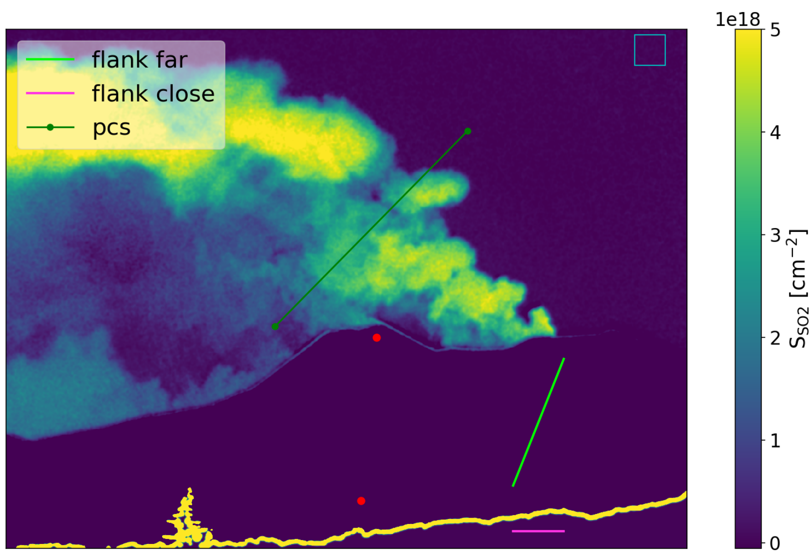


Fig. 13: SO2-CD image corrected for signal dilution using pixels along terrain features in the images (lime and blue lines) to estimate atmospheric extinction coefficients.

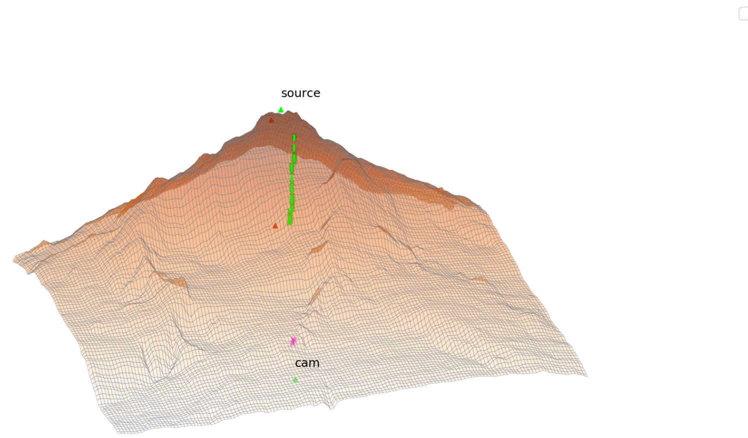


Fig. 14: 3D map showing results of pixel based distance retrieval to terrain features used for signal dilution correction (cf. prev. Fig.)

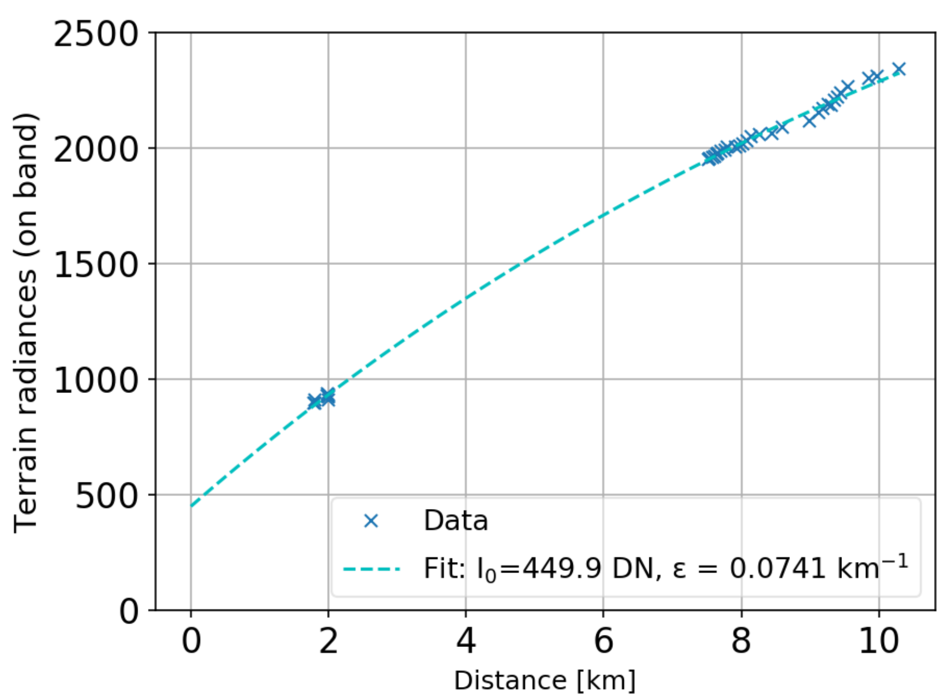


Fig. 15: Result of signal dilution correction fit to retrieve atmospheric extinction coefficients (on-band)

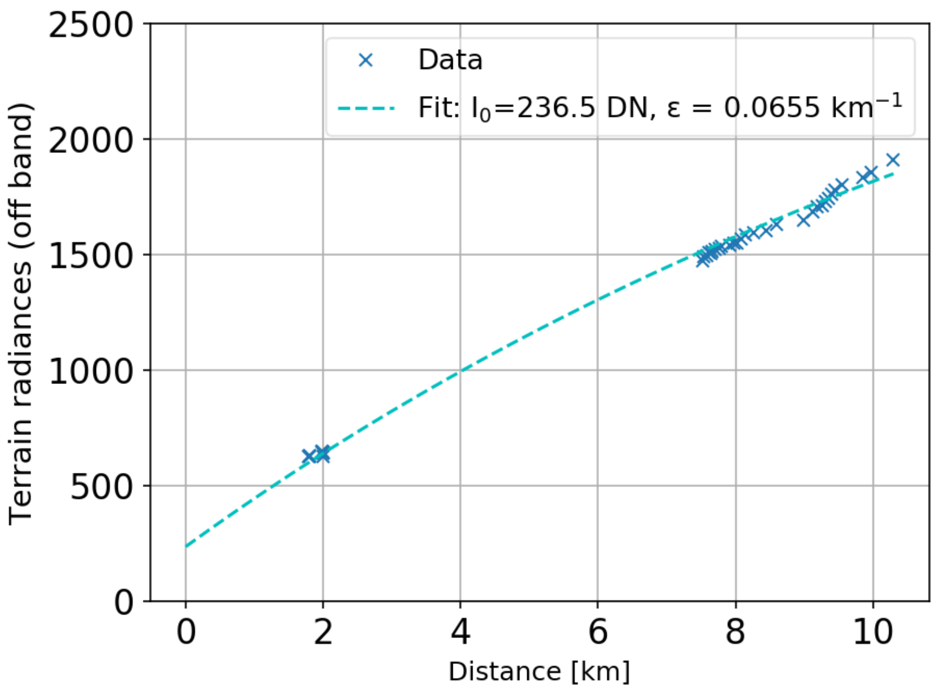


Fig. 16: Result of signal dilution correction fit to retrieve atmospheric extinction coefficients (off-band)

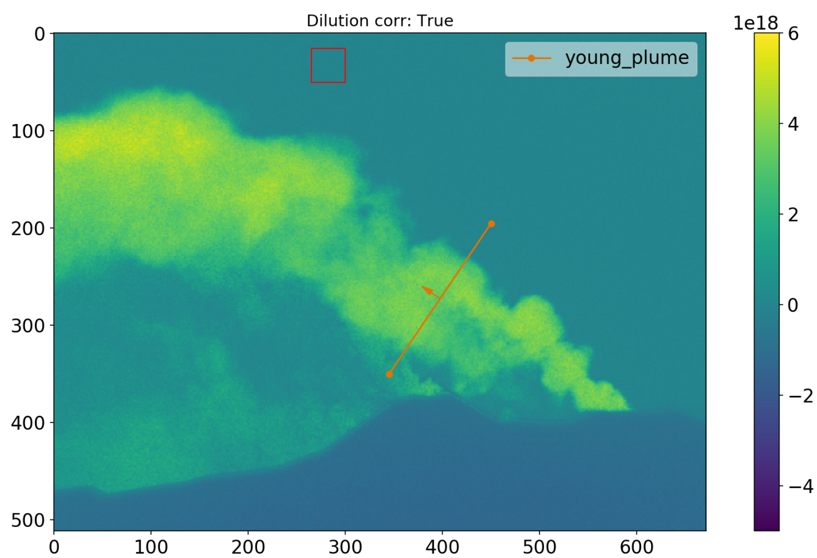


Fig. 17: Calibrated SO2-CD image of the Etna plume (not dilution corrected) including retrieval line L (young_plume) and area (red rectangle) used as quality check when performing emission rate analysis (cf. bottom panel, next plot).

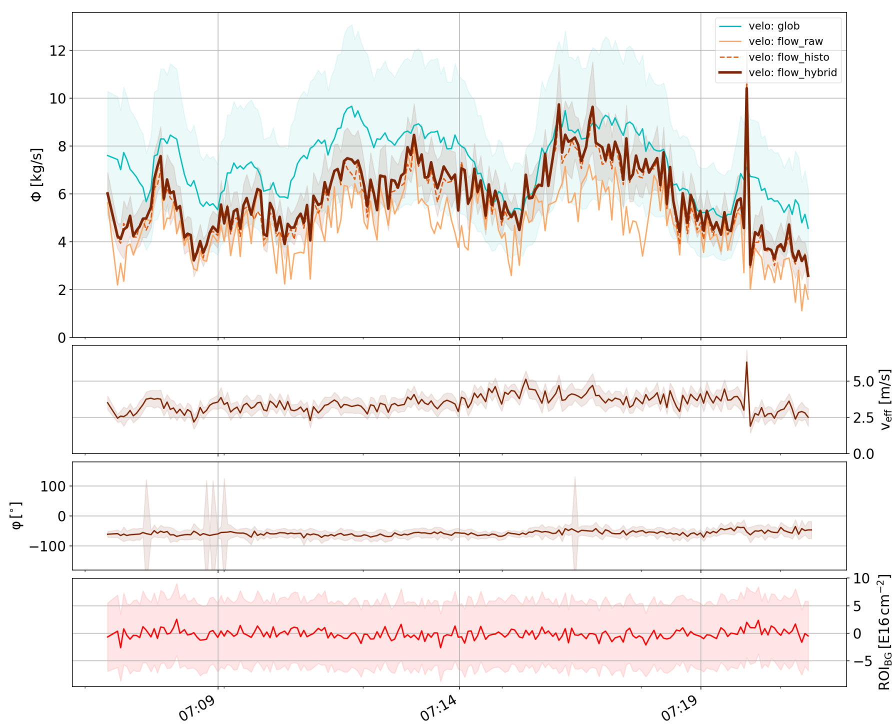


Fig. 18: Etna emission rates through L (see prev. Fig) using four different plume velocity retrievals (top, see legend), and velocity results from histogram analysis (2., 3. panel). Bottom: time series of retrieved background CDs in gas free rectangular area (cf. prev. Fig.).

CHAPTER 3

Primer on data import

3.1 Data import - Specifying custom camera information

In order to use all features of pyplis, certain specifications related to camera and image acquisition need to be defined. Basic information about the camera (e.g. detector specifics) and the corresponding file convention (image type, which data can be extracted from file names) are specified within `Camera` objects. This tutorial introduces the `Camera` class and how to set up your custom camera type based on your data format, including definitions of your file naming convention.

3.1.1 Specifying your file naming convention

At the very beginning of the analysis, the images need to be imported and separated by image type (e.g. on-band plume, off-band plume, dark, offset, on / off-band cell calibration). In order to use the automated separation for a given dataset (e.g. a single folder `IMG_DIR` containing images of all types) it is required that the image type can be identified from the file names.

The relevant information for identifying different image types (e.g. plume on-band, dark, offset) can be specified using either of the following two classes:

1. `Filter`: specifies file access information for all image types that

are NOT dark or offset images (e.g. on / off images plume / background)

1. `DarkOffsetInfo`: specifies different types of dark images and

offset images.

Such a collection of `Filter` and `DarkOffsetInfo` objects is then stored within a `Camera` object.

These information is used to separate the individual image types when creating a `Dataset` object. The latter searches all valid image files in a given folder `IMG_DIR` and creates `ImgList` objects for each `Filter` and `DarkImgList` objects for each `DarkOffsetInfo` object defined in the `Camera`. Each of these lists is then filled with the file paths of the corresponding image type located in `IMG_DIR`. The `Camera` object furthermore includes relevant specs of the camera (e.g. pixel geometry, lens).

The following list provides an overview of relevant parameters for filename access information using exemplary file-names of the ECII camera type.

3.1.2 The ECII camera standard

In the following, an exemplary `Camera` class is specified based on the ECII-camera standard and file naming convention (cf. *Example 0.2 - The camera class*).

To start with, an empty `Camera` instance is created:

```
cam = pyplis.Camera()
# prints the string representation which gives a nice overview over the
# relevant parameters
print cam
```

If you wish to store the camera as default you need to specify a unique camera ID (string) which is not yet used for any of the pyplis default cameras stored in the file *cam_info.txt* (package data). You can check all existing IDs using:

```
print pyplis.inout.get_all_valid_cam_ids()
```

Let's call our new camera "ecII_test":

```
cam.cam_id = "ecII_test"
```

Now specify some relevant attributes of the camera, starting with the image file type:

```
cam.file_type = "fts"
```

You can also provide information about detector and camera lens:

```
cam.focal_length = 25e-3 #m

# Detector geometry
cam.pix_height = 4.65e-6 # pixel height in m
cam.pix_width = 4.65e-6 # pixel width in m
cam.pixnum_x = 1344
cam.pixnum_y = 1024
```

In the following, the camera file naming convention is specified. This enables to extract certain information from the image file names (e.g. image acq. time, image type, exposure time).

Start with setting the file name delimiter of your file naming convention:

```
cam.delim = "_"
```

Based on that, specify the position of acquisition time (and date) in the image file names after splitting with delimiter:

```
cam.time_info_pos = 3
```

The acq. time strings in the file names need to be converted into `datetime` objects thus, specify the string for internal conversion (is done using `datetime.strptime()`):

```
cam.time_info_str = "%Y%m%d%H%M%S%f"
```

If the file name also includes the image exposure time, this can also be specified:

```
cam.texp_pos = "" #the ECII does not...
```

as well as the unit (choose from “s” or “ms” if applicable):

```
cam.texp_unit = ""
```

Furthermore, the image type identification can (and should) be specified in the camera, in order to make life easier. This ensures, that images of different types (e.g. on / off-band, dark, offset) can be identified and separated directly from the filename. The relevant information is specified in a collection of `Filter` and `DarkOffsetInfo` objects. Let’s start off with defining the different image access types for on and off-band images (these are stored in `Filter` objects, while dark / offset image access information is stored in `DarkOffsetInfo` objects, follows below):

```
# On-band images
on = pyplis.Filter(id="on", type="on", acronym="F01",
                   meas_type_acro="F01", center_wavelength=310)
# Off-band images
off = pyplis.Filter(type="off", acronym="F02",
                    meas_type_acro="F02", center_wavelength=330)
```

Now add the two filters to the camera (i.e. put them into a list and assign it to the camera):

```
filters = [on, off]

cam.default_filters = filters

# Checks and sets filters in cam
cam.prepare_filter_setup()
```

Tell the camera, which of the filters is the “central” filter for the emission rate analysis (usually “on”):

```
cam.main_filter_id = "on"
```

The latter information is used for internal linking of image lists within a `Dataset` object, for instance, if the camera contains multiple `type="on"` filters (i.e. on-band SO2).

Note: This parameter `main_filter_id` is irrelevant for standard setups like here

(i.e. one on and one off-band filter)

Similar to the filter setup (which specifies access to the actual images to be analysed), the filename access information for dark (`type=dark`) and offset (`type=offset`) image identification needs to be specified using `DarkOffsetInfo` instances:

```
offset_low_gain  = pyplis.DarkOffsetInfo(id="offset0",type="offset",
                                         acronym="D0L",
                                         meas_type_acro="D0L",
                                         read_gain=0)

offset_high_gain = pyplis.DarkOffsetInfo(id="offset1",type="offset",
                                          acronym="D0H", read_gain=1)
dark_low_gain    = pyplis.DarkOffsetInfo(id="dark0",type="dark",
                                          acronym="D1L", read_gain=0)
dark_high_gain   = pyplis.DarkOffsetInfo(id="dark1",type="dark",
                                          acronym="D1H", read_gain=1)
```

(continues on next page)

(continued from previous page)

```
# put the 4 dark info objects into a list and assign to the camera
dark_info = [offset_low_gain, offset_high_gain,
             dark_low_gain, dark_high_gain]

cam.dark_info = dark_info
```

Note: You might have recognised, that in the last 3 `DarkOffsetInfo``

objects, the meas_type_acro was not specified. This is because it is actually irrelevant for the ECII camera which does not include a sub string specifying different measurement modi like, for instance, the HD-Custom camera (i.e. K, M, D).

Now that all different image types are specified, the camera needs to know where to find the actual information in the file names (after splitting using `delim`). The position of the strings specified in the attribute `acronym` (see definitions of the `Filter` and `DarkOffsetInfo` objects above) can be set using:

```
cam.acronym_pos = 4
```

and the position of the strings specified in attribute `meas_type_acro`:

```
cam.meas_type_acro_pos = 4
```

Note: If `meas_type_acro` is irrelevant (like for this camera) it is required to

be set equal `acronym_pos`

Furthermore, the dark correction type needs to be specified, pyplis includes two options for that, the ECII uses option 1:

```
cam.DARK_CORR_OPT = 1
```

Todo: Include information about the two different dark correction modes

That's it! You might want to check if everything is in place:

```
print cam
```

If you are happy, you might want to check if the data access from the file names works. You can do a fast check using a file path `IMG_PATH` to one of your images and run:

```
acq_time, filter_id, meas_type, texp, warnings =\
                  cam.get_img_meta_from_filename(IMG_PATH)
```

You might also test it for a whole dataset of images located in a directory `IMG_DIR` and check if pyplis can identify the different image types. You can do this, for instance, by creating a `Dataset` object. First, create a measurement setup with minimum information:

```
meas_setup = pyplis.MeasSetup(base_dir=IMG_DIR, camera=cam)
```

and create a Dataset from that:

```
ds = pyplis.Dataset(meas_setup)
```

The Dataset object should now detect all individual image types and puts them into separate lists, which can be accessed using the IDs of the corresponding Filter objects, e.g.:

```
lst = ds.get_list("on")
print "Number of images in list: %d" %lst.nof
```

These lists are of type ImgList. Similarly, dark and offset image lists (DarkImgList classes) were created using the information stored in the DarkOffsetInfo objects specified in our camera:

```
dark_list_low_gain = ds.get_list("dark0")
offset_list_low_gain = ds.get_list("offset0")
```

You can also easily access all lists, that actually contain images (i.e. for which image matches could be found in IMG_DIR), e.g. all lists that contain images and correspond to one of the Filter objects:

```
all_imglists = ds.img_lists_with_data #this is a dictionary
print all_imglists.keys() #prints the list / Filter IDs
```

and similar, all DarkImgList objects that contain data:

```
all_darklists = ds.dark_lists_with_data #this is a dictionary
print all_darklists.keys() #prints the list IDs
```

If everything works out nicely, you can add the camera as new default using:

```
cam.save_as_default()
```

After saving the camera as new default, you can load it using:

```
import pyplis
cam = pyplis.Camera(cam_id="ecII_test")
print cam
```

Done!

CHAPTER 4

Example scripts

pyplis example scripts. The scripts require downloading the pyplis test data.

Note: The scripts are based on the latest commit in the GitHub repo. If you have installed an older version of pyplis, please use the corresponding scripts which are provided here.

4.1 Introductory scripts

These scripts give an introduction into basic features and classes of pyplis.

4.1.1 Example 0.1 - Image representation

Introduction into `Img` object and basic usage including correction for dark current and detector offset.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
```

(continues on next page)

(continued from previous page)

```
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis introduction script 1 - Image import.

This script loads an exemplary image which is part of the pyplis supplied
data. Image data in pyplis is represented by the ``Img`` class, which also
allows for storage of image meta data and keeps track of changes applied to
the image data (e.g. cropping, blurring, dark correction).

The script also illustrates how to manually work with image meta data
encoded in the image file name. The latter can be performed automatically in
pyplis using file name conventions (which can be specified globally, see next
script).
"""
from __future__ import (absolute_import, division)

# Check script version
from SETTINGS import check_version

# imports from SETTINGS.py
from SETTINGS import OPTPARSE, SAVE_DIR

from os.path import join, basename
from datetime import datetime
from matplotlib.pyplot import close
import pyplis

check_version()

# file name of test image stored in data folder
IMG_FILE_NAME = "test_201509160708_F01_335.fts"

IMG_DIR = join(pyplis._LIBDIR, "data")

if __name__ == "__main__":
    close("all")

    img_path = join(IMG_DIR, IMG_FILE_NAME)

    # Create Img object (Img objects can be initiated both with image file
    # paths but also with data in memory in form of a 2D numpy array)
    img = pyplis.image.Img(img_path)

    # log mean of uncropped image for testing mode
    avg = img.mean()

    # The file name of the image includes some image meta information which can
    # be set manually (this is normally done automatically by defining a file
    # name convention, see next script)

    # split filename using delimiter "_"
    spl = IMG_FILE_NAME.split(".")[0].split("_")

    # extract acquisition time and convert to datetime
    acq_time = datetime.strptime(spl[1], "%Y%m%d%H%M")
    # extract and convert exposure time from filename (convert from ms -> s)
```

(continues on next page)

(continued from previous page)

```
    texp = float(spl[3]) / 1000

    img.meta["start_acq"] = acq_time
    img.meta["texp"] = texp
    img.meta["f_num"] = 2.8
    img.meta["focal_length"] = 25e-3

    # add some blurring to the image
    img.add_gaussian_blurring(sigma_final=3)

    # crop the image edges
    roi_crop = [100, 100, 1244, 924]  # [x0, y0, x1, y1]
    img.crop(roi_abs=roi_crop)

    # apply down scaling (gauss pyramid)
    img.to_pyrlevel(2)

    # ## Show image
    img.show()

    img.save_as_fits(SAVE_DIR, "ex0_1_imgsave_test")

    img_reload = pyplis.Img(join(SAVE_DIR, "ex0_1_imgsave_test.fts"))

    # print image information
    print(img)

    # ## IMPORTANT STUFF FINISHED - everything below is of minor importance
    # for educational purposes

    (options, args) = OPTPARSE.parse_args()
    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt

        vals = [img.mean(), avg, int(spl[1]), texp,
                img_reload.meta["texp"],
                img_reload.meta["f_num"],
                img_reload.meta["focal_length"]]

        npt.assert_almost_equal([2526.4623422672885,
                                 2413.0870433989026,
                                 201509160708,
                                 0.335,
                                 0.335,
                                 2.8,
                                 0.025],
                                vals, 4)

        print("All tests passed in script: %s" % basename(__file__))
```

4.1.2 Example 0.2 - The camera class

Introduction into the `Camera` object using the example of the ECII camera standard.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Introduction script 2 - The Camera class.

This script introduces the camera class which is of fundamental importance
for image import (e.g. separating on, off, dark background images, etc.) and
also for the data analysis as it includes all relevant detector specifications,
such as number of pixels, pixel size, focal length, etc.

In this script, a newer version of the camera type "ecII" is created manually
in order to illustrate all relevant parameters. The only difference to the
classic ecII camera is, that the filter setup is different.

See also here for more information:

https://pyplis.readthedocs.io/en/latest/tutorials.html#data-import-specifying-
custom-camera-information
"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version, OPTPARSE

from os.path import basename
from numpy.testing import assert_array_equal

import pyplis

# Check script version
check_version()

# ## SCRIPT OPTIONS

# Save the new camera as default in database (cam_info.txt file that can be
# found in the "data" directory of the installation.
SAVE_TO_DATABASE = True
```

(continues on next page)

(continued from previous page)

```
# ## SCRIPT FUNCTION DEFINITIONS
def create_ecII_cam_new_filters():
    # Start with creating an empty Camera object
    cam = pyplis.setupclasses.Camera()

    # Specify the camera filter setup

    # Create on and off band filters. Obligatory parameters are "type" and
    # "acronym", "type" specifies the filter type ("on" or
    # "off"), "acronym" specifies how to identify this filter in the file
    # name. If "id" is unspecified it will be equal to "type". The parameter
    # "meas_type_acro" is only important if a measurement type (e.g. M -> meas,
    # C -> calib ...) is explicitely specified in the file name.
    # This is not the case for ECII camera but for the HD camera,
    # see specifications in file cam_info.txt for more info.

    on_band = pyplis.utils.Filter(id="on", type="on", acronym="F01",
                                  meas_type_acro="F01", center_wavelength=310)
    off_band = pyplis.utils.Filter(type="off", acronym="F02",
                                   center_wavelength=330)

    # put the two filter into a list and assign to the camera
    filters = [on_band, off_band]

    cam.default_filters = filters
    cam.prepare_filter_setup()

    # Similar to the filter setup, access info for dark and offset images needs
    # to be specified. The ECII typically records 4 different dark images, two
    # recorded at shortest exposure time -> offset signal predominant, one at
    # low and one at high read gain. The other two recorded at longest possible
    # exposure time -> dark current predominant, also at low and high read gain

    offset_low_gain = pyplis.utils.DarkOffsetInfo(id="offset0", type="offset",
                                                  acronym="D0L", read_gain=0)
    offset_high_gain = pyplis.utils.DarkOffsetInfo(id="offset1", type="offset",
                                                   acronym="D0H", read_gain=1)
    dark_low_gain = pyplis.utils.DarkOffsetInfo(id="dark0", type="dark",
                                                acronym="D1L", read_gain=0)
    dark_high_gain = pyplis.utils.DarkOffsetInfo(id="dark1", type="dark",
                                                 acronym="D1H", read_gain=1)

    # put the 4 dark info objects into a list and assign to the camera
    dark_info = [offset_low_gain, offset_high_gain,
                 dark_low_gain, dark_high_gain]

    cam.dark_info = dark_info

    # Now specify further information about the camera

    # camera ID (needs to be unique, i.e. not included in data base, call
    # pyplis.inout.get_all_valid_cam_ids() to check existing IDs)
    cam.cam_id = "ecII_new_test"

    # image file type
    cam.file_type = "fts"
```

(continues on next page)

(continued from previous page)

```
    # File name delimiter for information extraction
    cam.delim = "_"

    # position of acquisition time (and date) string in file name after
    # splitting with delimiter
    cam.time_info_pos = 3

    # datetime string conversion of acq. time string in file name
    cam.time_info_str = "%Y%m%d%H%M%S%f"

    # position of image filter type acronym in filename
    cam.filter_id_pos = 4

    # position of meas type info
    cam.meas_type_pos = 4

    # Define which dark correction type to use
    # 1: determine a dark image based on image exposure time using a dark img
    # (with long exposure -> dark current predominant) and a dark image with
    # shortest possible exposure (-> detector offset predominant). For more
    # info see function model_dark_image in processing.py module
    # 2: subtraction of a dark image recorded at same exposure time than the
    # actual image
    cam.darkcorr_opt = 1

    # If the file name also includes the exposure time, this can be specified
    # here:
    cam.texp_pos = ""  # the ECII does not...

    # the unit of the exposure time (choose from "s" or "ms")
    cam.texp_unit = ""

    # define the main filter of the camera (this is only important for cameras
    # which include, e.g. several on band filters.). The ID need to be one of
    # the filter IDs specified above
    cam.main_filter_id = "on"

    # camera focal length can be specified here (but does not need to be, in
    # case of the ECII cam, there is no "default" focal length, so this is left
    # empty)
    cam.focal_length = ""

    # Detector geometry
    cam.pix_height = 4.65e-6  # pixel height in m
    cam.pix_width = 4.65e-6  # pixel width in m
    cam.pixnum_x = 1344
    cam.pixnum_y = 1024

    cam._init_access_substring_info()

    cam.io_opts = dict(USE_ALL_FILES=False,
                       SEPARATE_FILTERS=True,
                       INCLUDE_SUB_DIRS=True,
                       LINK_OFF_TO_ON=True)

    # Set the custom image import method
    cam.image_import_method = pyplis.custom_image_import.load_ecII_fits
```

(continues on next page)

(continued from previous page)

```
    # That's it...
    return cam


# ## SCRIPT MAIN FUNCTION
if __name__ == "__main__":

    cam = create_ecII_cam_new_filters()

    print(cam)

    try:
        # you can add the cam to the database (raises error if ID conflict
        # occurs, e.g. if the camera was already added to the database)
        cam.save_as_default()
    except KeyError:
        print("Camera already exists in database")
    cam_reload = pyplis.Camera("ecII_brandnew")
    # ## IMPORTANT STUFF FINISHED - everything below is of minor importance
    # for educational purposes

    (options, args) = OPTPARSE.parse_args()
    # apply some tests. This is done only if TESTMODE is active: testmode can
    # be activated globally (see SETTINGS.py) or can also be activated from
    # the command line when executing the script using the option --test 1
    if int(options.test):
        # quick and dirty test

        cam_dict_nominal = {'darkcorr_opt': 1,
                            '_fid_subnum_max': 1,
                            '_fname_access_flags': {'filter_id': False,
                                                    'meas_type': False,
                                                    'start_acq': False,
                                                    'texp': False},
                            '_mtype_subnum_max': 1,
                            '_time_info_subnum': 1,
                            'cam_id': 'ecII_new_test',
                            'delim': '_',
                            'file_type': 'fts',
                            'filter_id_pos': 4,
                            'focal_length': '',
                            'image_import_method':
                                pyplis.custom_image_import.load_ecII_fits,
                            'main_filter_id': 'on',
                            'meas_type_pos': 4,
                            'pix_height': 4.65e-06,
                            'pix_width': 4.65e-06,
                            'pixnum_x': 1344,
                            'pixnum_y': 1024,
                            'ser_no': 9999,
                            'texp_pos': '',
                            'texp_unit': '',
                            'time_info_pos': 3,
                            'time_info_str': '%Y%m%d%H%M%S%f'}

        from collections import OrderedDict
        geom_data_nominal = OrderedDict([('lon', None),
```

(continues on next page)

(continued from previous page)

```
                                            ('lat', None),
                                            ('altitude', None),
                                            ('azim', None),
                                            ('azim_err', None),
                                            ('elev', None),
                                            ('elev_err', None),
                                            ('alt_offset', 0.0)])

        arr_nominal = list(geom_data_nominal.items())
        arr_nominal.extend(list(cam_dict_nominal.items()))

        arr_vals = list(cam.geom_data.items())
        for k in cam_dict_nominal:
            arr_vals.append((k, cam.__dict__[k]))

        assert_array_equal(arr_nominal, arr_vals)

        print("All tests passed in script: %s" % basename(__file__))
```

4.1.3 Example 0.3 - Introduction into ImgList objects

Manual creation of `ImgList` objects and basic features.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis introduction script 3 - manual creation of image lists.

This script gives an introduction into the creation and the handling of
ImgList objects. Image lists normally contain images of a certain type (e.g.
on, off, dark, offset). The separation by these file types is normally done
automatically within a Dataset object (see e.g. following script
ex0_4_imglists_auto.py or example script ex01_analysis_setup.py) using a
certain camera type (see previous script ex0_2_camera_setup.py).

In this example these features are NOT used but rather, an on and off band
image list is created manually without previous definition of the Camera type
(i.e. file naming convention, etc.). The script starts with creating an ImgList
containing all images of type FITS found in the test data image base directory.
```

(continues on next page)

(continued from previous page)

```
On and off band lists are then consecutively extracted from this list using
the list method ``separate_by_substr_filename``.

Furthermore, some basic image preparation features of ImgList objects are
introduced (e.g. linking of lists, dark correction, automatic blurring,
cropping, size reduction).
"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version, IMG_DIR, SAVEFIGS, SAVE_DIR, FORMAT, DPI,\
    OPTPARSE

import pyplis
from matplotlib.pyplot import subplots, close, show
from datetime import datetime

from os.path import join, isfile, basename
from os import listdir

# Check script version
check_version()

# ## RELEVANT DIRECTORIES AND PATHS
OFFSET_FILE = join(IMG_DIR, "EC2_1106307_1R02_2015091607064723_D0L_Etna.fts")
DARK_FILE = join(IMG_DIR, "EC2_1106307_1R02_2015091607064865_D1L_Etna.fts")

if __name__ == "__main__":
    close("all")

    # ## Get all images in the image path which are FITS files (actually all)
    all_paths = [join(IMG_DIR, f) for f in listdir(IMG_DIR) if
                 isfile(join(IMG_DIR, f)) and f.endswith("fts")]

    # Let pyplis know that this is the ECII camera standard, so that it is
    # able to import meta information from the FITS header
    cam = pyplis.Camera("ecII")  # loads default info for ECII camera

    # ## Now put them all into an image list

    # Note that the files are not separated by filter type, or dark and offset,
    # etc. so the list simply contains all images of type fts which were found
    # in IMG_DIR
    list_all_imgs = pyplis.imagelists.ImgList(all_paths, list_id="all",
                                              camera=cam)

    # Split the list by on band file type (which is identified by acronym
    # "F01" at 4th position in file name after splitting using delimiter "_")
    # creates two new list objects, one containing matches (i.e. on band images
    # the other containing the rest)
    on_list, rest = list_all_imgs.separate_by_substr_filename(sub_str="F01",
                                                              sub_str_pos=4,
                                                              delim="_")
    # now extract all off band images from the "rest" list
    off_list, rest = rest.separate_by_substr_filename(sub_str="F02",
                                                      sub_str_pos=4,
                                                      delim="_")
```

(continues on next page)

(continued from previous page)

```
# Link the off band list to on band list (the index in the offband list is
# changed whenever the index is changed in the on band list based on acq.
# time stamp). Note, that a Camera class was not defined here (see prev.
# script). Specify the required information in the camera (manually, not
# required if camera was defined beforehand)
on_list.camera.delim = "_"
on_list.camera.time_info_pos = 3
on_list.camera.time_info_str = "%Y%m%d%H%M%S%f"
off_list.camera.delim = "_"
off_list.camera.time_info_pos = 3
off_list.camera.time_info_str = "%Y%m%d%H%M%S%f"

on_list.link_imglist(off_list)

# ## Load dark and offset images and set them in the on-band image list
dark_img = pyplis.image.Img(DARK_FILE,
                            import_method=cam.image_import_method)
offset_img = pyplis.image.Img(OFFSET_FILE,
                              import_method=cam.image_import_method)

on_list.add_master_dark_image(dark_img)
on_list.add_master_offset_image(offset_img)

# Go to image number 100 in on-band list
on_list.goto_img(100)

# ## Change image preparation settings (these are applied on image load)

# region of interest
on_list.roi_abs = [100, 100, 1300, 900]

# activate cropping in image list
on_list.crop = 1

# scale down to pyramid level 2
on_list.pyrlevel = 2

# activate automatic dark correction in list
on_list.DARK_CORR_OPT = 1  # see previous script
on_list.darkcorr_mode = True

# blur the on band images
on_list.add_gaussian_blurring(3)

# get current on band image (no. 100 in list), all previously set
# preparation settings are applied to this image
on_img = on_list.current_img()

# get current off band image (no. 100 in list, index was automatically
# changed since it is linked to on band list. Note that this image is
# unedited
off_img = off_list.current_img()

fig, ax = subplots(1, 2, figsize=(18, 6))

on_img.show(ax=ax[0])
on_time_str = datetime.strftime(on_img.start_acq, "%Y-%m-%d %H:%M:%S")
```

(continues on next page)

(continued from previous page)

```
    ax[0].set_title("Current img (on-band list): %s" % on_time_str)

    # show the current off band image as well (this image is unedited)
    off_img.show(ax=ax[1])
    off_time_str = datetime.strftime(off_img.start_acq, "%Y-%m-%d %H:%M:%S")
    ax[1].set_title("Current img (off-band list): %s" % off_time_str)

    on_list.edit_info()

    on_list.skip_files = 3
    on_list.goto_img(15)

    # ## IMPORTANT STUFF FINISHED
    if SAVEFIGS:
        fig.savefig(join(SAVE_DIR, "ex0_3_out_1.%s" % FORMAT),
                    format=FORMAT, dpi=DPI)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt

        npt.assert_array_equal([501, 1, 500, 0],
                               [on_list.nof + off_list.nof,
                                on_list.this.edit_log["darkcorr"],
                                sum(on_list.this.shape),
                                on_list.gaussian_blurring -
                                on_list.this.edit_log["blurring"]])

        npt.assert_allclose([402.66284],
                            [off_img.mean() - on_img.mean()],
                            rtol=1e-7, atol=0)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except Exception:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.1.4 Example 0.4 - Introduction into the Dataset class

Automatic image type separation using the `Dataset` object and the ECII camera standard.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
```

(continues on next page)

(continued from previous page)

```
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis introduction script 4 - Automatic creation of image lists.

The previous script gave an introduction into the manual creation of
``ImgList`` objects and some basic image preparation features.
In this script, a number of ImgList objects (on, off, dark low gain,
dark high gain, offset low gain, offset high gain) is created automatically
using the Camera class created in example script ex0_2_camera_setup.py (ECII
camera)

Based on the information stored in the Camera class, a MeasSetup class is
created. The latter class collects all meta information relevant for an
emission rate analysis. Apart from the camera specs, this may include source
definitions  contains information about the camera specs a the
image base directory (note that in this example, start / stop acq. time stamps
are ignored, i.e. all images available in the specified directory are imported)
"""
from __future__ import (absolute_import, division)

# Imports from SETTINGS.py
from SETTINGS import check_version, IMG_DIR, OPTPARSE

import pyplis
from os.path import basename

# ## IMPORTS FROM OTHER EXAMPLE SCRIPTS
from ex0_2_camera_setup import create_ecII_cam_new_filters

# Check script version
check_version()


if __name__ == "__main__":
    # create the camera which was
    cam = create_ecII_cam_new_filters()

    # now throw all this stuff into the BaseSetup objec
    stp = pyplis.setupclasses.MeasSetup(IMG_DIR, camera=cam)

    # Create a Dataset which creates separate ImgLists for all types (dark,
    # offset, etc.)
    ds = pyplis.dataset.Dataset(stp)

    # The image lists can be accessed in different ways for instance using
    # the method "all_lists", which returns a Python list containing all
    # ImgList objects that were created within the Dataset
```

(continues on next page)

(continued from previous page)

```
    all_imglists = ds.all_lists()

    # print some information about each of the lists
    for lst in all_imglists:
        print("list_id: %s, list_type: %s, number_of_files: %s"
              % (lst.list_id, lst.list_type, lst.nof))

    # single lists can be accessed using "get_list(<id>)" using a valid ID,
    # e.g.:
    on_list = ds.get_list("on")
    off_list = ds.get_list("off")

    on_list.goto_img(50)  # this also changes the index in the off band list...

    # ... because it is linked to the on band list (automatically set in
    # Dataset)
    print("\nImgLists linked to ImgList on: %s" % on_list.linked_lists.keys())
    print("Current file number on / off list: %d / %d\n" % (on_list.cfn,
                                                             off_list.cfn))

    # Detected dark and offset image lists are also automatically linked to the
    # on and off band image list, such that dark image correction can be
    # applied
    on_list.darkcorr_mode = True
    off_list.darkcorr_mode = True

    # the current image preparation settings can be accessed via the
    # edit_info method
    on_list.edit_info()

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt

        npt.assert_array_equal([501, 2, 2368, 0, 50],
                               [on_list.nof + off_list.nof,
                                on_list.this.is_darkcorr +
                                off_list.this.is_darkcorr,
                                sum(on_list.this.shape),
                                on_list.gaussian_blurring -
                                on_list.this.edit_log["blurring"],
                                on_list.cfn])

        npt.assert_allclose(actual=[on_list.get_dark_image().mean()],
                            desired=[190.56119],
                            rtol=1e-7)

        print("All tests passed in script: %s" % basename(__file__))
```

4.1.5 Example 0.5 - Optical flow live view

Live view of optical flow calculation using `OpticalFlowFarneback` object (requires webcam).

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis introduction script 5 - Optical flow Farneback liveview using webcam.

Create an OpticalFlowFarneback object and activate live view (requires webcam)
"""
from __future__ import (absolute_import, division)

# Check script version
from SETTINGS import check_version

import pyplis

check_version()

flow = pyplis.plumespeed.OptflowFarneback()
flow.live_example()
```

4.1.6 Example 0.6 - Plume cross section lines

Creation and orientation of `LineOnImage` objects for emission rate retrievals.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
```

(continues on next page)

(continued from previous page)

```
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis introduction script no 6 - LineOnImage objects and their orientation.

This script introduces how to create LineOnImage objects within the image plane
and specify their orientation (i.e. the direction into which the normal vector
of the line points). This is mainly important for emission rate retrievals,
where, for instance velcoity displacement vectors (e.g. from an optical flow
algorithm) have to be multiplied with the normal vector of such a line (using
the dot product).
"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version, SAVEFIGS, SAVE_DIR, FORMAT, DPI, OPTPARSE

from pyplis import LineOnImage
from os.path import join, basename
from matplotlib.pyplot import show, subplots, close
from matplotlib.cm import get_cmap

# Check script version
check_version()


def create_example_lines():
    """Create some exemplary lines."""
    lines_r = []  # lines with orientation "right" are stored within this list
    lines_l = []  # lines with orientation "left" are stored within this list

    cmap = get_cmap("jet")
    # horizontal line, normal orientation to the top (0 deg)
    lines_r.append(LineOnImage(x0=10, y0=10, x1=90, y1=10,
                               normal_orientation="right",
                               color=cmap(0),
                               line_id="Line 1"))

    # horizontal line, normal orientation to the bottom (180 deg)
    lines_l.append(LineOnImage(10, 10, 90, 10, normal_orientation="left",
                               color=cmap(0),
                               line_id="Line 1"))

    # Vertical line normal to the right (90 deg)
    lines_r.append(LineOnImage(10, 10, 10, 90, normal_orientation="right",
                               color=cmap(50),
                               line_id="Line 2"))

    # Vertical line normal to the left (270 deg)
    lines_l.append(LineOnImage(10, 10, 10, 90, normal_orientation="left",
                               color=cmap(50),
                               line_id="Line 2"))

    # Slanted line 45 degrees
    lines_r.append(LineOnImage(20, 30, 50, 60, normal_orientation="right",
                               color=cmap(100),
```

(continues on next page)

(continued from previous page)

```
                                line_id="Line 3"))

    # Slanted line 45 degrees
    lines_l.append(LineOnImage(20, 30, 50, 60, normal_orientation="left",
                                color=cmap(100),
                                line_id="Line 3"))

    # Slanted line 45 degrees
    lines_r.append(LineOnImage(90, 10, 10, 90, normal_orientation="right",
                                color=cmap(150),
                                line_id="Line 4"))

    # Slanted line 45 degrees
    lines_l.append(LineOnImage(90, 10, 10, 90, normal_orientation="left",
                                color=cmap(150),
                                line_id="Line 4"))

    lines_r.append(LineOnImage(60, 20, 80, 90, normal_orientation="right",
                                color=cmap(200),
                                line_id="Line 5"))

    lines_l.append(LineOnImage(60, 20, 80, 90, normal_orientation="left",
                                color=cmap(200),
                                line_id="Line 5"))

    lines_r.append(LineOnImage(40, 20, 30, 90, normal_orientation="right",
                                color=cmap(250),
                                line_id="Line 6"))
    lines_l.append(LineOnImage(40, 20, 30, 90, normal_orientation="left",
                                color=cmap(250),
                                line_id="Line 6"))

    return lines_r, lines_l


if __name__ == "__main__":
    close("all")
    fig, ax = subplots(1, 2, figsize=(18, 9))

    lines_r, lines_l = create_example_lines()

    for k in range(len(lines_r)):
        line = lines_r[k]
        # print "%d: %s" %(k, line.orientation_info)
        normal = line.normal_vector
        lbl = "%s" % line.line_id
        line.plot_line_on_grid(ax=ax[0], include_normal=1,
                                include_roi_rot=True, label=lbl)
    for k in range(len(lines_l)):
        line = lines_l[k]
        normal = line.normal_vector
        lbl = "%s" % line.line_id
        line.plot_line_on_grid(ax=ax[1], include_normal=1,
                                include_roi_rot=True, label=lbl)

    ax[0].set_title("Orientation right")
    ax[0].legend(loc="best", fontsize=8, framealpha=0.5)
```

(continues on next page)

(continued from previous page)

```
    ax[0].set_xlim([0, 100])
    ax[0].set_ylim([100, 0])

    ax[1].set_title("Orientation left")
    ax[1].legend(loc="best", fontsize=8, framealpha=0.5)
    ax[1].set_xlim([0, 100])
    ax[1].set_ylim([100, 0])
    # ## IMPORTANT STUFF FINISHED
    if SAVEFIGS:
        fig.savefig(join(SAVE_DIR, "ex0_6_out_1.%s" % FORMAT),
                    format=FORMAT, dpi=DPI)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt

        npt.assert_array_equal([sum([x.length() for x in lines_l]),
                                sum([x.length() for x in lines_r]),
                                sum([sum(x.coords) for x in lines_r]),
                                int(sum([sum(x.normal_vector + y.normal_vector)
                                    for x, y in zip(lines_l, lines_r)])),
                                int(sum([sum(x.center_pix) for x in lines_l])),
                                ],
                               [459, 459, 1030, 0, 515])

        npt.assert_allclose(
            actual=[sum([sum(x) for x in lines_l[2].rect_roi_rot])],
            desired=[368.79393923],
            rtol=1e-7)

        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except Exception:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.1.7 Example 0.7 - Manual cell calibration

Manual cell calibration based on a set of background and cell images (on / off).

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
```

(continues on next page)

(continued from previous page)

```
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis introduction script no. 5: manual cell calibration.

Perform manual cell calibration based on 3 cell images and one background
image. The calibration data consists of 3 cells which were put in front of the
lense successively and a background images both before and after the
cell images.

This script creates an empty CellCalibEngine object in which 6 cell images
are assigned (3 on and 3 off band) with their corresponding SO2 column
densities. Further, 2 background images are assigned for each filter, one
before and one after the cells were put in front of the camera. These are used
to determine tau images from the cell images. Variations in the background
intensities are corrected for (for details see manuscript).

Note, that this calibration does not include a dark correction of the images
before the calibration, therefore, the results are slightly different compared
to the results from ex05_cell_calib_auto.py.
"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version

import pyplis
from os.path import join, basename
from matplotlib.pyplot import close, show
from time import time

# IMPORT GLOBAL SETTINGS
from SETTINGS import SAVEFIGS, SAVE_DIR, FORMAT, DPI, OPTPARSE, IMG_DIR

# Check script version
check_version()

# SPECIFY IMAGE PATHS FOR EACH CELL AND BACKGROUND IMAGES
BG_BEFORE_ON = "EC2_1106307_1R02_2015091607000845_F01_Etna.fts"
BG_BEFORE_OFF = "EC2_1106307_1R02_2015091607001021_F02_Etna.fts"

A53_ON = "EC2_1106307_1R02_2015091607003032_F01_Etna.fts"
A53_OFF = "EC2_1106307_1R02_2015091607003216_F02_Etna.fts"

A37_ON = "EC2_1106307_1R02_2015091607005847_F01_Etna.fts"
A37_OFF = "EC2_1106307_1R02_2015091607010023_F02_Etna.fts"

A57_ON = "EC2_1106307_1R02_2015091607013835_F01_Etna.fts"
A57_OFF = "EC2_1106307_1R02_2015091607014019_F02_Etna.fts"
```

(continues on next page)

(continued from previous page)

```
BG_AFTER_ON = "EC2_1106307_1R02_2015091607020256_F01_Etna.fts"
BG_AFTER_OFF = "EC2_1106307_1R02_2015091607020440_F02_Etna.fts"

# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    close("all")
    start = time()
    # define camera for ECII custom image import
    cam = pyplis.Camera("ecII")
    cellcalib = pyplis.cellcalib.CellCalibEngine(cam)

    # now add all cell images manually, specifying paths, the SO2 column
    # densities of each cell, and the corresponding cell ID as well as image
    # type (on, off)
    cellcalib.set_cell_images(img_paths=join(IMG_DIR, A53_ON),
                              cell_gas_cd=4.15e17,
                              cell_id="a53", filter_id="on")

    cellcalib.set_cell_images(img_paths=join(IMG_DIR, A53_OFF),
                              cell_gas_cd=4.15e17,
                              cell_id="a53", filter_id="off")

    cellcalib.set_cell_images(img_paths=join(IMG_DIR, A37_ON),
                              cell_gas_cd=8.59e17,
                              cell_id="a37", filter_id="on")

    cellcalib.set_cell_images(img_paths=join(IMG_DIR, A37_OFF),
                              cell_gas_cd=8.59e17,
                              cell_id="a37", filter_id="off")

    cellcalib.set_cell_images(img_paths=join(IMG_DIR, A57_ON),
                              cell_gas_cd=1.92e18,
                              cell_id="a57", filter_id="on")

    cellcalib.set_cell_images(img_paths=join(IMG_DIR, A57_OFF),
                              cell_gas_cd=1.92e18,
                              cell_id="a57", filter_id="off")

    # put the onband background images into a Python list ....
    bg_on_paths = [join(IMG_DIR, BG_BEFORE_ON), join(IMG_DIR, BG_AFTER_ON)]

    # ... and add them to the calibration object ...
    cellcalib.set_bg_images(img_paths=bg_on_paths, filter_id="on")

    # ... same for off band background images
    bg_off_paths = [join(IMG_DIR, BG_BEFORE_OFF), join(IMG_DIR, BG_AFTER_OFF)]
    cellcalib.set_bg_images(img_paths=bg_off_paths, filter_id="off")

    # Prepare calibration data (i.e. CellCalibData objets) for on, off and
    # AA images. This function determines tau images for each cell using
    # background images scaled to the present background intensities at the
    # acq. time stamp of each cell using temporal interpolation of the provided
    # background images. This results in 3 tau images for each filter (on, off)
    # and from that, 3 AA images are determined. Each of these collection of
    # 3 tau images (on, off, AA) is then stored within a CellCalibData object
    # which can be accessed using e.g. cellcalib.calib_data["aa"]
    cellcalib.prepare_calib_data(on_id="on", off_id="off", darkcorr=False)
```

(continues on next page)

(continued from previous page)

```
    stop = time()

    ax = cellcalib.plot_all_calib_curves()
    ax.set_title("Manual cell calibration\n(NO dark correction performed)")

    aa_calib = cellcalib.calib_data["aa"]

    print("Time elapsed for preparing calibration data: %.4f s"
          % (stop - start))
    # ## IMPORTANT STUFF FINISHED
    if SAVEFIGS:
        ax.figure.savefig(join(SAVE_DIR, "ex0_7_out_1.%s" % FORMAT),
                          format=FORMAT, dpi=DPI)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt

        slope, offs = aa_calib.calib_coeffs
        actual = [cellcalib.tau_stacks["aa"].sum(),
                  cellcalib.tau_stacks["aa"].mean(),
                  aa_calib.cd_vec.sum(),
                  slope,
                  offs]

        npt.assert_allclose(actual=actual,
                            desired=[1007480.56,
                                     0.24401481,
                                     3.194000052267778e+18,
                                     4.779782684462987e+18,
                                     -2.7244424951657216e+16],
                            rtol=1e-7)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.2 Examples for emission rate analysis

The following scripts are directly related for emission rate analysis and build on top of each other ending with a full emission rate analysis in *Example 12 - Emission rate analysis (Etna example data)*.

4.2.1 Example 1 - Creation of analysis setup and Dataset

This script introduces the `MeasSetup` object and how it can be used to specify all relevant information to create a `Dataset` for emission rate analysis.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliß (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script no. 1 - Analysis setup for example data set.

In this script an example data set, recorded on the 16/9/15 7:06-7:22 at
Mt. Etna is setup. Most of the following example scripts will use the
information specified here.

A typical analysis setup for plume image data contains information about the
camera (e.g. optics, file naming convention, see also ex0_2_camera_setup.py),
gas source, the measurement geometry and the wind conditions. These information
generally needs to be specified by the user before the analysis. The
information is stored within a MeasSetup object which can be used as a basis
for further analysis.
If not all neccessary information is provided, a MeasSetup object will be
created nonetheless but analysis options might be limited.

Such a MeasSetup object can be used as input for Dataset objects which
creates the analysis environment. The main purpose of Dataset classes
is to automatically separate images by their type (e.g. on / off-band
images, dark, offset) and create ImgList classes from that. ImgList
classes typically contain images of one type. The Dataset also links
relevant ImgList to each other, e.g. if the camera has an off-band
filter, and off-band images can be found in the specified directory,
then it is linked to the list containing on-band images. This means,
that the current image index in the off-band list is automatically
updated whenever the index is changed in the on-band list. If
acquisition time is available in the images, then the index is updated
based on the closest acq. time of the off-band images, based on the
current on-band acq. time. If not, it is updated by index (e.g. on-band
contains 100 images and off-band list 50. The current image number in
the on-band list is 50, then the off-band index is set to 25). Also,
dark and offset images lists are linked both to on and off-band image
lists, such that the image can be correccted for dark and offset
automatically on image load (the latter needs to be activated in the
lists using ``darkcorr_mode=True``).
```

(continues on next page)

(continued from previous page)

```
This script shows how to setup a MeasSetup object and create a Dataset object
from it. As example, the first image of the on-band image time series is
displayed.

The Dataset object created here is used in script  ex04_prep_aa_imglist.py
which shows how to create an image list displaying AA images.
"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version, IMG_DIR, OPTPARSE
import pyplis as pyplis
from datetime import datetime
from matplotlib.pyplot import show, close

# Check script version
check_version()

# SCRIPT FUNCTION DEFINITIONS


def create_dataset():
    """Initialize measurement setup and creates dataset from that."""
    start = datetime(2015, 9, 16, 7, 6, 00)
    stop = datetime(2015, 9, 16, 7, 22, 00)
    # Define camera (here the default ecII type is used)
    cam_id = "ecII"

    # the camera filter setup
    filters = [pyplis.utils.Filter(type="on", acronym="F01"),
               pyplis.utils.Filter(type="off", acronym="F02")]

    # camera location and viewing direction (altitude will be retrieved
    # automatically)
    geom_cam = {"lon": 15.1129,
                "lat": 37.73122,
                "elev": 20.0,
                "elev_err": 5.0,
                "azim": 270.0,
                "azim_err": 10.0,
                "alt_offset": 15.0,
                "focal_length": 25e-3}  # altitude offset (above topography)

    # Create camera setup
    # the camera setup includes information about the filename convention in
    # order to identify different image types (e.g. on band, off band, dark..)
    # it furthermore includes information about the detector specifics (e.g.
    # dimension, pixel size, focal length). Measurement specific parameters
    # (e.g. lon, lat, elev, azim) where defined in the dictinary above and
    # can be passed as additional keyword dictionary using **geom_cam
    # Alternatively, they could also be passed directly, e.g.:

    # cam = pyplis.setup.Camera(cam_id, filter_list=filters, lon=15.1129,
    #                           lat=37.73122)

    cam = pyplis.setupclasses.Camera(cam_id, filter_list=filters,
                                     **geom_cam)
```

(continues on next page)

(continued from previous page)

```
    # Load default information for Etna. This information is stored in
    # the source_info.txt file of the Pyplis information. You may also access
    # information about any volcano via the available online access to the NOAA
    # database using the method pyplis.inout.get_source_info_online(source_id).

    source = pyplis.setupclasses.Source("etna")

    # Provide wind direction
    wind_info = {"dir": 0.0,
                 "dir_err": 1.0}

#                "dir_err"  : 15.0}

    # Create BaseSetup object (which creates the MeasGeometry object)
    stp = pyplis.setupclasses.MeasSetup(IMG_DIR, start, stop, camera=cam,
                                        source=source,
                                        wind_info=wind_info)
    print(stp.LINK_OFF_TO_ON)
    # Create analysis object (from BaseSetup)
    # The dataset takes care of finding all vali
    return pyplis.Dataset(stp)


# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    close("all")
    ds = create_dataset()

    # get on-band image list
    on_list = ds.get_list("on")
    on_list.goto_next()
    off_list = ds.get_list("off")

    # activate dark correction in both lists. Dark and offset image lists are
    # automatically assigned to plume on and off-band image lists on initiation
    # of the dataset object
    on_list.darkcorr_mode = True
    off_list.darkcorr_mode = True

    print("On-band list contains %d images, current image index: %d"
          % (on_list.nof, on_list.cfn))

    img = on_list.current_img()

    # plume distance image retrieved from MeasGeometry class...
    plume_dists = on_list.plume_dists

    # ...these may be overwritten or set manually if desired
    on_list.plume_dists = 10000

    # The same applies for the integration step lengths for emission rate
    # retrievals
    step_lengths = on_list.integration_step_length
    on_list.integration_step_length = 1.8  # m

    img_shift = img.duplicate()
```

(continues on next page)

(continued from previous page)

```
# images can be shifted using the scipy.ndimage.interpolation.shift method
# this may be required for image registration in dual camera systems.
# Whether this is supposed to be done automatically can be specified using
# the REG_SHIFT_OFF option in a MeasSetup class. It may also be specified
# directly for your cam in the custom camera definition file cam_info.txt
# using io_opts:REG_SHIFT_OFF=1 (see e.g. defintion of camera with ID
# "usgs"). Also, a default registration offset can be defined here using
#
img_shift.shift(dx_abs=-30, dy_abs=55)
img_shift.show(tit="Shifted")
# Set pixel intensities below 2000 to 0 (method of Img class)
img.set_val_below_thresh(val=0, threshold=2000)
# show modified image
img.show()
print(str(img))  # image object has an informative string representation

# IMPORTANT STUFF FINISHED (Below follow tests and display options)

# Import script options
(options, args) = OPTPARSE.parse_args()

# If applicable, do some tests. This is done only if TESTMODE is active:
# testmode can be activated globally (see SETTINGS.py) or can also be
# activated from the command line when executing the script using the
# option --test 1
if int(options.test):
    import numpy.testing as npt
    from os.path import basename

    actual = [plume_dists.mean(), plume_dists.std(),
              on_list.get_dark_image().mean()]
    npt.assert_allclose(actual=actual,
                        desired=[10909.873427010458, 221.48844132471388,
                                 190.56119],
                        rtol=1e-7)

    npt.assert_array_equal([418, 2, 2368, 1, 1, 0,
                            20150916070600,
                            20150916072200],
                           [on_list.nof + off_list.nof,
                            on_list.this.is_darkcorr +
                            off_list.this.is_darkcorr,
                            sum(on_list.this.shape),
                            on_list.cfn,
                            off_list.cfn,
                            sum(img.img[img.img < 2000]),
                            int(ds.setup.start.strftime("%Y%m%d%H%M%S")),
                            int(ds.setup.stop.strftime("%Y%m%d%H%M%S"))])

    print("All tests passed in script: %s" % basename(__file__))
try:
    if int(options.show) == 1:
        show()
except BaseException:
    print("Use option --show 1 if you want the plots to be displayed")
```

4.2.2 Example 2 - Measurement Geometry

This script introduces the `MeasGeometry` object including some basic features.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script no. 2 - Features of the MeasGeometry class.

In this script, some important features of the MeasGeometry class are
introduced. The class itself is automatically created in the MeasSetup
object which was created in example script ex01_analysis_setup.py and was
passed as input for a Dataset object. The relevant MeasGeometry class is stored
within the Dataset object and can be accessed via the ``meas_geometry``
attribute.

As a first feature, the viewing direction of the camera is retrieved from the
image using the position of the south east (SE) crater of Mt.Etna. The result
is then visualized in a 2D map to give an overview of the geometry. The map
further includes the initial viewing direction (see example script
ex01_analysis_setup.py) which was logged in the field using a compass and a
mechanical inclinometer.

Further, the distance to the plume is retrieved on a pixel basis (represented
as image).
"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version

from geonum import GeoPoint
from matplotlib.pyplot import subplots, show, close
from os.path import join

# IMPORT GLOBAL SETTINGS
from SETTINGS import SAVEFIGS, SAVE_DIR, FORMAT, DPI, OPTPARSE

# IMPORTS FROM OTHER EXAMPLE SCRIPTS
from ex01_analysis_setup import create_dataset

# Check script version
check_version()
```

(continues on next page)

(continued from previous page)

```
# SCRIPT FUNCTION DEFINITIONS


def find_viewing_direction(meas_geometry, draw_result=True):
    """Correct viewing direction using location of Etna SE crater.

    Defines location of Etna SE crater within images (is plotted into current
    plume onband image of dataset) and uses its geo location to retrieve the
    camera viewing direction

    :param meas_geometry: :class:`MeasGeometry` object

    """
    # Position of SE crater in the image (x, y)
    se_crater_img_pos = [806, 736]

    # Geographic position of SE crater (extracted from Google Earth)
    # The GeoPoint object (geonum library) automatically retrieves the altitude
    # using SRTM data
    se_crater = GeoPoint(37.747757, 15.002643, name="SE crater")

    print("Retrieved altitude SE crater (SRTM): %s" % se_crater.altitude)

    # The following method finds the camera viewing direction based on the
    # position of the south east crater.
    new_elev, new_azim, _, basemap =\
        meas_geometry.find_viewing_direction(pix_x=se_crater_img_pos[0],
                                             pix_y=se_crater_img_pos[1],
                                             # for uncertainty estimate
                                             pix_pos_err=100,
                                             geo_point=se_crater,
                                             draw_result=draw_result,
                                             update=True)  # overwrite settings

    print("Updated camera azimuth and elevation in MeasGeometry, new values: "
          "elev = %.1f, azim = %.1f" % (new_elev, new_azim))

    return meas_geometry, basemap


def plot_plume_distance_image(meas_geometry):
    """Determine and plot image plume distance and pix-to-pix distance imgs."""
    # This function returns three images, the first corresponding to pix-to-pix
    # distances in horizontal direction and the second (ignored here) to
    # the vertical (in this case they are the same since pixel height and
    # pixel width are the same for this detector). The third image gives
    # camera to plume distances on a pixel basis
    (dist_img, _, plume_dist_img) =\
        meas_geometry.compute_all_integration_step_lengths()

    fig, ax = subplots(2, 1, figsize=(7, 8))

    # Show pix-to-pix distance image
    dist_img.show(cmap="gray", ax=ax[0], zlabel="Pix-to-pix distance [m]")
    ax[0].set_title("Parameterised pix-to-pix dists")
```

(continues on next page)

(continued from previous page)

```
    # Show plume distance image (convert pixel values to from m -> km)
    (plume_dist_img / 1000.0).show(cmap="gray",
                                   ax=ax[1], zlabel="Plume distance [km]")
    ax[1].set_title("Plume dists")
    return fig


# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    close("all")

    # Create the Dataset object (see ex01)
    ds = create_dataset()

    # execute function defined above (see above for definition and information)
    geom_corr, map_ = find_viewing_direction(ds.meas_geometry)

    # execute 2. script function (see above for definition and information)
    fig = plot_plume_distance_image(ds.meas_geometry)

    # You can compute the plume distance for the camera CFOV pixel column just
    # by calling the method plume_dist() without specifying the input azimuth
    # angle...
    plume_dist_cfov = geom_corr.plume_dist()[0][0]

    # ... and the corresponding uncertainty
    plume_dist_err_cfov = geom_corr.plume_dist_err()

    # You can also retrieve an array containing the camera azimuth angles for
    # each pixel column...
    all_azimuths = geom_corr.all_azimuths_camfov()

    # ... and use this to compute plume distances on a pixel column basis
    plume_dists_all_cols = geom_corr.plume_dist(all_azimuths)

    # If you want, you can update information about camera, source or
    # meteorolgy using either of the following methods (below we apply a
    # change in the wind-direction such that the plume propagation direction
    # is changed from S to SE )
    # geom_corr.update_cam_specs()
    # geom_corr.update_source_specs()

    geom_corr.update_wind_specs(dir=315)
    geom_corr.draw_map_2d()  # this figure is only displayed and not saved

    # recompute plume distance of CFOV pixel
    plume_dist_cfov_new = geom_corr.plume_dist()[0][0]

    print("Comparison of plume distances after change of wind direction:\n"
          "Previous: %.3f m\n"
          "New: %.3f m" % (plume_dist_cfov, plume_dist_cfov_new))
    # Using this a
    # IMPORTANT STUFF FINISHED
    if SAVEFIGS:
        map_.ax.figure.savefig(join(SAVE_DIR, "ex02_out_1.%s" % FORMAT),
                               format=FORMAT, dpi=DPI)
        fig.savefig(join(SAVE_DIR, "ex02_out_2.%s" % FORMAT), format=FORMAT,
```

(continues on next page)

(continued from previous page)

```
                    dpi=DPI)

    # IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt
        from os.path import basename

        npt.assert_array_equal([],
                               [])

        # check some propoerties of the basemap (displayed in figure)

        # map basemap coordinates to lon / lat values
        lon, lat = map_(8335, 9392, inverse=True)
        npt.assert_allclose(actual=[lon, lat, map_.delta_lon, map_.delta_lat],
                            desired=[15.075131135, 37.76678834,
                                     0.149263852982,
                                     0.118462659944],
                            rtol=1e-7)

        # check some basic properties / values of the geometry
        npt.assert_allclose(actual=[geom_corr.cam_elev,
                                    geom_corr.cam_elev_err,
                                    geom_corr.cam_azim,
                                    geom_corr.cam_azim_err,
                                    plume_dist_cfov,
                                    plume_dist_err_cfov,
                                    plume_dists_all_cols.mean(),
                                    plume_dist_cfov_new],
                            desired=[1.547754e+01,
                                     1.064556e+00,
                                     2.793013e+02,
                                     1.065411e+00,
                                     1.073102e+04,
                                     1.645586e+02,
                                     1.076047e+04,
                                     9.961593e+03],
                            rtol=1e-6)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.2.3 Example 3 - Plume background analysis

Introduction into `PlumeBackgroundModel` object the default modes for retrieval of plume background intensities and plume optical density images.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliß (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script no. 3 - Plume background analysis.

This example script introduces features related to plume background modelling
and tau image calculations.
"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version

import numpy as np
from os.path import join
import pyplis
from matplotlib.pyplot import show, subplots, close

# IMPORT GLOBAL SETTINGS
from SETTINGS import SAVEFIGS, SAVE_DIR, FORMAT, DPI, IMG_DIR, OPTPARSE

# Check script version
check_version()

# SCRIPT OPTIONS

# If this is True, then sky reference areas are set in auto mode (note that
# in this case, the tests at the end of the script will fail!)
USE_AUTO_SETTINGS = False

# intensity threshold to init mask for bg surface fit
POLYFIT_2D_MASK_THRESH = 2600

# Choose the background correction modes you want to use

BG_CORR_MODES = [0,  # 2D poly surface fit (without sky radiance image)
                 1,  # Scaling of sky radiance image
                 4,  # Scaling + linear gradient correction in x & y direction
```

(continues on next page)

(continued from previous page)

```
                  6]  # Scaling + quadr. gradient correction in x & y direction


# Image file paths relevant for this script
PLUME_FILE = join(IMG_DIR, 'EC2_1106307_1R02_2015091607065477_F01_Etna.fts')
BG_FILE = join(IMG_DIR, 'EC2_1106307_1R02_2015091607022602_F01_Etna.fts')
OFFSET_FILE = join(IMG_DIR, 'EC2_1106307_1R02_2015091607064723_D0L_Etna.fts')
DARK_FILE = join(IMG_DIR, 'EC2_1106307_1R02_2015091607064865_D1L_Etna.fts')

# SCRIPT FUNCTION DEFINITIONS


def init_background_model():
    """Create background model and define relevant sky reference areas."""
    # Create background modelling object
    m = pyplis.plumebackground.PlumeBackgroundModel()

    # Define default gas free areas in plume image
    w, h = 40, 40  # width/height of rectangles

    m.scale_rect = [1280, 20, 1280 + w, 20 + h]
    m.xgrad_rect = [20, 20, 20 + w, 20 + h]
    m.ygrad_rect = [1280, 660, 1280 + w, 660 + h]

    # Define coordinates of horizontal and vertical profile lines

    # row number of profile line for horizontal corretions in the sky
    # gradient...
    m.xgrad_line_rownum = 40
    # ... and start / stop columns for the corrections
    m.xgrad_line_startcol = 20
    m.xgrad_line_stopcol = 1323

    # col number of profile line for vertical corretions in the sky gradient...
    m.ygrad_line_colnum = 1300
    # ... and start / stop rows for the corrections
    m.ygrad_line_startrow = 10
    m.ygrad_line_stoprow = 700
    # Order of polyonmial fit applied for the gradient correction
    m.ygrad_line_polyorder = 2

    return m


def load_and_prepare_images():
    """Load images defined above and prepare them for the background analysis.

    Returns
    --------
        - Img, plume image
        - Img, plume image vignetting corrected
        - Img, sky radiance image

    """
    # get custom load method for ECII
    fun = pyplis.custom_image_import.load_ecII_fits
    # Load the image objects and peform dark correction
```

(continues on next page)

(continued from previous page)

```
    plume, bg = pyplis.Img(PLUME_FILE, fun), pyplis.Img(BG_FILE, fun)
    dark, offset = pyplis.Img(DARK_FILE, fun), pyplis.Img(OFFSET_FILE, fun)

    # Model dark image for tExp of plume image
    dark_plume = pyplis.image.model_dark_image(plume.meta["texp"],
                                               dark, offset)
    # Model dark image for tExp of background image
    dark_bg = pyplis.image.model_dark_image(bg.meta["texp"],
                                            dark, offset)

    plume.subtract_dark_image(dark_plume)
    bg.subtract_dark_image(dark_bg)
    # Blur the images (sigma = 1)
    plume.add_gaussian_blurring(1)
    bg.add_gaussian_blurring(1)

    # Create vignetting correction mask from background image
    vign = bg.img / bg.img.max()  # NOTE: potentially includes y & x gradients
    plume_vigncorr = pyplis.Img(plume.img / vign)
    return plume, plume_vigncorr, bg


def autosettings_vs_manual_settings(bg_model):
    """Perform automatic retrieval of sky reference areas.

    If you are lazy... (i.e. you dont want to define all these reference areas)
    then you could also use the auto search function, a comparison is plotted
    here.
    """
    auto_params = pyplis.plumebackground.find_sky_reference_areas(plume)
    current_params = bg_model.settings_dict()

    fig, axes = subplots(1, 2, figsize=(16, 6))
    axes[0].set_title("Manually set parameters")
    pyplis.plumebackground.plot_sky_reference_areas(plume, current_params,
                                                    ax=axes[0])
    pyplis.plumebackground.plot_sky_reference_areas(plume, auto_params,
                                                    ax=axes[1])
    axes[1].set_title("Automatically set parameters")

    return auto_params, fig


def plot_pcs_profiles_4_tau_images(tau0, tau1, tau2, tau3, pcs_line):
    """Plot PCS profiles for all 4 methods."""
    fig, ax = subplots(1, 1)
    tau_imgs = [tau0, tau1, tau2, tau3]

    for k in range(4):
        img = tau_imgs[k]
        profile = pcs_line.get_line_profile(img)
        ax.plot(profile, "-", label=r"Mode %d: $\phi=%.3f$"
                % (BG_CORR_MODES[k], np.mean(profile)))

    ax.grid()
    ax.set_ylabel(r"$\tau_{on}$", fontsize=20)
    ax.set_xlim([0, pcs_line.length()])
```

(continues on next page)

(continued from previous page)

```
    ax.set_xticklabels([])
    ax.set_xlabel("PCS", fontsize=16)
    ax.legend(loc="best", fancybox=True, framealpha=0.5, fontsize=12)
    return fig


# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    close("all")

    # Create a background model with relevant sky reference areas
    bg_model = init_background_model()

    # Define exemplary plume cross section line
    pcs_line = pyplis.LineOnImage(x0=530,
                                  y0=730,
                                  x1=890,
                                  y1=300,
                                  line_id="example PCS",
                                  color="lime")

    plume, plume_vigncorr, bg = load_and_prepare_images()

    auto_params, fig0 = autosettings_vs_manual_settings(bg_model)

    # Script option
    if USE_AUTO_SETTINGS:
        bg_model.update(**auto_params)

    # Model 4 exemplary tau images

    # list to store figures of tau plotted tau images
    _tau_figs = []

    # mask for corr mode 0 (i.e. 2D polyfit)
    mask = np.ones(plume_vigncorr.img.shape, dtype=np.float32)
    mask[plume_vigncorr.img < POLYFIT_2D_MASK_THRESH] = 0

    # First method: retrieve tau image using poly surface fit
    tau0 = bg_model.get_tau_image(plume_vigncorr,
                                  mode=BG_CORR_MODES[0],
                                  surface_fit_mask=mask,
                                  surface_fit_polyorder=1)

    # Plot the result and append the figure to _tau_figs
    _tau_figs.append(bg_model.plot_tau_result(tau0, PCS=pcs_line))

    # Second method: scale background image to plume image in "scale" rect
    tau1 = bg_model.get_tau_image(plume, bg, mode=BG_CORR_MODES[1])
    _tau_figs.append(bg_model.plot_tau_result(tau1, PCS=pcs_line))

    # Third method: Linear correction for radiance differences based on two
    # rectangles (scale, ygrad)
    tau2 = bg_model.get_tau_image(plume, bg, mode=BG_CORR_MODES[2])
    _tau_figs.append(bg_model.plot_tau_result(tau2, PCS=pcs_line))

    # 4th method: 2nd order polynomial fit along vertical profile line
```

(continues on next page)

(continued from previous page)

```
    # For this method, determine tau on tau off and AA image
    tau3 = bg_model.get_tau_image(plume, bg, mode=BG_CORR_MODES[3])
    _tau_figs.append(bg_model.plot_tau_result(tau3, PCS=pcs_line))

    fig6 = plot_pcs_profiles_4_tau_images(tau0, tau1, tau2, tau3, pcs_line)

    if SAVEFIGS:
        fig0.savefig(join(SAVE_DIR, "ex03_out_1.%s" % FORMAT), format=FORMAT,
                     dpi=DPI, transparent=True)
        for k in range(len(_tau_figs)):
            # _tau_figs[k].suptitle("")
            _tau_figs[k].savefig(join(SAVE_DIR, "ex03_out_%d.%s"
                                      % ((k + 2), FORMAT)),
                                 format=FORMAT, dpi=DPI)

        fig6.savefig(join(SAVE_DIR, "ex03_out_6.%s" % FORMAT), format=FORMAT,
                     dpi=DPI)
    # IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt
        from os.path import basename
        m = bg_model

        # test settings for clear sky reference areas
        npt.assert_array_equal([2680, 3960, 160, 6, 1300, 10, 700, 40, 20,
                                1323, 567584],
                               [sum(m.scale_rect),
                                sum(m.ygrad_rect),
                                sum(m.xgrad_rect),
                                m.mode,
                                m.ygrad_line_colnum,
                                m.ygrad_line_startrow,
                                m.ygrad_line_stoprow,
                                m.xgrad_line_rownum,
                                m.xgrad_line_startcol,
                                m.xgrad_line_stopcol,
                                int(m.surface_fit_mask.sum())])

        m.update(**auto_params)
        # test settings for clear sky reference areas
        npt.assert_array_equal([2682, 4142, 1380, 6, 1337, 1, 790, 6, 672,
                                1343, 567584],
                               [sum(m.scale_rect),
                                sum(m.ygrad_rect),
                                sum(m.xgrad_rect),
                                m.mode,
                                m.ygrad_line_colnum,
                                m.ygrad_line_startrow,
                                m.ygrad_line_stoprow,
```

(continues on next page)

(continued from previous page)

```
                                m.xgrad_line_rownum,
                                m.xgrad_line_startcol,
                                m.xgrad_line_stopcol,
                                int(m.surface_fit_mask.sum())])

        # test all tau-modelling results
        actual = [tau0.mean(), tau1.mean(), tau2.mean(), tau3.mean()]
        npt.assert_allclose(actual=actual,
                            desired=[0.11395558008662043,
                                     0.25279653,
                                     0.13842879832119934,
                                     0.13943940574220634],
                            rtol=1e-7)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.2.4 Example 4 - Preparation of AA image list

Script showing how to prepare an `ImgList` containing on-band plume images, such that `aa_mode` can be activated (i.e. images are loaded as AA images).

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliß (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script no. 4 - Prepare AA image list (from onband list).

Script showing how to work in AA mode using ImgList object
"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version

import pyplis
from matplotlib.pyplot import close
from os.path import join
```

(continues on next page)

(continued from previous page)

```
# IMPORT GLOBAL SETTINGS
from SETTINGS import SAVEFIGS, SAVE_DIR, FORMAT, DPI, IMG_DIR, OPTPARSE, PCS1

# IMPORTS FROM OTHER EXAMPLE SCRIPTS
from ex01_analysis_setup import create_dataset
from ex02_meas_geometry import find_viewing_direction

# Check script version
check_version()


# SCRIPT FUNCTION DEFINITIONS
def prepare_aa_image_list(bg_corr_mode=6):
    """Get and prepare onband list for aa image mode.

    The relevant gas free areas for background image modelling are set
    automatically (see also ex. 3 for details)

    :return: - on list in AA mode
    """
    dataset = create_dataset()
    geom, _ = find_viewing_direction(dataset.meas_geometry, False)

    # Set plume background images for on and off
    # this is the same image which is also used for example script
    # ex03_plume_background.py demonstrating the plume background routines
    path_bg_on = join(IMG_DIR,
                      'EC2_1106307_1R02_2015091607022602_F01_Etna.fts')
    path_bg_off = join(IMG_DIR,
                       'EC2_1106307_1R02_2015091607022820_F02_Etna.fts')

    # Get on and off lists and activate dark correction
    on_lst = dataset.get_list("on")
    off_list = dataset.get_list("off")

    # Deactivate automatic reload in list while changing some list
    # attributes
    on_lst.auto_reload = False
    off_list.auto_reload = False

    on_lst.darkcorr_mode = True
    off_list.darkcorr_mode = True

    # Prepare on and offband background images
    bg_on = pyplis.Img(path_bg_on)
    bg_on.subtract_dark_image(on_lst.get_dark_image())

    bg_off = pyplis.Img(path_bg_off)
    bg_off.subtract_dark_image(off_list.get_dark_image())

    # set the background images within the lists
    on_lst.set_bg_img(bg_on)
    off_list.set_bg_img(bg_off)

    # automatically set gas free areas
    on_lst.bg_model.set_missing_ref_areas(on_lst.current_img())
```

(continues on next page)

(continued from previous page)

```
    # Now update some of the information from the automatically set sky ref
    # areas
    on_lst.bg_model.xgrad_line_startcol = 20
    on_lst.bg_model.xgrad_line_rownum = 25
    off_list.bg_model.xgrad_line_startcol = 20
    off_list.bg_model.xgrad_line_rownum = 25

    # set background modelling mode
    on_lst.bg_model.mode = bg_corr_mode
    off_list.bg_model.mode = bg_corr_mode

    on_lst.aa_mode = True  # activate AA mode

    off_list.auto_reload = True
    on_lst.auto_reload = True
    print("INITIATED AA LIST")
    on_lst.meas_geometry = geom
    return on_lst


# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    from matplotlib.pyplot import show
    from time import time

    close("all")
    aa_list = prepare_aa_image_list()

    t0 = time()
    # Deactivate auto reload while changing some settings (else, after each
    # of the following operations the images are reloaded and edited, which)
    aa_list.auto_reload = False
    aa_list.goto_img(50)

    aa_list.add_gaussian_blurring(1)
    # aa_list.pyrlevel = 2
    aa_list.roi_abs = [300, 300, 1120, 1000]
    aa_list.crop = True
    # now reactive image reload in list (loads image no. 50 with all changes
    # that where set in the previous lines)
    aa_list.auto_reload = True

    # store some results for tests below
    shape_log, mean_log = sum(aa_list.this.shape), aa_list.this.mean()

    ax = aa_list.show_current(zlabel=r"$\tau_{AA}$")
    print("Elapsed time: %s s" % (time() - t0))

    aa_list.crop = False
    ax1 = aa_list.bg_model.plot_sky_reference_areas(aa_list.current_img())
    fig = aa_list.bg_model.plot_tau_result(aa_list.current_img())

    # import plume cross section and computed integrated optical density
    # for current image
    img = aa_list.this
    ax2 = img.show(vmin=-0.1, vmax=0.3)
    pcs = PCS1.convert(to_pyrlevel=0)
```

(continues on next page)

(continued from previous page)

```
    pcs.plot_line_on_grid(ax=ax2)
    pix_steps = aa_list.meas_geometry.compute_all_integration_step_lengths()[0]
    integrated_aa = pcs.integrate_profile(img, pix_steps)

    # IMPORTANT STUFF FINISHED
    if SAVEFIGS:
        ax.figure.savefig(join(SAVE_DIR, "ex04_out_1.%s" % FORMAT),
                          format=FORMAT, dpi=DPI)

    # IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt
        from os.path import basename

        m = aa_list.bg_model
        npt.assert_array_equal([2682, 4144, 1380, 6, 1337, 1, 791, 25, 20,
                                1343],
                               [sum(m.scale_rect),
                                sum(m.ygrad_rect),
                                sum(m.xgrad_rect),
                                m.mode,
                                m.ygrad_line_colnum,
                                m.ygrad_line_startrow,
                                m.ygrad_line_stoprow,
                                m.xgrad_line_rownum,
                                m.xgrad_line_startcol,
                                m.xgrad_line_stopcol])

        actual = [aa_list.meas_geometry.cam_elev,
                  aa_list.meas_geometry.cam_azim,
                  aa_list.meas_geometry.plume_dist()[0, 0],
                  aa_list.this.mean(),
                  shape_log, mean_log]

        npt.assert_allclose(actual=actual,
                            desired=[15.477542212645357,
                                     279.30130009369515,
                                     10731.024327931776,
                                     0.009083584068527644,
                                     1520,
                                     0.014380159209694215],
                            rtol=1e-7)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.2.5 Example 5 - Automatic cell calibration

This scripts shows how to perform automatic cell calibration based on a time series of on and off band images containing both, suitable background images and images from different SO2 cells (in both wavelength channels, cf. *Example 0.7 - Manual cell calibration*).

Code

```python
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliß (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script no. 5 - Automatic cell calibration.

Script showing how to use the automatic cell calibration engine which only
requires to specify start / stop time stamps of a calibration window. Based on
that sub time windows for each cell as well as suitable background images are
detected and separated into individual image lists (for all filters, i.e. here
on / off).

"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version

import pyplis
from datetime import datetime
from time import time
from os.path import join
from matplotlib.pyplot import show, close

# IMPORT GLOBAL SETTINGS
from SETTINGS import SAVEFIGS, SAVE_DIR, FORMAT, DPI, IMG_DIR, OPTPARSE

# Check script version
check_version()

# File path for storing cell AA calibration data including sensitivity
# correction mask
AA_CALIB_FILE = join(SAVE_DIR, "ex05_cellcalib_aa.fts")


# SCRIPT FUNCTION DEFINITIONS
def perform_auto_cell_calib():
    # Calibration time stamps
```

(continues on next page)

(continued from previous page)

```
    start = datetime(2015, 9, 16, 6, 59, 00)
    stop = datetime(2015, 9, 16, 7, 3, 00)

    # Gas CDs in cells and cell ids
    # See supplementary package data about DOAS fit retrieval
    calib_cells = {'a37': [8.59e17, 2.00e17],
                   'a53': [4.15e17, 1.00e17],
                   'a57': [19.24e17, 3.00e17]}

    # the camera used
    cam_id = "ecII"

    # The camera filter setup is different from the ECII default setup and is
    # therefore defined explicitely
    filters = [pyplis.utils.Filter(type="on", acronym="F01"),
               pyplis.utils.Filter(type="off", acronym="F02")]

    # create camera setup, this includes the filename convention for
    # image separation
    cam = pyplis.setupclasses.Camera(cam_id=cam_id, filter_list=filters)

    # Create CellCalibSetup class for initiation of CellCalib object
    setup = pyplis.setupclasses.MeasSetup(IMG_DIR, start, stop,
                                          camera=cam,
                                          cell_info_dict=calib_cells)

    # Create CellCalibEngine object, read on...
    # This is a DataSet object and performs file separation and creation of
    # on / off, dark / offset lists for all images in the specified time window
    c = pyplis.cellcalib.CellCalibEngine(setup)

    # the following high level method calls several functions in the
    # CellCalibEngine class, most importantly the method find_cells for on and
    # off band image time series, which detects sub time windows for each cell
    # and background images. After the individual time windows are detected for
    # each cell and filter, the method _assign_calib_specs is called, which
    # assigns the SO2 CD amount (specified above in dictionary calib_cells)
    # to the detected sub time windows (both for on and off) based on the depth
    # of the intensity dip (in the onband) for each sub time window (should
    # become clear from the plot produced in this script). Then it creates
    # CellImgList objects for each of the cells and for the detected background
    # images (i.e. resulting in (M + 1) x 2 lists, with M being the number of
    # detected intensity dips, the + 1 is the corresponding background list and
    # times 2 for on / off)
    c.find_and_assign_cells_all_filter_lists()

    return c


# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    close("all")
    start = time()
    c = perform_auto_cell_calib()

    # prepare CellCalibData objects for on, off and aa images. These can
    # be accessed via c.calib_data[key] where key is "on", "off", "aa"
```

(continues on next page)

(continued from previous page)

```
    c.prepare_calib_data(pos_x_abs=None,  # change for a specific pix
                         pos_y_abs=None,  # change for a specific pix
                         radius_abs=1,  # radius of retrieval disk
                         on_id="on",  # ImgList ID of onband filter
                         off_id="off")  # ImgList ID of offband filter
    stop = time()
    # Plot search result of on
    ax0 = c.plot_cell_search_result("on", include_tit=False)
    ax1 = c.plot_cell_search_result("off", include_tit=False)
    # Plot all calibration curves for center pixel and in a radial
    # neighbourhood of 20 pixels
    ax2 = c.plot_all_calib_curves()
    ax2.set_xlim([0, 0.7])
    ax2.set_ylim([0, 2.25e18])
    # IMPORTANT STUFF FINISHED
    if SAVEFIGS:
        ax0.figure.savefig(join(SAVE_DIR, "ex05_2_out_1.%s" % FORMAT),
                           format=FORMAT, dpi=DPI)
        ax1.figure.savefig(join(SAVE_DIR, "ex05_2_out_2.%s" % FORMAT),
                           format=FORMAT, dpi=DPI)
        ax2.figure.savefig(join(SAVE_DIR, "ex05_2_out_3.%s" % FORMAT),
                           format=FORMAT, dpi=DPI)

    ax0.set_title("Cell search result on band", fontsize=18)
    ax1.set_title("Cell search result off band", fontsize=18)
    ax2.set_title("Calibration polynomials", fontsize=18)

    # You can explicitely access the individual CellCalibData objects
    aa_calib = c.calib_data["aa"]

    aa_calib.fit_calib_data()
    c.plot_calib_curve("on")
    mask = c.get_sensitivity_corr_mask("aa")

    aa_calib.save_as_fits(AA_CALIB_FILE)
    print("Time elapsed for preparing calibration data: %.4f s"
          % (stop - start))

    # IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt
        from os.path import basename
        calib_reload = pyplis.CellCalibData()
        calib_reload.load_from_fits(AA_CALIB_FILE)
        calib_reload.fit_calib_data(polyorder=2,
                                    through_origin=True)
        # test some basic features of calibraiton dataset (e.g. different
        # ImgList classes for on and off and the different cells)
        npt.assert_array_equal([c.cell_search_performed,
```

(continues on next page)

(continued from previous page)

```
                                    c.cell_lists["on"]["a37"].nof,
                                    c.cell_lists["on"]["a53"].nof,
                                    c.cell_lists["on"]["a57"].nof,
                                    c.cell_lists["off"]["a37"].nof,
                                    c.cell_lists["off"]["a53"].nof,
                                    c.cell_lists["off"]["a57"].nof,
                                    calib_reload.calib_id,
                                    calib_reload.pos_x_abs,
                                    calib_reload.pos_y_abs],
                                   [True, 2, 3, 3, 2, 3, 3, "aa", 672, 512])
        d = c._cell_info_auto_search

        vals_approx = [d["a37"][0],
                       d["a53"][0],
                       d["a57"][0],
                       aa_calib.calib_fun(0, *aa_calib.calib_coeffs),
                       calib_reload.calib_fun(0, *calib_reload.calib_coeffs)]
        npt.assert_allclose(actual=vals_approx,
                            desired=[8.59e+17,
                                     4.15e+17,
                                     1.924e+18,
                                     -1.8313164653590516e+16,
                                     0.0],
                            rtol=1e-7)

        # explicitely check calibration data for on, off and aa (plotted in
        # this script)
        actual = [c.calib_data["on"].calib_coeffs.mean(),
                  c.calib_data["off"].calib_coeffs.mean(),
                  aa_calib.calib_coeffs.mean(),
                  calib_reload.calib_coeffs.mean()]
        npt.assert_allclose(actual=actual,
                            desired=[1.8926803829028974e+18,
                                     -3.742539080945949e+19,
                                     2.153653759747737e+18,
                                     2.1197681750384312e+18],
                            rtol=1e-7)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.2.6 Example 6 - DOAS calibration

Introduction into DOAS calibration including FOV search using both, the Pearson and the IFR method.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
```

(continues on next page)

(continued from previous page)

```
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script no. 6 - DOAS calibration and FOV search.

Script showing how to work with DOAS calibration data

In this script, a stack of plume AA images from the Etna test data is imported
as well as a time series of DOAS SO2-CDs in the plume retrieved using the
software DOASIS (see directory "spectra" in test data folder for corresponding
analysis details, the folder also contains the RAW data and the jscript code
for analysing the spectra). The DOAS result import is performed using the
Python package ``pydoas``.

Based on these data, position and shape of the DOAS FOV within the camera
uimages is identified using both FOV search methods (IFR and Pearson). The
results of the FOV search are plotted as well as the corresponding calibration
curves retrieved for both FOV parametrisations.

Note
------

In case a MemoryError occurs while determining the AA image stack, then the
stack (3D numpy array) is too large for the RAM. In this case, try
increasing script option PYRLEVEL_ROUGH_SEARCH.

"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version

import pyplis
import pydoas
import numpy.testing as npt
import numpy as np
from datetime import timedelta
from matplotlib.pyplot import close, show, subplots
from os.path import join, exists
from os import remove

# IMPORT GLOBAL SETTINGS
from SETTINGS import SAVEFIGS, SAVE_DIR, FORMAT, DPI, IMG_DIR, OPTPARSE

# IMPORTS FROM OTHER EXAMPLE SCRIPTS
from ex04_prep_aa_imglist import prepare_aa_image_list

# Check script version
check_version()
```

(continues on next page)

(continued from previous page)

```
# SCRIPT OPTONS

# reload and save stack in folder SAVE_DIR, results in increased
# running time due to stack calculation (is automatically switched on if
# the stack is not found at this location)
RELOAD_STACK = False

PYRLEVEL_ROUGH_SEARCH = 2
# Default search settings are at pyramid level 2, the FOV results are upscaled
# to original resolution, if the following option is set 1, then, based on
# the result from pyrlevel=2, another stack is determined at pyrlevel = 0
# (i.e. in full resolution) within a ROI around the center position from the
# search performed at pyrlevel=PYRLEVEL_ROUGH_SEARCH (defined 2 lines below)
DO_FINE_SEARCH = False


# RELEVANT DIRECTORIES AND PATHS

# Directory containing DOAS result files
DOAS_DATA_DIR = join(IMG_DIR, "..", "spectra", "plume_prep", "min10Scans",
                     "ResultFiles")

STACK_PATH = join(SAVE_DIR, "ex06_aa_imgstack.fts")

# SCRIPT FUNCTION DEFINITIONS


def load_doas_results(lt_to_utc_shift=timedelta(-1. / 12)):
    """Specify DOAS data import from DOASIS fit result files.

    In order to perform the DOAS FOV search, as many spectrum datapoints
    as possible are needed. Therefore, only 10 spectra were added (to reduce
    noise) per plume spectrum. The DOAS fit was performed in a wavelength
    range between 314 - 326 nm (fit ID: "f01").
    """
    # This dictionary specifies which information is supposed to be imported
    # from the DOAS fit result files stored in DOAS_DATA_DIR. In the example
    # shown here, only the SO2 fit results are imported from fit scenario
    # with ID "f01" (key of dict). The corresponding value of each key is
    # a list of format ["header_id", ["fit_id1", "fit_id2", ..., "fit_idN"]]
    # specifying the identification string of the species in the result file
    # headers and the second entry is a list specifying all fit scenario IDs
    # from which this species is supposed to be imported (here only f01)
    fit_import_info = {"so2": ["SO2_Hermans_298_air_conv_satCorr1e18",
                               ["f01"]
                               ]}

    # Create a result import setup for the DOAS data based on the import
    # dictionary and the image base directory of the result files ...
    doas_import_setup =\
        pydoas.dataimport.ResultImportSetup(DOAS_DATA_DIR,
                                            result_import_dict=fit_import_info)

    # ... and create a result dataset from that
    doas_dataset = pydoas.analysis.DatasetDoasResults(doas_import_setup)
```

(continues on next page)

(continued from previous page)

```
    # get the SO2 fit results from the dataset. Individual results of certain
    # species can be accessed using the species ID (key in ``fit_import_info``
    # dict) and its fit ID (one of the fit IDs specified for this species, here
    # f01).
    # Note, that the DOAS data was stored using local time, thus they need to
    # be shifted (2h back) to match the camera data time stamps (which are in
    # UTC), otherwise the temporal merging of the two datasets (for the DOAS
    # calibration) does not work
    results_utc = doas_dataset.get_results("so2", "f01").shift(lt_to_utc_shift)
    return results_utc


def make_aa_stack_from_list(aa_list, roi_abs=None, pyrlevel=None,
                            save=True, stack_path=STACK_PATH,
                            save_dir=SAVE_DIR):
    """Get and prepare onband list for aa image mode."""
    # Deactivate auto reload to change some settings (if auto_reload is active
    # list images are reloaded whenever a setting is changed in the list. This
    # can slow down things, thus, if you intend to change a couple of settings
    # you might deactivate auto_reload, adapt the settings and then re-activate
    # auto_reload
    aa_list.auto_reload = False
    if roi_abs is not None:
        aa_list.roi_abs = roi_abs
        aa_list.crop = True
    aa_list.pyrlevel = pyrlevel
    aa_list.auto_reload = True

    # Stack all images in image list at pyrlevel 2 and cropped using specified
    # roi (uncropped if roi_abs=None).
    stack = aa_list.make_stack()
    if save:
        try:
            remove(stack_path)
        except BaseException:
            pass
        stack.save_as_fits(save_dir=save_dir,
                           save_name="ex06_aa_imgstack.fts")
    return stack


def get_stack(reload_stack=RELOAD_STACK, stack_path=STACK_PATH,
              pyrlevel=PYRLEVEL_ROUGH_SEARCH):
    """Load stack data based on current settings."""
    if not exists(stack_path):
        reload_stack = 1

    if not reload_stack:
        stack = pyplis.processing.ImgStack()
        stack.load_stack_fits(stack_path)
        if stack.pyrlevel != pyrlevel:
            reload_stack = True
    aa_list = None
    if reload_stack:
        # import AA image list
        aa_list = prepare_aa_image_list()
        # Try creating stack
```

(continues on next page)

(continued from previous page)

```
        stack = make_aa_stack_from_list(aa_list, pyrlevel=pyrlevel)

    return stack, aa_list


# Test functions used at the end of the script
def test_calib_pears_init(calib):
    calib.fit_calib_data(polyorder=1)
    cc = pyplis.helpers.get_img_maximum(calib.fov.corr_img.img)
    assert cc == (124, 159), cc

    pyrl = calib.fov.pyrlevel
    assert pyrl == 2, pyrl
    res_dict = calib.fov.result_pearson
    npt.assert_allclose([res_dict['rad_rel'],
                         np.max(100 * res_dict['corr_curve'].values)],
                        [3, 95], atol=1)

    fov_ext = calib.fov.pixel_extend(abs_coords=True)
    (fov_x, fov_y) = calib.fov.pixel_position_center(abs_coords=True)

    npt.assert_allclose([fov_ext, fov_x, fov_y],
                        [res_dict['rad_rel'] * 2**pyrl, 636, 496], atol=1)

    npt.assert_allclose(calib.calib_coeffs,
                        [8.58e+18, 2.71e+17], rtol=1e-1)


def test_calib_pears_fine(calib):
    cc = pyplis.helpers.get_img_maximum(calib.fov.corr_img.img)
    npt.assert_allclose((186, 180), cc, atol=1)

    pyrl = calib.fov.pyrlevel
    assert pyrl == 0, pyrl
    res_dict = calib.fov.result_pearson
    npt.assert_allclose([res_dict['rad_rel'],
                         np.max(100 * res_dict['corr_curve'].values)],
                        [6, 95], atol=1)

    fov_ext = calib.fov.pixel_extend(abs_coords=True)
    (fov_x, fov_y) = calib.fov.pixel_position_center(abs_coords=True)

    npt.assert_allclose([fov_ext, fov_x, fov_y],
                        [6, 630, 493], atol=1)

    npt.assert_allclose(calib.calib_coeffs,
                        [8.38e+18, 2.92e+17], rtol=1e-1)


def test_calib_ifr(calib):
    cc = pyplis.helpers.get_img_maximum(calib.fov.corr_img.img)
    npt.assert_allclose((123, 157), cc, atol=1)

    pyrl = calib.fov.pyrlevel
    assert pyrl == 2, pyrl
    npt.assert_allclose(calib.fov.result_ifr['popt'][1:5],
                        [158.6, 122.9, 15.4, 1.5], rtol=1e-1)
```

(continues on next page)

(continued from previous page)

```
    (fov_x, fov_y) = calib.fov.pixel_position_center(abs_coords=True)

    npt.assert_allclose([fov_x, fov_y, calib_ifr.fov.sigma_x_abs,
                         calib_ifr.fov.sigma_y_abs],
                        [635, 492, 61.5, 41.6], atol=2)

    npt.assert_allclose(calib.calib_coeffs,
                        [9.38e+18, 1.75e+17], rtol=1e-1)


# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    # close all plots
    close("all")

    # import DOAS results
    doas_time_series = load_doas_results()

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    if options.test:
        # if test mode is active, the image stack is always recomputed from
        # scratch and the option DO_FINE_SEARCH is activated, since the tests
        # are based on this. This will lead to an increased computation time
        # Test mode can be activated / deactivated in SETTINGS.py or via
        # the script option --test 1 (on) or --test 0 (off)
        RELOAD_STACK = True
        PYRLEVEL_ROUGH_SEARCH = 2
        DO_FINE_SEARCH = True

    # reload or create the AA image stack based on current script settings
    stack, aa_list = get_stack()

    s = pyplis.doascalib.DoasFOVEngine(stack, doas_time_series)
    calib_pears = s.perform_fov_search(method="pearson")
    calib_ifr = s.perform_fov_search(method="ifr", ifrlbda=4e-3)

    # plot the FOV search results
    ax0 = calib_pears.fov.plot()
    ax1 = calib_ifr.fov.plot()

    calib_pears.fit_calib_data()
    calib_ifr.fit_calib_data()

    fig, ax2 = subplots(1, 1)
    calib_pears.plot(add_label_str="Pearson", color="b", ax=ax2)

    calib_ifr.plot(add_label_str="IFR", color="g", ax=ax2)
    ax2.set_title("Calibration curves Pearson vs. IFR method")
    ax2.grid()
    ax2.set_ylim([0, 1.8e18])
    ax2.set_xlim([0, 0.20])
    ax2.legend(loc=4, fancybox=True, framealpha=0.7, fontsize=11)
    axes = [ax0, ax1, ax2]
```

(continues on next page)

(continued from previous page)

```
    if DO_FINE_SEARCH:
        """Perform FOV search within ROI around result from pearson fov
        search at full resolution (pyrlevel=0)
        """
        if aa_list is None:
            aa_list = prepare_aa_image_list()
        # remember some properties of the current image stack that is stored in
        # the DoasFOVEngine object (i.e. the merged one in low pixel res.)

        num_merge, h, w = s.img_stack.shape
        s_fine = s.run_fov_fine_search(aa_list, doas_time_series,
                                       method="pearson")

        calib_pears_fine = s_fine.calib_data
        calib_pears_fine.plot()
        calib_pears_fine.fov.plot()

    calib_pears.save_as_fits(save_dir=SAVE_DIR,
                             save_name="ex06_doascalib_aa.fts")
    calib_ifr.save_as_fits(save_dir=SAVE_DIR,
                           save_name="ex06_doascalib_aa_ifr_method.fts")

    # you can also change the order of the calibration polynomial and
    # force it to go through the origin
    calib_pears.fit_calib_data(polyorder=2, through_origin=True)
    calib_pears.plot_calib_fun(add_label_str="Pearson (WRONG,\n"
                               "2nd order, through origin)",
                               color="r", ax=ax2)
    ax2.legend(loc=4, fancybox=True, framealpha=0.7, fontsize=11)
    # IMPORTANT STUFF FINISHED
    if SAVEFIGS:
        for k in range(len(axes)):
            ax = axes[k]
            ax.set_title("")
            ax.figure.savefig(join(SAVE_DIR, "ex06_out_%d.%s"
                                   % ((k + 1), FORMAT)),
                              format=FORMAT, dpi=DPI)

    # IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        from os.path import basename

        num, h, w = stack.shape
        num2 = s.img_stack.shape[0]  # stack after fine FOV search
        prep = stack.img_prep

        # check some basic properties of the data used for the different FOV
        # searches
        npt.assert_array_equal(
            [len(doas_time_series), num, num_merge, h, w, stack.pyrlevel,
             prep["darkcorr"] * prep["is_tau"] * prep["is_aa"], num2],
            [120, 209, 88, 256, 336, 2, 1, 209])
```

(continues on next page)

(continued from previous page)

```
        test_calib_pears_init(calib_pears)
        test_calib_ifr(calib_ifr)
        test_calib_pears_fine(calib_pears_fine)

        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.2.7 Example 7 - AA sensitivity correction masks

Combine the results from *Example 5 - Automatic cell calibration* and *Example 6 - DOAS calibration* in order to retrieve AA sensitivity correction masks normalised to the position of the DOAS FOV.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script no. 7 - AA sensitivity correction masks.

In this script, cell and DOAS calibration (see previous 2 scripts) of the Etna
test dataset are opposed. Furthermore, it is illustrated, how to create
correction masks for pixel variations in the SO2 sensitivity due to shifts in
the filter transmission windows.

The cell calibration is re-performed (using method ``perform_auto_cell_calib``)
from example script 5. The results from the DOAS calibration
(see prev. example) were stored as a FITS file (including FOV information)
and the results are imported here.
"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version

import pyplis
from os.path import join, exists
import numpy as np
```

(continues on next page)

(continued from previous page)

```
from matplotlib.pyplot import close, subplots, show
from matplotlib.patches import Circle
import six

# IMPORT GLOBAL SETTINGS
from SETTINGS import SAVEFIGS, SAVE_DIR, FORMAT, DPI, OPTPARSE

# IMPORTS FROM OTHER EXAMPLE SCRIPTS
from ex05_cell_calib_auto import perform_auto_cell_calib
from ex04_prep_aa_imglist import prepare_aa_image_list

# Check script version
check_version()

CELL_AA_CALIB_FILE = join(SAVE_DIR, "ex05_cellcalib_aa.fts")
# RELEVANT DIRECTORIES AND PATHS

# fits file containing DOAS calibration information (from ex6)
DOAS_CALIB_FILE = join(SAVE_DIR, "ex06_doascalib_aa.fts")


# SCRIPT FUNCTION DEFINITIONS
def draw_doas_fov(fov_x, fov_y, fov_extend, ax):
    # add FOV position to plot of examplary AA image
    c = Circle((fov_x, fov_y), fov_extend, ec="k", fc="lime", alpha=.5)
    ax.add_artist(c)
    ax.text(fov_x, (fov_y - fov_extend * 1.3), "DOAS FOV")
    ax.set_xlim([0, 1343]), ax.set_ylim([1023, 0])
    return ax


def plot_pcs_comparison(aa_init, aa_imgs_corr, pcs1, pcs2):
    fig, axes = subplots(1, 2, figsize=(18, 6))
    p10 = pcs1.get_line_profile(aa_init.img)
    p20 = pcs2.get_line_profile(aa_init.img)

    num = len(p10)

    axes[0].set_title("Line %s" % pcs1.line_id)
    axes[1].set_title("Line %s" % pcs2.line_id)

    axes[0].plot(p10, "-", label=r"Init $\phi=%.3f$" % (sum(p10) / num))
    axes[1].plot(p20, "-", label=r"Init $\phi=%.3f$" % (sum(p20) / num))

    for cd, aa_corr in six.iteritems(aa_imgs_corr):
        p1 = pcs1.get_line_profile(aa_corr.img)
        p2 = pcs2.get_line_profile(aa_corr.img)

        axes[0].plot(p1, "-", label=r"Cell CD: %.2e $\phi=%.3f$"
                     % (cd, sum(p1) / num))
        axes[1].plot(p2, "-", label=r"Cell CD: %.2e $\phi=%.3f$"
                     % (cd, sum(p2) / num))

    axes[0].legend(loc='best', fancybox=True, framealpha=0.5, fontsize=10)
    axes[1].legend(loc='best', fancybox=True, framealpha=0.5, fontsize=10)
    axes[0].grid()
    axes[1].grid()
```

(continues on next page)

(continued from previous page)

```
    return fig, axes


# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    close("all")

    if not exists(DOAS_CALIB_FILE):
        raise IOError("Calibration file could not be found at specified "
                      "location:\n %s\nYou might need to run example 6 first")

    # Load AA list
    aa_list = prepare_aa_image_list()
    aa_list.add_gaussian_blurring(2)

    # Load DOAS calbration data and FOV information (see example 6)
    doascalib = pyplis.doascalib.DoasCalibData()
    doascalib.load_from_fits(file_path=DOAS_CALIB_FILE)
    doascalib.fit_calib_data()

    # Get DOAS FOV parameters in absolute coordinates
    fov_x, fov_y = doascalib.fov.pixel_position_center(abs_coords=True)
    fov_extend = doascalib.fov.pixel_extend(abs_coords=True)

    # Load cell calibration (see example 5)
    cellcalib = perform_auto_cell_calib()

    # get cell calibration
    cellcalib.prepare_calib_data(
        pos_x_abs=fov_x,  # change if you want it for a specific pix
        pos_y_abs=fov_y,  # change if you want it for a specific pix
        radius_abs=fov_extend,  # radius of retrieval disk
        on_id="on",  # ImgList ID of onband filter
        off_id="off")  # ImgList ID of offband filter

    cell_aa_calib = cellcalib.calib_data["aa"]

    # Define lines on image for plume profiles
    pcs1 = pyplis.LineOnImage(620, 700, 940, 280,
                              line_id="center")
    pcs2 = pyplis.LineOnImage(40, 40, 40, 600,
                              line_id="edge")

    # Plot DOAS calibration polynomial
    ax0 = doascalib.plot(add_label_str="DOAS")
    ax0 = cellcalib.calib_data["aa"].plot(ax=ax0, c="r")
    ax0.set_title("")
    ax0.set_xlim([0, 0.5])

    # Get current AA image from image list
    aa_init = aa_list.current_img()

    # now determine sensitivity correction masks from the different cells
    masks = {}
    aa_imgs_corr = {}
    for cd in cell_aa_calib.cd_vec:
        mask = cellcalib.get_sensitivity_corr_mask("aa",
```

(continues on next page)

(continued from previous page)

```
                                                        pos_x_abs=fov_x,
                                                        pos_y_abs=fov_y,
                                                        radius_abs=fov_extend,
                                                        cell_cd_closest=cd)
        masks[cd] = mask
        aa_imgs_corr[cd] = pyplis.Img(aa_init.img / mask.img)

    # get mask corresponding to minimum cell CD
    mask = list(masks.values())[np.argmin(list(masks.keys()))]

    # assing mask to aa_list
    aa_list.senscorr_mask = mask

    # activate AA sensitivity correction in list
    aa_list.sensitivity_corr_mode = True

    # set DOAS calibration data in list ...
    aa_list.calib_data = doascalib

    # ... and activate calibration mode
    aa_list.calib_mode = True
    ax = aa_list.current_img().show(zlabel=r"$S_{SO2}$ [cm$^{-2}$]")

    # plot the two lines into the exemplary AA image
    pcs1.plot_line_on_grid(ax=ax, color="r")
    pcs2.plot_line_on_grid(ax=ax, color="g")
    ax.legend(loc='best', fancybox=True, framealpha=0.5, fontsize=10)
    ax = draw_doas_fov(fov_x, fov_y, fov_extend, ax=ax)

    fig, _ = plot_pcs_comparison(aa_init, aa_imgs_corr, pcs1, pcs2)

    # IMPORTANT STUFF FINISHED

    if SAVEFIGS:
        ax0.figure.savefig(join(SAVE_DIR, "ex07_out_1.%s" % FORMAT),
                           format=FORMAT, dpi=DPI)
        ax.figure.savefig(join(SAVE_DIR, "ex07_out_2.%s" % FORMAT),
                          format=FORMAT, dpi=DPI)
        fig.savefig(join(SAVE_DIR, "ex07_out_3.%s" % FORMAT), format=FORMAT,
                    dpi=DPI)

    # Save the sensitivity correction mask from the cell with the lowest SO2 CD
    so2min = np.min(list(masks.keys()))
    mask = masks[so2min]
    mask.save_as_fits(SAVE_DIR, "ex07_aa_corr_mask")

    # assign mask in doascalib and resave
    doascalib.senscorr_mask = mask
    doascalib.save_as_fits(DOAS_CALIB_FILE)

    # IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
```

(continues on next page)

(continued from previous page)

```
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt
        from os.path import basename

        npt.assert_array_equal([],
                               [])

        npt.assert_allclose(actual=[],
                            desired=[],
                            rtol=1e-7)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.2.8 Example 8 - Plume velocity retrieval (Cross correlation)

In this script an exemplary plume velocity retrieval is performed using the signal cross correlation algorithm. The velocity is retrieved based on two plume cross sections and a time series of plume AA images (using the AA `ImgList` created in *Example 4 - Preparation of AA image list*).

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script no. 8 - Plume velocity retrieval by cross correlation.

"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version

from matplotlib.pyplot import close, show, subplots
from os.path import join
from time import time

```

(continues on next page)

(continued from previous page)

```
from pyplis.plumespeed import VeloCrossCorrEngine

# IMPORT GLOBAL SETTINGS
from SETTINGS import SAVEFIGS, SAVE_DIR, FORMAT, DPI, OPTPARSE, LINES

# IMPORTS FROM OTHER EXAMPLE SCRIPTS
from ex04_prep_aa_imglist import prepare_aa_image_list

# Check script version
check_version()

# SCRIPT OPTONS

# distance in pixels between two lines used for cross correlation analysis
OFFSET_PIXNUM = 40
RELOAD = 0  # reload AA profile images for PCS lines

# start / stop indices of considered images in image list (only relevant if PCS
# profiles are reloaded, i.e. Opt RELOAD=True)
START_IDX = 10
STOP_IDX = 200

# PCS line for which velocity is supposed to retrieved
PCS = LINES[0]  # orange "young_plume" line

# Color of PCS offset line used to perform cross correlation analysis
# (relevant for illustration)
COLOR_PCS_OFFS = "c"

# RELEVANT DIRECTORIES AND PATHS
# the time series of PCS profiles for both lines are stored as
# ProfileTimeSeriesImg objects using the following names. The images
# contain the PCS profiles (y-axis) for each image in the list (y-axis)
# and have thus dimension MXN where M denotes the pixel number of the lines
# and N denotes the total number of images from which the profiles are
# extracted. The images will be stored in SAVE_DIR after this script is run
# once. After that, re-running the script and applying the cross-correlation
# analysis will be much faster, since the profiles are imported from the
# two precomupted images and do not need to be extracted by looping over
# the image list.
PCS_PROFILES_PIC_NAME = "ex08_ica_tseries_pcs.fts"
OFFSET_PROFILES_PIC_NAME = "ex08_ica_tseries_offset.fts"


# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    close("all")
    axes = []
    # prepare the AA image list (see ex4)
    aa_list = prepare_aa_image_list()
    aa_list.pyrlevel = 1

    t0 = time()
    cc = VeloCrossCorrEngine(aa_list, PCS)
    cc.create_parallel_pcs_offset(offset_pix=40,
                                  color=COLOR_PCS_OFFS,
                                  linestyle="--")
```

(continues on next page)

(continued from previous page)

```
    reloaded = False  # just a flag for output below
    try:
        if RELOAD:
            raise Exception
        cc.load_pcs_profile_img(join(SAVE_DIR, PCS_PROFILES_PIC_NAME),
                                line_id="pcs")
        cc.load_pcs_profile_img(join(SAVE_DIR, OFFSET_PROFILES_PIC_NAME),
                                line_id="pcs_offset")
    except BaseException:
        cc.get_pcs_tseries_from_list(start_idx=START_IDX,
                                     stop_idx=STOP_IDX)
        cc.save_pcs_profile_images(save_dir=SAVE_DIR,
                                   fname1=PCS_PROFILES_PIC_NAME,
                                   fname2=OFFSET_PROFILES_PIC_NAME)
        reloaded = True
    t1 = time()
    # the run method of the high level VeloCrossCorrEngine class is
    # basically a wrapper method for the low-level find_signal_correlation
    # function which is part of the plumespeed.py module. Before calling
    # the latter, the ICA time-series are extracted from the two
    # ProfileTimeSeriesImg objects which were computed above from the
    # ImgList class containing AA images, and which are stored as FITS
    # files for fast re-computing of this script. The following run
    # command passes valid input parameters to the find_signal_correlation
    # method.
    velo = cc.run(cut_border_idx=10,
                  reg_grid_tres=100,
                  freq_unit="L",
                  sigma_smooth=2,
                  plot=0)
    t2 = time()
    fig, ax = subplots(1, 2, figsize=(20, 6))
    axes.append(cc.plot_pcs_lines(ax=ax[0]))
    cc.plot_ica_tseries_overlay(ax=ax[1])
    axes.append(cc.plot_corrcoeff_tseries())

    print("Result performance analysis\n"
          "Images reloaded from list: %s\n"
          "Number of images: %d\n"
          "Create ICA images: %.3f s\n"
          "Cross-corr analysis: %.3f s"
          % (reloaded, (STOP_IDX - START_IDX), (t1 - t0), (t2 - t1)))

    print("Retrieved plume velocity of v = %.2f m/s" % velo)

    # IMPORTANT STUFF FINISHED
    if SAVEFIGS:
        for k in range(len(axes)):
            axes[k].figure.savefig(join(SAVE_DIR, "ex08_out_%d.%s"
                                        % ((k + 1), FORMAT)),
                                   format=FORMAT, dpi=DPI)

    # IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()
```

(continues on next page)

(continued from previous page)

```
    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt
        from os.path import basename

        npt.assert_array_equal([],
                               [])

        npt.assert_allclose(actual=[],
                            desired=[],
                            rtol=1e-7)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.2.9 Example 9 - Plume velocity retrieval (Optical flow Farneback)

This script gives an introduction into plume velocity retrievals using the Farneback optical flow algorithm (`OpticalFlowFarneback`) and a histogram based post analysis.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliß (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script no. 9 - Optical flow Plume velocity retrieval."""
from __future__ import (absolute_import, division)

from os.path import join
import pyplis
# IMPORT GLOBAL SETTINGS
from SETTINGS import SAVEFIGS, SAVE_DIR, FORMAT, DPI, OPTPARSE, LINES
from matplotlib.pyplot import (close, show, subplots, figure, xticks, yticks,
                               sca, rcParams)
```

(continues on next page)

(continued from previous page)

```
# IMPORTS FROM OTHER EXAMPLE SCRIPTS
from ex04_prep_aa_imglist import prepare_aa_image_list
from SETTINGS import check_version

rcParams["font.size"] = 16
PCS1, PCS2 = LINES

# Check script version
check_version()

# SCRIPT OPTIONS

PEAK_SIGMA_TOL = 2

# perform histogram analysis for all images in time series
HISTO_ANALYSIS_ALL = 1
# applies multi gauss fit to retrieve local predominant displacement
# direction, if False, then the latter is calculated from 1. and 2. moment
# of histogram (Faster but more sensitive to additional peaks in histogram)
HISTO_ANALYSIS_MULTIGAUSS = True
HISTO_ANALYSIS_START_IDX = 0
HISTO_ANALYSIS_STOP_IDX = None  # 207

# Gauss pyramid level
PYRLEVEL = 1
BLUR = 0
ROI_CONTRAST = [0, 0, 1344, 730]
MIN_AA = 0.05


def analyse_and_plot(lst, lines):
    fig = figure(figsize=(14, 8))

    ax0 = fig.add_axes([0.01, 0.15, 0.59, 0.8])
    # ax0.set_axis_off()
    ax1 = fig.add_axes([0.61, 0.15, 0.16, 0.8])
    ax2 = fig.add_axes([0.78, 0.15, 0.16, 0.8])
    mask = lst.get_thresh_mask(MIN_AA)
    fl = lst.optflow
    fl.plot(ax=ax0)  # , in_roi=True)
    for line in lines:
        m = mask * line.get_rotated_roi_mask(fl.flow.shape[:2])
        line.plot_line_on_grid(ax=ax0, include_normal=1,
                               include_roi_rot=1)
        try:
            _, mu, sigma = fl.plot_orientation_histo(pix_mask=m,
                                                     apply_fit=True, ax=ax1,
                                                     color=line.color)
            ax1.legend_.remove()
            low, high = mu - sigma, mu + sigma
            fl.plot_length_histo(pix_mask=m, ax=ax2, dir_low=low,
                                 dir_high=high, color=line.color)
        except BaseException:
            pass
    # pyplis.helpers.set_ax_lim_roi(roi_disp, ax0)
    ax0.get_xaxis().set_ticks([])
    ax0.get_yaxis().set_ticks([])
```

(continues on next page)

(continued from previous page)

```
    ax0.set_title("")

    ymax = max([ax1.get_ylim()[1], ax2.get_ylim()[1]])
    ax1.set_title("")
    ax1.set_xlabel(r"$\varphi\,[^\circ]$", fontsize=20)
    ax1.set_ylim([0, ymax])
    ax1.get_yaxis().set_ticks([])
    ax2.yaxis.tick_right()
    ax2.yaxis.set_label_position("right")
    ax2.set_xlabel(r"$|\mathbf{f}|$ [pix]", fontsize=20)
    ax2.set_ylabel("Counts / bin", fontsize=20)
    ax2.set_ylim([0, ymax])

    ax2.set_title("")
    ax2.legend_.remove()

    sca(ax1)
    xticks(rotation=40, ha="right")
    sca(ax2)
    yticks(rotation=90, va="center")

    return fig


# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    close("all")
    figs = []
    # Prepare aa image list (see example 4)
    aa_list = prepare_aa_image_list()

    # the aa image list includes the measurement geometry, get pixel
    # distance image where pixel values correspond to step widths in the plume,
    # obviously, the distance values depend on the downscaling factor, which
    # is calculated from the analysis pyramid level (PYRLEVEL)
    dist_img, _, _ = aa_list.meas_geometry.compute_all_integration_step_lengths(   #␣
↪noqa: E501
        pyrlevel=PYRLEVEL)
    # set the pyramid level in the list
    aa_list.pyrlevel = PYRLEVEL
    # add some blurring.. or not (if BLUR = 0)
    aa_list.add_gaussian_blurring(BLUR)

    # Access to the optical flow module in the image list. If optflow_mode is
    # active in the list, then, whenever the list index changes (e.g. using
    # list.goto_next(), or list.goto_img(100)), the optical flow field is
    # calculated between the current list image and the next one
    fl = aa_list.optflow
    # (! note: fl is only a pointer, i.e. the "=" is not making a copy of the
    # object, meaning, that whenever something changes in "fl", it also does
    # in "aa_list.optflow")

    # Now activate optical flow calculation in list (this slows down the
    # speed of the analysis, since the optical flow calculation is
    # comparatively slow
    s = aa_list.optflow.settings
    s.hist_dir_gnum_max = 10
```

(continues on next page)

(continued from previous page)

```
    s.hist_dir_binres = 10
    s.hist_sigma_tol = PEAK_SIGMA_TOL

    s.roi_rad = ROI_CONTRAST

    aa_list.optflow_mode = True

    plume_mask = pyplis.Img(aa_list.get_thresh_mask(MIN_AA))
    plume_mask.show(tit="AA threshold mask")

    figs.append(analyse_and_plot(aa_list, LINES))

    figs.append(fl.plot_flow_histograms(PCS1, plume_mask.img))
    figs.append(fl.plot_flow_histograms(PCS2, plume_mask.img))

    # Show an image containing plume speed magnitudes (ignoring direction)
    velo_img = pyplis.Img(fl.to_plume_speed(dist_img))
    velo_img.show(vmin=0, vmax=10, cmap="Greens",
                  tit="Optical flow plume velocities",
                  zlabel="Plume velo [m/s]")

    # Create two objects used to store time series information about the
    # retrieved plume properties
    plume_props_l1 = pyplis.plumespeed.LocalPlumeProperties(PCS1.line_id)
    plume_props_l2 = pyplis.plumespeed.LocalPlumeProperties(PCS2.line_id)

    if HISTO_ANALYSIS_ALL:
        aa_list.goto_img(HISTO_ANALYSIS_START_IDX)
        if HISTO_ANALYSIS_STOP_IDX is None:
            HISTO_ANALYSIS_STOP_IDX = aa_list.nof - 1
        for k in range(HISTO_ANALYSIS_START_IDX, HISTO_ANALYSIS_STOP_IDX):
            plume_mask = aa_list.get_thresh_mask(MIN_AA)
            plume_props_l1.get_and_append_from_farneback(
                fl, line=PCS1, pix_mask=plume_mask,
                dir_multi_gauss=HISTO_ANALYSIS_MULTIGAUSS)
            plume_props_l2.get_and_append_from_farneback(
                fl, line=PCS2, pix_mask=plume_mask,
                dir_multi_gauss=HISTO_ANALYSIS_MULTIGAUSS)
            aa_list.goto_next()

# ==============================================================================
#         plume_props_l1 = plume_props_l1.interpolate()
#         plume_props_l2 = plume_props_l2.interpolate()
# ==============================================================================

        fig, ax = subplots(2, 1, figsize=(10, 9))

        plume_props_l1.plot_directions(ax=ax[0],
                                       color=PCS1.color,
                                       label="PCS1")
        plume_props_l2.plot_directions(ax=ax[0], color=PCS2.color,
                                       label="PCS2")

        plume_props_l1.plot_magnitudes(normalised=True, ax=ax[1],
                                       date_fmt="%H:%M:%S", color=PCS1.color,
                                       label="PCS1")
        plume_props_l2.plot_magnitudes(normalised=True, ax=ax[1],
```

(continues on next page)

(continued from previous page)

```
                                    date_fmt="%H:%M:%S", color=PCS2.color,
                                    label="PCS2")
        ax[0].set_xticklabels([])
        # ax[0].legend(loc='best', fancybox=True, framealpha=0.5, fontsize=14)
        # ax[0].set_title("Movement direction")
        # ax[1].set_title("Displacement length")
        figs.append(fig)
        # Save the time series as txt
        plume_props_l1.save_txt(join(SAVE_DIR,
                                  "ex09_plumeprops_young_plume.txt"))
        plume_props_l2.save_txt(join(SAVE_DIR,
                                  "ex09_plumeprops_aged_plume.txt"))

    if SAVEFIGS:
        for k in range(len(figs)):
            figs[k].savefig(join(SAVE_DIR, "ex09_out_%d.%s"
                                 % ((k + 1), FORMAT)),
                            format=FORMAT, dpi=DPI)

    # IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt
        from os.path import basename

        npt.assert_array_equal([],
                               [])

        npt.assert_allclose(actual=[],
                            desired=[],
                            rtol=1e-7)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.2.10 Example 10 - Import plume background images

Create a `Dataset` object for a time interval containing only plume background images (on / off).

Note: Stand alone script that is not required for any of the following scripts

Code

```python
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliß (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script no. 10 - Create background image dataset."""
from __future__ import (absolute_import, division)

from SETTINGS import check_version

import pyplis
from datetime import datetime
from matplotlib.pyplot import show

# IMPORT GLOBAL SETTINGS
from SETTINGS import IMG_DIR, OPTPARSE

# Check script version
check_version()

# SCRIPT FUNCTION DEFINITIONS


def get_bg_image_lists():
    """Initialize measurement setup and creates dataset from that."""
    start = datetime(2015, 9, 16, 7, 2, 0o5)
    stop = datetime(2015, 9, 16, 7, 2, 30)
    # Define camera (here the default ecII type is used)
    cam_id = "ecII"

    # the camera filter setup
    filters = [pyplis.utils.Filter(type="on", acronym="F01"),
               pyplis.utils.Filter(type="off", acronym="F02")]

    # create camera setup
    cam = pyplis.setupclasses.Camera(cam_id=cam_id, filter_list=filters)

    # Create BaseSetup object (which creates the MeasGeometry object)
    stp = pyplis.setupclasses.MeasSetup(IMG_DIR, start, stop, camera=cam)

    ds = pyplis.dataset.Dataset(stp)
    on, off = ds.get_list("on"), ds.get_list("off")
    on.darkcorr_mode = True
    off.darkcorr_mode = True
    return on, off
```

(continues on next page)

(continued from previous page)

```
# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    on, off = get_bg_image_lists()
    on.show_current()
    off.show_current()

    # IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt
        from os.path import basename

        npt.assert_array_equal([],
                               [])

        npt.assert_allclose(actual=[],
                            desired=[],
                            rtol=1e-7)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.2.11 Example 11 - Image based signal dilution correction

This script introduces the image based signal dilution correction including automatic retrieval of terrain distances on a pixel basis.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliß (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
```

(continues on next page)

(continued from previous page)

```
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script no. 11 - Image based signal dilution correction.

This script illustrates how extinction coefficients can be retrieved from
image data using the DilutionCorr class and by specifying suitable terrain
features in the images.

The extinction coefficients are retrieved from one on an from one off band
image recorded ~15 mins before the dataset used in the other examples. The
latter data is less suited for illustrating the feature since it contains
less terrain (therfore more sky background).

The two example images are then corrected for dilution and the results are
plotted (as comparison of the retrieved emission rate along an exemplary
plume cross section)
"""
from __future__ import (absolute_import, division)

from SETTINGS import check_version

import pyplis as pyplis
from geonum import GeoPoint
from matplotlib.pyplot import show, close, subplots, Rectangle, plot
from datetime import datetime
from os.path import join, exists

from pyplis.dilutioncorr import DilutionCorr
from pyplis.doascalib import DoasCalibData

# IMPORT GLOBAL SETTINGS
from SETTINGS import IMG_DIR, SAVEFIGS, SAVE_DIR, FORMAT, DPI, OPTPARSE
# IMPORTS FROM OTHER EXAMPLE SCRIPTS
from ex10_bg_imglists import get_bg_image_lists

# Check script version
check_version()

# SCRIPT OPTONS
# lower boundary for I0 value in dilution fit
I0_MIN = 0.0
AA_THRESH = 0.03

# exemplary plume cross section line for emission rate retrieval (is also used
# for full analysis in ex12)
PCS_LINE = pyplis.LineOnImage(x0=530, y0=586, x1=910, y1=200, line_id="pcs")
# Retrieval lines for dilution correction (along these lines, topographic
# distances and image radiances are determined for fitting the atmospheric
# extinction coefficients)
TOPO_LINE1 = pyplis.LineOnImage(1100, 650, 1000, 900, line_id="flank far",
                                color="lime",
                                linestyle="-")

TOPO_LINE2 = pyplis.LineOnImage(1000, 990, 1100, 990, line_id="flank close",
                                color="#ff33e3",
                                linestyle="-")
```

(continues on next page)

(continued from previous page)

```
# all lines in this array are used for the analysis
USE_LINES = [TOPO_LINE1, TOPO_LINE2]

# specify pixel resolution of topographic distance retrieval (every nth pixel
# is used)
SKIP_PIX_LINES = 10

# Specify region of interest used to extract the ambient intensity (required
# for dilution correction)
AMBIENT_ROI = [1240, 10, 1300, 70]

# Specify plume velocity (for emission rate estimate)
PLUME_VELO = 4.14  # m/s (result from ex8)

# RELEVANT DIRECTORIES AND PATHS
CALIB_FILE = join(SAVE_DIR, "ex06_doascalib_aa.fts")

# SCRIPT FUNCTION DEFINITIONS


def create_dataset_dilution():
    """Create a :class:`pyplis.dataset.Dataset` object for dilution analysis.

    The test dataset includes one on and one offband image which are recorded
    around 6:45 UTC at lower camera elevation angle than the time series shown
    in the other examples (7:06 - 7:22 UTC). Since these two images contain
    more topographic features they are used to illustrate the image based
    signal dilution correction.

    This function sets up the measurement (geometry, camera, time stamps) for
    these two images and creates a Dataset object.
    """
    start = datetime(2015, 9, 16, 6, 43, 00)
    stop = datetime(2015, 9, 16, 6, 47, 00)
    # the camera filter setup
    cam_id = "ecII"
    filters = [pyplis.utils.Filter(type="on", acronym="F01"),
               pyplis.utils.Filter(type="off", acronym="F02")]

    geom_cam = {"lon": 15.1129,
                "lat": 37.73122,
                "elev": 15.0,  # from field notes, will be corrected
                "elev_err": 5.0,
                "azim": 274.0,  # from field notes, will be corrected
                "azim_err": 10.0,
                "focal_length": 25e-3,
                "alt_offset": 7}  # meters above topography

    # create camera setup
    cam = pyplis.setupclasses.Camera(cam_id=cam_id, filter_list=filters,
                                     **geom_cam)

    # Load default information for Etna
    source = pyplis.setupclasses.Source("etna")

    # Provide wind direction
```

(continues on next page)

(continued from previous page)

```
    wind_info = {"dir": 0.0,
                 "dir_err": 15.0}

    # Create BaseSetup object (which creates the MeasGeometry object)
    stp = pyplis.setupclasses.MeasSetup(IMG_DIR, start, stop, camera=cam,
                                        source=source,
                                        wind_info=wind_info)
    return pyplis.dataset.Dataset(stp)


def find_view_dir(geom):
    """Perform a correction of the viewing direction using crater in img.

    :param MeasGeometry geom: measurement geometry
    :param str which_crater: use either "ne" (northeast) or "se" (south east)
    :return: - MeasGeometry, corrected geometry
    """
    # Use position of NE crater in image
    posx, posy = 1051, 605  # pixel position of NE crate in image
    # Geo location of NE crater (info from Google Earth)
    ne_crater = GeoPoint(37.754788, 14.996673, 3287, name="NE crater")

    geom.find_viewing_direction(pix_x=posx, pix_y=posy, pix_pos_err=100,
                                geo_point=ne_crater, draw_result=True)
    return geom


def prepare_lists(dataset):
    """Prepare on and off lists for dilution analysis.

    Steps:

        1. get on and offband list
        #. load background image list on and off (from ex10)
        #. set image preparation and assign background images to on / off list
        #. configure plume background model settings

    :param Dataset dataset: the dilution dataset (see
        :func:`create_dataset_dilution`)
    :return:
        - ImgList, onlist
        - ImgList, offlist

    """
    onlist = dataset.get_list("on")
    offlist = dataset.get_list("off")
    # dark_corr_mode already active
    bg_onlist, bg_offlist = get_bg_image_lists()

    # prepare img pre-edit
    onlist.darkcorr_mode = True
    onlist.gaussian_blurring = 2
    offlist.darkcorr_mode = True
    offlist.gaussian_blurring = 2

    # prepare background images in lists (when assigning a background image
    # to a ImgList, then the blurring amount of the BG image is automatically
```

(continues on next page)

(continued from previous page)

```
    # set to the current blurring level of the list, and if the latter is
    # zero, then the BG image is blurred using filter width=1)
    onlist.bg_img = bg_onlist.current_img()
    offlist.bg_img = bg_offlist.current_img()

    # prepare plume background modelling setup in both lists
    onlist.bg_model.mode = 6
    onlist.bg_model.set_missing_ref_areas(onlist.current_img())
    onlist.bg_model.xgrad_line_startcol = 10
    offlist.bg_model.update(**onlist.bg_model.settings_dict())

    return onlist, offlist


def plot_retrieval_points_into_image(img):
    """Plot terrain distance retrieval lines into an image."""
    ax = img.show(vmin=-5e16, vmax=5e18, zlabel=r"$S_{SO2}$ [cm$^{-2}$]")
    ax.set_title("Retrieval lines")
    for line in USE_LINES:
        line.plot_line_on_grid(ax=ax, marker="", color=line.color,
                               lw=2, ls=line.linestyle)
    for x, y, dist in dil._add_points:
        plot(x, y, " or")
    return ax


# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    if not exists(CALIB_FILE):
        raise IOError("Calibration file could not be found at specified "
                      "location:\n %s\nYou might need to run example 6 first")

    close("all")
    pcs_line = PCS_LINE
    calib = DoasCalibData()
    calib.load_from_fits(CALIB_FILE)

    # create dataset and correct viewing direction
    ds = create_dataset_dilution()
    geom = find_view_dir(ds.meas_geometry)

    # get plume distance image
    pix_dists, _, plume_dists = geom.compute_all_integration_step_lengths()

    # Create dilution correction class
    dil = DilutionCorr(USE_LINES, geom, skip_pix=SKIP_PIX_LINES)

    # you can also manually add  a single pixel position in the image that is
    # used to retrieve topographic distances (this function allows you also to
    # set the distance to the feature manually, since sometimes the SRTM data-
    # set is incomplete or has too low resolution)
    dil.add_retrieval_point(700, 930)
    dil.add_retrieval_point(730, 607)

    # Determine distances to the two lines defined above (every 6th pixel)
    for line_id in dil.line_ids:
        dil.det_topo_dists_line(line_id)
```

(continues on next page)

(continued from previous page)

```
# Plot the results in a 3D map
basemap = dil.plot_distances_3d(alt_offset_m=10, axis_off=False)

# retrieve pixel distances for pixels on the line
# (for emission rate estimate)
pix_dists_line = pcs_line.get_line_profile(pix_dists)

# get pixel coordinates of PCS center position ...
col, row = pcs_line.center_pix

# ... and get uncertainty in plume distance estimate for the column
pix_dist_err = geom.pix_dist_err(col)

# Prepare on and off-band list for retrieval of extinction coefficients
onlist, offlist = prepare_lists(ds)

# Activate vignetting correction in both lists (required to extract
# measured intensities along the terrain features)
onlist.vigncorr_mode = True
offlist.vigncorr_mode = True

# get the vignetting corrected images
on_vigncorr = onlist.current_img()
off_vigncorr = offlist.current_img()

# estimate ambient intensity for both filters
ia_on = on_vigncorr.crop(AMBIENT_ROI, True).mean()
ia_off = off_vigncorr.crop(AMBIENT_ROI, True).mean()

# perform dilution anlysis and retrieve extinction coefficients (on-band)
ext_on, _, _, ax0 = dil.apply_dilution_fit(img=on_vigncorr,
                                           rad_ambient=ia_on,
                                           i0_min=I0_MIN,
                                           plot=True)

ax0.set_ylabel("Terrain radiances (on band)", fontsize=14)
ax0.set_ylim([0, 2500])

# perform dilution anlysis and retrieve extinction coefficients (off-band)
ext_off, i0_off, _, ax1 = dil.apply_dilution_fit(img=off_vigncorr,
                                                 rad_ambient=ia_off,
                                                 i0_min=I0_MIN,
                                                 plot=True)
ax1.set_ylabel("Terrain radiances (off band)", fontsize=14)
ax1.set_ylim([0, 2500])

# determine plume pixel mask from AA image
onlist.aa_mode = True

plume_pix_mask = onlist.get_thresh_mask(AA_THRESH)
plume_pix_mask[840:, :] = 0  # remove tree in lower part of the image
onlist.aa_mode = False

# assign the just retrieved extinction coefficients to the respective
# image lists
onlist.ext_coeffs = ext_on
```

(continues on next page)

(continued from previous page)

```
    offlist.ext_coeffs = ext_off

    # save the extinction coefficients into a txt file (re-used in example
    # script 12). They are stored as pandas.Series object in the ImgList
    onlist.ext_coeffs.to_csv(join(SAVE_DIR, "ex11_ext_scat_on.txt"))
    offlist.ext_coeffs.to_csv(join(SAVE_DIR, "ex11_ext_scat_off.txt"))

    # now activate automatic dilution correction in both lists
    # get dilution corrected on and off-band image
    onlist.dilcorr_mode = True
    offlist.dilcorr_mode = True

    # get current dilution corrected raw images (i.e. in intensity space)
    on_corr = onlist.this
    off_corr = offlist.this

    # now activate tau mode (note that dilution correction mode is still
    # active)
    onlist.tau_mode = True
    offlist.tau_mode = True

    # extract tau images
    tau_on_corr = onlist.this
    tau_off_corr = offlist.this

    # determine corrected SO2-CD image from the image lists
    so2_img_corr = calib(tau_on_corr - tau_off_corr)
    so2_img_corr.edit_log["is_tau"] = True  # for plotting
    so2_cds_corr = pcs_line.get_line_profile(so2_img_corr)

    (phi_corr,
     phi_corr_err) = pyplis.fluxcalc.det_emission_rate(
        cds=so2_cds_corr,
        velo=PLUME_VELO,
        pix_dists=pix_dists_line,
        cds_err=calib.err(),
        pix_dists_err=pix_dist_err)

    # determine uncorrected so2-CD image from the image lists
    offlist.dilcorr_mode = False
    onlist.dilcorr_mode = False

    # the "this" attribute returns the current list image (same as
    # method "current_img()")
    so2_img_uncorr = calib(onlist.this - offlist.this)

    # Retrieve column density profile along PCS in uncorrected image
    so2_cds_uncorr = pcs_line.get_line_profile(so2_img_uncorr)
    # Calculate flux and uncertainty
    (phi_uncorr,
     phi_uncorr_err) = pyplis.fluxcalc.det_emission_rate(
        cds=so2_cds_uncorr,
        velo=PLUME_VELO,
        pix_dists=pix_dists_line,
        cds_err=calib.err(),
        pix_dists_err=pix_dist_err)
```

(continues on next page)

(continued from previous page)

```
    # IMPORTANT STUFF FINISHED (below follow some plots)
    ax2 = plot_retrieval_points_into_image(so2_img_corr)
    pcs_line.plot_line_on_grid(ax=ax2, ls="-", color="g")
    ax2.legend(loc="best", framealpha=0.5, fancybox=True, fontsize=20)
    ax2.set_title("Dilution corrected AA image", fontsize=12)
    ax2.get_xaxis().set_ticks([])
    ax2.get_yaxis().set_ticks([])

    x0, y0, w, h = pyplis.helpers.roi2rect(AMBIENT_ROI)
    ax2.add_patch(Rectangle((x0, y0), w, h, fc="none", ec="c"))

    fig, ax3 = subplots(1, 1)
    ax3.plot(so2_cds_uncorr, ls="-", color="#ff33e3",
             label=r"Uncorr: $\Phi_{SO2}=$%.2f (+/- %.2f) kg/s"
                   % (phi_uncorr / 1000.0, phi_uncorr_err / 1000.0))
    ax3.plot(so2_cds_corr, "-g", lw=3,
             label=r"Corr: $\Phi_{SO2}=$%.2f (+/- %.2f) kg/s"
                   % (phi_corr / 1000.0, phi_corr_err / 1000.0))

    ax3.set_title("Cross section profile", fontsize=12)
    ax3.legend(loc="best", framealpha=0.5, fancybox=True, fontsize=12)
    ax3.set_xlim([0, len(pix_dists_line)])
    ax3.set_ylim([0, 5e18])
    ax3.set_ylabel(r"$S_{SO2}$ [cm$^{-2}$]", fontsize=14)
    ax3.set_xlabel("PCS", fontsize=14)
    ax3.grid()

    # also plot plume pixel mask
    ax4 = pyplis.Img(plume_pix_mask).show(cmap="gray", tit="Plume pixel mask")

    if SAVEFIGS:
        ax = [ax0, ax1, ax2, ax3, ax4]
        for k in range(len(ax)):
            ax[k].set_title("")  # remove titles for saving
            ax[k].figure.savefig(join(SAVE_DIR, "ex11_out_%d.%s"
                                      % (k, FORMAT)),
                                 format=FORMAT, dpi=DPI)
        basemap.ax.set_axis_off()
        basemap.ax.view_init(15, 345)
        basemap.ax.figure.savefig(join(SAVE_DIR, "ex11_out_5.%s" % FORMAT),
                                  format=FORMAT, dpi=DPI)

# IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt
        from os.path import basename

        npt.assert_array_equal([],
                               [])
```

(continues on next page)

(continued from previous page)

```
        npt.assert_allclose(actual=[],
                            desired=[],
                            rtol=1e-7)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

4.2.12 Example 12 - Emission rate analysis (Etna example data)

Perform emission rate analysis for the example data. The analysis is performed along one plume cross section (in the image center) and using three different plume velocity retrievals.

Code

```
# -*- coding: utf-8 -*-
#
# Pyplis is a Python library for the analysis of UV SO2 camera data
# Copyright (C) 2017 Jonas Gliss (jonasgliss@gmail.com)
#
# This program is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License a
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
"""Pyplis example script 12 - Etna emission rate retrieval.

This example import results from the previous examples, for instance the AA
image list including measurement geometry (ex 4), the DOAS calibration
information (which was stored as FITS file, see ex. 6) and the AA sensitivity
correction mask retrieved from the cell calibration and normalised to the
position of the DOAS FOV (ex 7). The emission rates are retrieved for three
different plume velocity retrievals: 1. using the global velocity vector
retrieved from the cross correlation algorithm (ex8), 2. using the raw output
of the optical flow Farneback algorithm (``flow_raw``) and 3. using the
histogram based post analysis of the optical flow field (``flow_histo``).
The analysis is performed using the EmissionRateAnalysis class which basically
checks the AA list and activates ``calib_mode`` (-> images are loaded as
calibrated gas CD images) and loops over all images to retrieve the emission
rates for the 3 velocity modes. Here, emission rates are retrieved along 1
exemplary plume cross section. This can be easily extended by adding additional
PCS lines in the EmissionRateAnalysis class using ``add_pcs_line``.
The results for each velocity mode and for each PCS line are stored within
EmissionRateResults classes.
"""
```

(continues on next page)

(continued from previous page)

```
from __future__ import (absolute_import, division)

from SETTINGS import check_version

from os.path import join, exists
from matplotlib.pyplot import close, show, GridSpec, figure, rc_context
from matplotlib.cm import get_cmap

import pyplis
# IMPORT GLOBAL SETTINGS
from SETTINGS import SAVEFIGS, SAVE_DIR, FORMAT, DPI, OPTPARSE, LINES

# IMPORTS FROM OTHER EXAMPLE SCRIPTS
from ex04_prep_aa_imglist import prepare_aa_image_list

rc_context({'font.size': '18'})

# Check script version
check_version()

PCS = LINES[0]

# If false, then only the working environment is initalised
DO_EVAL = True

# Dilution correction
DILCORR = True

# You can specify here if you only want a certain number of images analysed
START_INDEX = 0
STOP_INDEX = None  # 20


# SCRIPT OPTONS
PYRLEVEL = 1
PLUME_VELO_GLOB = 4.29  # m/s
PLUME_VELO_GLOB_ERR = 1.5
# applies multi gauss fit to retrieve local predominant displacement
# direction, if False, then the latter is calculated from 1. and 2. moment
# of histogram (Faster but more sensitive to additional peaks in histogram)
HISTO_ANALYSIS_MULTIGAUSS = True
# molar mass of SO2
MMOL = 64.0638  # g/mol
# minimum required SO2-CD for emission-rate retrieval
CD_MIN = 5e16

# activate background check mode, if True, emission rates are only
# retrieved  for images showing SO2-CDs within specified interval around
# zero in BG reference rectangle LOG_ROI_SKY (see above). This can be
# used to ensure that significant systematic errors are induced in case
# the plume background retrieval failed. The latter could, for instance
# happen, if, for instance a cloud moves through one of the background
# reference areas used to model the background (cf. example script 3)
REF_CHECK_LOWER = -5e16
REF_CHECK_UPPER = 5e16
REF_CHECK_MODE = True
```

(continues on next page)

(continued from previous page)

```
# the following ROI is in the upper right image corner, where no gas occurs in
# the time series. It is used to log mean, min and max for each analysed image
# this information can be used to check, whether the plume background retrieval
# worked well
LOG_ROI_SKY = [530, 30, 600, 100]  # correspond to pyrlevel 1

# RELEVANT DIRECTORIES AND PATHS

# DOAS calibration results from example script 6
CALIB_FILE = join(SAVE_DIR, "ex06_doascalib_aa.fts")

# Scattering extinction coeffcients from example script 11 (stored as txt)
EXT_ON = join(SAVE_DIR, "ex11_ext_scat_on.txt")
EXT_OFF = join(SAVE_DIR, "ex11_ext_scat_off.txt")

# AA sensitivity correction mask retrieved from cell calib in script 7
CORR_MASK_FILE = join(SAVE_DIR, "ex07_aa_corr_mask.fts")

# time series of predominant displacement vector from histogram analysis of
# optical flow field in ROI around the PCS line "young_plume" which is used
# here for the emission rate retrieval. These information is optional, and is
# calculated during the evaluation if not provided
RESULT_PLUMEPROPS_HISTO = join(SAVE_DIR, "ex09_plumeprops_young_plume.txt")

# SCRIPT FUNCTION DEFINITIONS


def plot_and_save_results(ana, line_id="young_plume", date_fmt="%H:%M"):

    # plot colors for different optical flow retrievals
    cmap = get_cmap("Oranges")

    c_optflow_hybrid = cmap(255)
    c_optflow_histo = cmap(175)
    c_optflow_raw = cmap(100)

    fig = figure(figsize=(16, 12))
    gs = GridSpec(4, 1, height_ratios=[.6, .2, .2, .2], hspace=0.05)
    ax3 = fig.add_subplot(gs[3])
    ax0 = fig.add_subplot(gs[0], sharex=ax3)
    ax1 = fig.add_subplot(gs[1], sharex=ax3)
    ax2 = fig.add_subplot(gs[2], sharex=ax3)
    ax1.yaxis.tick_right()
    ax1.yaxis.set_label_position("right")
    ax3.yaxis.tick_right()
    ax3.yaxis.set_label_position("right")

    # Get emission rate results for the PCS line
    res0 = ana.get_results(line_id=line_id, velo_mode="glob")
    res1 = ana.get_results(line_id=line_id, velo_mode="flow_raw")
    res2 = ana.get_results(line_id=line_id, velo_mode="flow_histo")
    res3 = ana.get_results(line_id=line_id, velo_mode="flow_hybrid")

    res0.save_txt(join(SAVE_DIR, "ex12_flux_velo_glob.txt"))
    res1.save_txt(join(SAVE_DIR, "ex12_flux_flow_raw.txt"))
    res2.save_txt(join(SAVE_DIR, "ex12_flux_flow_histo.txt"))
    res3.save_txt(join(SAVE_DIR, "ex12_flux_flow_hybrid.txt"))
```

(continues on next page)

(continued from previous page)

```
    # Plot emission rates for the different plume speed retrievals
    res0.plot(yerr=True, date_fmt=date_fmt, ls="-", ax=ax0,
              color="c", ymin=0, alpha_err=0.08)
    res1.plot(yerr=False, ax=ax0, ls="-", color=c_optflow_raw, ymin=0)
    res2.plot(yerr=False, ax=ax0, ls="--", color=c_optflow_histo, ymin=0)
    res3.plot(yerr=True, ax=ax0, lw=3, ls="-", color=c_optflow_hybrid, ymin=0)

    # ax[0].set_title("Retrieved emission rates")
    ax0.legend(loc='best', fancybox=True, framealpha=0.5, fontsize=12)
    ax0.grid()

    # Plot effective velocity retrieved from optical flow histogram analysis
    res3.plot_velo_eff(ax=ax1, date_fmt=date_fmt, color=c_optflow_hybrid)
    # ax[1].set_title("Effective plume speed
    #                   (from optflow histogram analysis)")
    ax1.set_ylim([0, ax1.get_ylim()[1]])

    # Plot time series of predominant plume direction (retrieved from optical
    # flow histogram analysis and stored in object of type LocalPlumeProperties
    # which is part of plumespeed.py module
    ana.pcs_lines[line_id].plume_props.plot_directions(ax=ax2,
                                                       date_fmt=date_fmt,
                                                       color=c_optflow_hybrid)

    ax2.set_ylim([-180, 180])
    pyplis.helpers.rotate_xtick_labels(ax=ax2)
    ax0.set_xticklabels([])
    ax1.set_xticklabels([])
    ax2.set_xticklabels([])
    # tight_layout()

    ax3 = ana.plot_bg_roi_vals(ax=ax3, date_fmt="%H:%M")
    # gs.tight_layout(fig, h_pad=0)#0.03)
    gs.update(hspace=0.05, top=0.97, bottom=0.07)
    return fig


# SCRIPT MAIN FUNCTION
if __name__ == "__main__":
    close("all")
    figs = []
    if not exists(CALIB_FILE):
        raise IOError("Calibration file could not be found at specified "
                      "location:\n%s\nPlease run example 6 first")
    if not exists(CORR_MASK_FILE):
        raise IOError("Cannot find AA correction mask, please run example"
                      "script 7 first")

    # convert the retrieval line to the specified pyramid level (script option)
    pcs = PCS.convert(to_pyrlevel=PYRLEVEL)

    # now try to load results of optical flow histogram analysis performed for
    # this line in script no. 9. and assign them to the pcs line. This has the
    # advantage, that  missing velocity vectors (i.e. from images where optical
    # flow analysis failed) can be interpolated. It is, however, not
    # necessarily required to do this in advance. In the latter case the
```

(continues on next page)

(continued from previous page)

```
    # emission rates show gaps at all images, where the optical flow was
    # considered not reliable
    try:
        p = pyplis.LocalPlumeProperties()
        p.load_txt(RESULT_PLUMEPROPS_HISTO)
        p = p.to_pyrlevel(PYRLEVEL)
        fig = p.plot(color="r")
        # p.interpolate()
        # p = p.apply_significance_thresh(0.2).interpolate()
        # p = p.apply_median_filter(3).apply_gauss_filter(2)
        fig = p.plot(date_fmt="%H:%M", fig=fig)

        pcs.plume_props = p
    except BaseException:
        print("Local plume properties could not be loaded and will be "
              "calculated during the emission rate analysis")

    # Load AA list
    # includes viewing direction corrected geometry
    aa_list = prepare_aa_image_list()

    aa_list.pyrlevel = PYRLEVEL

    if DILCORR:
        aa_list.import_ext_coeffs_csv(EXT_ON)
        aa_list.get_off_list().import_ext_coeffs_csv(EXT_OFF)

    # Load DOAS calbration data and FOV information (see example 6)
    doascalib = pyplis.doascalib.DoasCalibData()
    doascalib.load_from_fits(file_path=CALIB_FILE)
    doascalib.fit_calib_data()

    # Load AA corr mask and set in image list(is normalised to DOAS FOV see
    # ex7)
    aa_corr_mask = pyplis.Img(CORR_MASK_FILE)

    aa_list.senscorr_mask = aa_corr_mask

    # set DOAS calibration data in image list
    aa_list.calib_data = doascalib

    ana = pyplis.EmissionRateAnalysis(
        imglist=aa_list,
        bg_roi=LOG_ROI_SKY,
        pcs_lines=pcs,
        velo_glob=PLUME_VELO_GLOB,
        velo_glob_err=PLUME_VELO_GLOB_ERR,
        ref_check_lower_lim=REF_CHECK_LOWER,
        ref_check_upper_lim=REF_CHECK_UPPER,
        velo_dir_multigauss=HISTO_ANALYSIS_MULTIGAUSS,
        senscorr=True,
        dilcorr=DILCORR)

    ana.settings.ref_check_mode = REF_CHECK_MODE

    ana.settings.velo_modes["flow_raw"] = 1
    ana.settings.velo_modes["flow_histo"] = True
```

(continues on next page)

(continued from previous page)

```
    ana.settings.velo_modes["flow_hybrid"] = 1
    ana.settings.min_cd = CD_MIN

    # plot all current PCS lines into current list image (feel free to define
    # and add more PCS lines above)
    ax = ana.plot_pcs_lines(
        vmin=-
        5e18,
        vmax=6e18,
        tit="Dilution corr: %s" %
        DILCORR)
    ax = ana.plot_bg_roi_rect(ax=ax, to_pyrlevel=PYRLEVEL)
    figs.append(ax.figure)

    if not DO_EVAL:
        aa_list.dilcorr_mode = not DILCORR
        aa_list.show_current(
            vmin=-
            5e18,
            vmax=6e18,
            tit="Dilution corr: %s" %
            (not DILCORR))
        # you can check the settings first
        print(ana.settings)
        # check if optical flow works
        ana.imglist.optflow_mode = True
        aa_mask = ana.imglist.get_thresh_mask(CD_MIN)
        ana.imglist.optflow.plot_flow_histograms(line=pcs, pix_mask=aa_mask)

    else:
        ana.run_retrieval(start_index=START_INDEX,
                          stop_index=STOP_INDEX)

        figs.append(plot_and_save_results(ana))
        # the EmissionRateResults class has an informative string
        # representation
        print(ana.get_results("young_plume", "flow_histo"))

    if SAVEFIGS:
        for k in range(len(figs)):
            figs[k].savefig(join(SAVE_DIR, "ex12_out_%d.%s" % (k + 1, FORMAT)),
                            format=FORMAT, dpi=DPI)

    # IMPORTANT STUFF FINISHED (Below follow tests and display options)

    # Import script options
    (options, args) = OPTPARSE.parse_args()

    # If applicable, do some tests. This is done only if TESTMODE is active:
    # testmode can be activated globally (see SETTINGS.py) or can also be
    # activated from the command line when executing the script using the
    # option --test 1
    if int(options.test):
        import numpy.testing as npt
        from os.path import basename

        npt.assert_array_equal([],
```

(continues on next page)

(continued from previous page)

```
                                  [])

        npt.assert_allclose(actual=[],
                            desired=[],
                            rtol=1e-7)
        print("All tests passed in script: %s" % basename(__file__))
    try:
        if int(options.show) == 1:
            show()
    except BaseException:
        print("Use option --show 1 if you want the plots to be displayed")
```

CHAPTER 5

API

Note: The code documentation is currently in the process of being changed to the NumPy standard.

5.1 Setup classes

Setup classes to specify relevant parameters for the emission-rate analysis.

The most important ones are:

1. *Source*: emission source specifications
2. *Camera*: camera specifications
3. *MeasSetup*: full measurement setup

class pyplis.setupclasses.**Source**(*name=”*, *info_dict=None*, ***kwargs*)
Object containing information about emission source.

name
string ID of source

Type str

lon
longitude of source

Type float

lat
latitude of source

Type float

altitude
altitude of source

Type float

suppl_info
dictionary containing supplementary information (e.g. source type, region, country)

Type dict

Parameters

- **name** (*str*) – string ID of source (default is "")
- **info_dict** (*dict*) – dictionary contatining source information (is only loaded if all necessary parameters are available and in the right format)

Note: If input param `name` is a valid default ID (e.g. "Etna") then the source information is extracted from the database and the parameter `info_dict` is ignored.

__init__(*name=", info_dict=None, **kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

source_id
Get ID of source.

Returns `self.name`

Return type str

info_available
Check if main information is available.

geo_data
Return dictionary containing lon, lat and altitude.

to_dict()
Return dictionary of all parameters.

Returns dictionary representation of class

Return type dict

load_source_info(*name=None, try_online=True*)
Try to load source info from external database.

Try to find source info in pyplis database file my_sources.txt and if it cannot be found there, try online, if applicable.

Parameters

- **name** (*str*) – if provided, a volcano with the corresponding name is searched. If not provided, the current name is used
- **try_online** (*bool*) – if True, online search is attempted in case information cannot be found in my_sources.txt

save_to_database()
Save the current information as a new source.

The information is stored in the *my_sources.txt* file that can be found in the pyplis installation folder *my_pyplis*

get_info(*name=None*, *try_online=True*)
Load source info from database.

Looks if desired source (specified by argument *name*) can be found in the *my_sources.txt* file and if not, tries to find information about the source online (if **:param:'try_online'** is True)

Parameters

- **name** (*str*) – source ID
- **try_online** (*bool*) – if True, also search online database

Returns Dictionary containing source information

Return type dict

class pyplis.setupclasses.**FilterSetup**(*filter_list=None*, *default_key_on=None*, *default_key_off=None*, ***filters*)
A collection of *pyplis.utils.Filter* objects.

This collection specifies a filter setup for a camera. A typical setup would be one on and one off band filter. An instance of this class is created automatically as an attribute of *Camera* objects.

Parameters

- **filters** (*list*) – list of *pyplis.utils.Filter* objects specifying camera filter setup
- **default_key_on** (*str*) – string ID of default on band filter (only relevant if collection contains more than one on band filter)
- **default_key_off** (*str*) – string ID of default off band filter (only relevant if collection contains more than one off band filter)

__init__(*filter_list=None*, *default_key_on=None*, *default_key_off=None*, ***filters*)
x.__init__(...) initializes x; see help(type(x)) for signature

filters
Get dict containing filters (only getter, for backwards compat).

on_band
Return default on band filter.

off_band
Return default on band filter.

ids_off
List with all offband filter ids.

ids_on
List with all onband filter ids.

default_key_on
Return default onband key.

default_key_off
Return default offband key.

has_on
Check if collection contains an onband filter.

has_off
Check if collection contains an onband filter.

number_of_filters
Return the current number of filters in this collection.

init_filters(*filter_list=None*, ***filters*)
Initialize the filter collection (old settings will be deleted).

The filters will be written into the dictionary `self._filters` in the list order, keys are the filter ids

Parameters

- **filters** (*list*) – list of `pyplis.utils.Filter` objects specifying camera filter setup
- ****filters** – pairs of filter IDs and instances of `Filter` that may be used instead of (or in addition to) input *filter_list*

update_filters_from_dict(*filter_dict*)
Add filter objects from a dictionary.

Parameters **filter_dict** (*dict*) – dictionary, containing filter information

set_default_filter_keys(*default_key_on=None*, *default_key_off=None*)
Deprecated.

check_default_filters()
Check if default filter keys are set.

get_ids_on_off()
Get all filters sorted by their type (On or Off).

Returns

2-element tuple containing

- list, contains all on band IDs
- list, contains all off band IDs

Return type tuple

print_setup()
Print the current setup.

Returns print string representation

Return type str

class pyplis.setupclasses.**Camera**(*cam_id=None*, *filter_list=None*, *default_filter_on=None*, *default_filter_off=None*, *ser_no=9999*, ***geom_info*)
Base class to specify a camera setup.

Class representing a UV camera system including detector specifications, optics, file naming convention and the bandpass filters that are equipped with the camera (managed via an instance of the `FilterSetup` class).

Parameters

- **cam_id** (*str*) – camera ID (e.g "ecII"), if this ID corresponds to one of the default cameras, the information is automatically loaded from supplementary file *cam_info.txt*
- **filter_list** (*list*) – list containing `pyplis.utils.Filter` objects specifying the camera filter setup. If unspecified (empty list) and input param `cam_id` is a valid default ID, then the default filter setup of the camera will be loaded.
- **default_filter_on** (*str*) – string ID of default on band filter (only relevant if collection contains more than one on band filter)
- **default_filter_off** (*str*) – string ID of default off band filter (only relevant if collection contains more than one off band filter)
- **ser_no** (*int*) – optional, camera serial number

- ****geom_info** – additional keyword args specifying geometrical information, e.g. lon, lat, altitude, elev, azim

Examples

Example creating a new camera (using ECII default info with custom filter setup):

```
import pyplis

#the custom filter setup
filters= [pyplis.utils.Filter(type="on", acronym="F01"),
          pyplis.utils.Filter(type="off", acronym="F02")]

cam = pyplis.setupclasses.Camera(cam_id="ecII", filter_list=filters,
                                  lon=15.11, lat=37.73, elev=18.0,
                                  elev_err=3, azim=270.0,
                                  azim_err=10.0, focal_lengh=25e-3)
print cam
```

__init__(*cam_id=None*, *filter_list=None*, *default_filter_on=None*, *default_filter_off=None*, *ser_no=9999*, ***geom_info*)

Init object.

Parameters

- **cam_id** (*str*) – string ID of camera (e.g. "ecII")
- **info_dict** (*dict*) – dictionary containing camera info (only loaded if all parameters are available in the right format)

Note: if input cam_id is valid (i.e. can be found in database) then any additional input using `info_dict` is ignored.

lon

Camera longitude.

lat

Camera latitude.

altitude

Camera altitude in m.

Note: This is typically the local topography altitude, which can for instance be accessed automatically based on camera position (lat, lon) using *get_altitude_srtm()*. Potential offsets (i.e. elevated positioning due to tripod or measurement from a house roof) can be specified using *alt_offset*.

elev

Return viewing elevation angle (center pixel) in degrees.

0 refers to horizon, 90 to zenith

elev_err

Uncertainty in viewing elevation angle in degrees.

azim

Return viewing azimuth angle in deg relative to north (center pixel).

azim_err
Uncertainty in viewing azimuth angle in degrees.

alt_offset
Height of camera position above topography in m.

This offset can be added in case the camera is positioned above the ground and is only required if **:param:'altitude'** corresponds to the topographic elevation

update_settings(***settings*)
Call for *update()* (old name).

load_default(*cam_id*)
Redefinition of method from base class `CameraBaseInfo`.

update(***settings*)
Update camera parameters.

Parameters settings (*dict*) – dictionary containing camera parametrs (valid keys are all keys of `self.__dict__` and from dictionary `self.geom_data`)

get_altitude_srtm()
Try load camera altitude based on lon, lat and SRTM topo data.

Note: Requires `geonum` package to be installed and *lon* and *lat* to be set.

prepare_filter_setup(*filter_list=None*, *default_key_on=None*, *default_key_off=None*)
Create *FilterSetup* object.

This method defines the camera filter setup based on an input list of `Filter` instances.

Parameters

- **filter_list** (*list*) – list containing *pyplis.utils.Filter* objects
- **default_filter_on** (*str*) – string specifiying the string ID of the main onband filter of the camera (usually "on"). If unspecified (None), then the ID of the first available on bandfilter in the filter input list will be used.
- **default_filter_off** (*str*) – string specifiying the string ID of the main offband filter of the camera (usually "on"). If unspecified (None), then the ID of the first available off band filter in the filter input list will be used.

to_dict()
Convert this object into a dictionary.

change_camera(*cam_id=None*, *make_new=False*, ***kwargs*)
Change current camera type.

Parameters

- **cam_id** (*str*) – ID of new camera
- **make_new** (*bool*) – if True, a new instance will be created and returned
- ****kwargs** – additional keyword args (see *__init__()*)

Returns either this object (if **:param:'make_new'** is False) or else, new instance

Return type *Camera*

dx_to_decimal_degree(*pix_num_x*)
Convert horizontal distance (in pixel units) into angular range.

Parameters **pix_num_x** (*int*) – number of pixels for which angular range is determined

Returns dx in units of decimal degrees

Return type float

dy_to_decimal_degree(*pix_num_y*)
Convert vertical distance (in pixel units) into angular range.

Parameters **pix_num_y** (*int*) – number of pixels for which angular range is determined

Returns dy in units of decimal degrees

Return type float

class pyplis.setupclasses.**FormSetup**(*line_dict=None*, *rect_dict=None*)
Setup class for all forms (lines, rectangles etc.) used for evaluation.

__init__(*line_dict=None*, *rect_dict=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

class pyplis.setupclasses.**BaseSetup**(*base_dir*, *start*, *stop*, ***opts*)
Abstract base class for basic measurement setup.

Specifies image base path and start / stop time stamps of measurement as well as the following boolean access flags:

1. *USE_ALL_FILES*
2. *SEPARATE_FILTERS*
3. *USE_ALL_FILE_TYPES*
4. *INCLUDE_SUB_DIRS*
5. *ON_OFF_SAME_FILE*
6. *LINK_OFF_TO_ON*
7. *REG_SHIFT_OFF*

Parameters

- **base_dir** (*str*) – Path were e.g. imagery data lies
- **start** (*datetime*) – start time of Dataset (can also be datetime.time)
- **stop** (*datetime*) – stop time of Dataset (can also be datetime.time)
- ****opts** – setup options for file import (see specs above)

__init__(*base_dir*, *start*, *stop*, ***opts*)
x.__init__(...) initializes x; see help(type(x)) for signature

start
Start time of setup.

stop
Stop time of setup.

USE_ALL_FILES
File import option (boolean).

If True, all files in image base folder are used (i.e. start / stop time stamps are disregarded)

SEPARATE_FILTERS
File import option (boolean).

If true, files are separated by filter type (e.g. "on", "off")

USE_ALL_FILE_TYPES
File import option (boolean).

If True, all files found are imported, disregarding the file type (i.e. if image file type is not specified. It is strongly recommended NOT to use this option)

INCLUDE_SUB_DIRS
File import option (boolean).

If True, sub directories are included into image search

ON_OFF_SAME_FILE
File import option (boolean).

If True, it is assumed, that each image file contains both on and offband images. In this case, both the off and the onband image lists are filled with the same file paths. Which image to load in each list is then handled within the `ImgList`itself on :func:`load` using the attribute `list_id` which is passed using the key `filter_id` to the respective customised image import method that has to be defined in the `custom_image_import` file of the pyplis installation and linked to your Camera settings in the `cam_info.txt` file which can be found in the data directory of the installation.

An example for such a file convention is the SO2 camera from CVO (USGS) See e.g. `load_usgs_multifits()` in `custom_image_import`.

LINK_OFF_TO_ON
File import option (boolean).

If True, the offband ImgList is automatically linked to the onband list on initiation of a `Dataset` object.

REG_SHIFT_OFF
File import option (boolean).

If True, the images in an offband image list that is linked to an onband image list (cf. *LINK_OFF_TO_ON*) are shifted using the registration offset specified in the `reg_shift_off` attribute of the *Camera* instance.

check_timestamps()
Check if timestamps are valid and set to current time if not.

base_info_check()
Check if all necessary information if available.

Checks if path and times are valid

Returns

- *2-element tuple, containing*
- *- bool, True or False*
- *- str, information*

class pyplis.setupclasses.**MeasSetup**(*base_dir=None*, *start=None*, *stop=None*, *camera=None*, *source=None*, *wind_info=None*, *cell_info_dict=None*, *rects=None*, *lines=None*, *auto_topo_access=True*, ***opts*)

Setup class for plume image data.

In this class, everything related to a full measurement setup is defined. This includes the image base directory, start / stop time stamps (if applicable), specifications of the emission source (i.e. *Source* object), camera specifications (i.e. *Camera* object) as well as meteorology information (i.e. wind direction and velocity). The latter is not represented as an own class in Pyplis but is stored as a Python dictionary. *MeasSetup* objects are the default input for *pyplis.dataset.Dataset* objects (i.e. also *pyplis.cellcalib.CellCalibEngine*).

Parameters

- **base_dir** (*str*) – Path were e.g. imagery data lies
- **start** (*datetime*) – start time of Dataset (may as well be datetime.time)
- **stop** (*datetime*) – stop time of Dataset (may as well be datetime.time)
- **camera** (Camera) – general information about the camera used
- **source** (Source) – information about emission source (e.g. lon, lat, altitude)
- ****opts** – setup options for file handling (currently only INCLUDE_SUB_DIRS option)

__init__(*base_dir=None, start=None, stop=None, camera=None, source=None, wind_info=None, cell_info_dict=None, rects=None, lines=None, auto_topo_access=True, **opts*)
x.__init__(...) initializes x; see help(type(x)) for signature

source
Emission source.

camera
Camera.

update_wind_info(*info_dict*)
Update wind info dict using valid entries from input dict.

Parameters info_dict (*dict*) – dictionary containing wind information

base_info_check()
Check if all req. info is available.

check_geometry_info()
Check if all req. info for measurement geometry is available.

Relevant parameters are:

1. **Lon, Lat of**
   i. source
   ii. camera
2. **Meteorology info**
   i. Wind direction
   ii. Wind velocity (rough estimate)
3. **Viewing direction of camera**
   i. Azimuth (N)
   ii. Elvation(from horizon)
4. Alitude of camera and source
5. **Camera optics**
   i. Pixel size
   ii. Number of pixels detector
   iii. focal length

update_meas_geometry()
Update the meas geometry based on current settings.

short_str()
Return a short info string.

5.2 Data Set object

Assorted data import functionality.

The `Dataset` object is doing all sorts of stuff related to the general data import setup, for instance the automated separation of image files by their type (e.g. on-band, off-band, dark, offset) using information from a file naming convention specified within a `Camera` object. For more information how to customise your data import see `pyplis.setupclasses` or read this little introductory tutorial

class pyplis.dataset.**Dataset**(*input=None*, *lst_type=<class 'pyplis.imagelists.ImgList'>*, *init=1*)
Class for data import management.

Default input is a `pyplis.setupclasses.MeasSetup` object, which specifies the camera used (e.g. file naming convention, detector specifics) the measurement geometry and information about the source and meteorological wind direction, start / stop time stamps and the image base directory.

setup
class containing measurement setup information

Type *MeasSetup*

lst_type
default type of ImgList objects (cf. `CellCalibEngine`)

Type object

lists_access_info
dictionary filled on data import based on camera specifications. Used to map API IDs of filters and dark / offset information (e.g. "on", "off", "dark0") onto internal list keys. It is normally not required to use this dictionary or apply changes to it. For EC2 camera standard this will look like:

```
>>> self.lists_access_info
OrderedDict([('on', ['F01', 'F01']),
             ('off', ['F02', 'F02']),
             ('offset0', ['D0L', 'D0L']),
             ('dark0', ['D1L', 'D1L']),
             ('offset1', ['D0H', 'D0H']),
             ('dark1', ['D1H', 'D1H'])])
```

Type OrderedDict

Parameters

- **input** – Usable `input` includes `MeasSetup` instance or a valid image directory. `input` is passed to `load_input()`.
- **lst_type** (*object*) – default type of image list objects (e.g. `ImgList`, `CellImgList`), defaults to `ImgList`.
- **init** (*bool*) – init

__init__(*input=None*, *lst_type=<class 'pyplis.imagelists.ImgList'>*, *init=1*)
x.__init__(...) initializes x; see help(type(x)) for signature

camera
Return camera base info object.

source
Get / set current Source.

cam_id
Return current camera ID.

base_dir
Getter / setter of current image base_dir.

USE_ALL_FILES
Return USE_ALL_FILES boolen from setup.

USE_ALL_FILE_TYPES
Return USE_ALL_FILE_TYPES option from setup.

INCLUDE_SUB_DIRS
Return boolean sub directory inclusion option.

LINK_OFF_TO_ON
I/O option defined in `BaseSetup`.

start
Getter / setter for current start time stamp.

stop
Getter / setter for current stop time stamp.

file_type
Return current image file type.

meas_geometry
Return current measurement geometry.

filters
Return the current filter setup.

filter_acronyms
Make a dictionary of filter IDs and corresponding acronyms.

num_of_filters
Return the number of filters in `self.filters`.

rects
Return rectangle collection.

lines
Return rectangle collection.

load_input (*input*)
Extract information from input and set / update self.setup.

Parameters **input** – Usable `input` includes `MeasSetup` instance or a valid image directory.

Returns `True`, if input could be utilised, `False` if not

Return type bool

set_setup (*stp*)
Set the current measurement setup.

Parameters **stp** (`MeasSetup`) – Class containing information about measurement setup

init_image_lists()
Create and fill image lists.

create_lists_default()
Initialize of default lists (if camera specs not available).

create_lists_cam()
Initialize of all image lists, old lists are deleted.

fill_image_lists()
Import all images and fill image list objects.

get_all_filepaths()
Find all valid image filepaths in current base directory.

Returns list containing all valid image file paths (Note, that these include all files found in the folder(s) in case the file type is not explicitely set in the camera class.)

Return type list

check_filename_info_access(*filepath*)
Check which information can be accessed from file name.

The access test is performed based on the filename access information specified in the `Camera` object of the measurement setup

Parameters **filepath** (*str*) – valid file path of an example image

Returns Dictionary containing information about which meta inforamtion could be identified from the image file path based on the current camera

Return type dict

change_img_base_dir(*img_dir*)
Set or update the current base_dir.

Parameters **p** (*str*) – new path

extract_files_time_ival(*all_paths*)
Extract all files belonging to specified time interval.

Parameters **all_paths** (*list*) – list of image filepaths

find_closest_img(*filename, in_list, acronym, meas_type_acro*)
Find closest-in-time image to input image file.

Parameters

- **filename** (*str*) – image filename
- **in_list** (*str*) – input list with filepaths
- **acronym** (*str*) – the acronym of the image type to be searched (e.g. an acronym for a dark image as specified in camera)
- **meas_type_acro** (*str*) – meas type acronym of image type to be searched (e.g. an acronym for a dark image as specified in camera)

all_lists()
Return list containing all available image lists.

Loops over `self._lists_intern` and the corresponding sub directories

dark_ids
Get all dark IDs.

assign_dark_offset_lists(*into_list=None*)
Assign dark and offset lists to image lists `self.lists`.

Assign dark and offset lists in filter lists for automatic dark and offset correction. The lists are set dependent on the read_gain mode of the detector

Parameters **into_list (None)** (`ImgList`) – optional input, if specified, the dark assignment is performed only in the input list

get_all_dark_offset_lists()
Get all dark and offset image lists.

dark_lists
Call and return *get_all_dark_offset_lists()*.

dark_lists_with_data
Return all dark/offset lists that include image data.

filter_ids
Get all dark IDs.

get_all_image_lists()
Get all image lists (without dark and offset lists).

img_lists
Wrap *get_all_image_lists()*.

img_lists_with_data
Wrap *get_all_image_lists()*.

check_dark_lists()
Check all dark lists whether they contain images or not.

find_master_dark(*dark_list*)
Search master dark image for a specific dark list.

Search a master dark image for all dark image lists that do not contain images

find_master_darks(*dark_ids=None*)
Search master dark image for dark image lists.

Search a master dark image for all dark image lists that do not contain images.

check_image_access_dark_lists()
Check whether dark and offset image lists contain at least one img.

images_available(*filter_id*)
Check if image list has images.

Parameters **filter_id** (*str*) – string (filter) ID of image list

current_image(*filter_id*)
Get current image of image list.

Parameters **filter_id** (*str*) – filter ID of image list

get_list(*list_id*)
Get image list for one filter.

Parameters **filter_id** (*str*) – filter ID of image list (e.g. "on")

get_current_img_prep_dict(*list_id=None*)
Get the current image preparation settings from one image list.

Parameters **list_id** (*str*) – ID of image list

load_images()
Load the current images in all image lists.

Note: Gives warning for lists containing no images

update_image_prep_settings(***settings*)
Update image preparation settings in all image lists.

update_times(*start*, *stop*)
Update start and stop times of this dataset and reload.

Parameters

- **start** (*datetime*) – new start time
- **stop** (*datetime*) – new stop time

duplicate()
Duplicate Dataset object.

show_current_img(*filter_id*, *add_forms=False*)
Plot current image.

Parameters **filter_id** (*str*) – filter ID of image list (e.g. "on")

plot_mean_value(*filter_id*, *yerr=1*, *rect=None*)
Plot the pixel mean value of specified filter.

Only pixel values in the time span covered by this dataset are used.

draw_map_2d(**args*, ***kwargs*)
Call and return *draw_map_2d()* of `self.meas_geometry`.

draw_map_3d(**args*, ***kwargs*)
Call and return *draw_map_3d()* of `self.meas_geometry`.

print_list_info()
Print overview information about image lists.

connect_meas_geometry()
Set pointer to current measurement geometry within image lists.

plot_tau_preview(*on_id='on'*, *off_id='off'*, *pyrlevel=2*)
Plot a preview of current tau_on, tau_off and AA images.

AA is plotted twice in 1st row of subplots in 2 diffent value ranges.

Parameters

- **on_id** (*str*) – string ID of onband filter ("on")
- **off_id** (*str*) – string ID of offband filter ("off")
- **pyrlevel** – provide any integer here to reduce the image sizes using a gaussian pyramide approach (2)

5.3 Geometrical calculations

Module containing functionality for all relevant geometrical calculations.

class pyplis.geometry.**MeasGeometry**(*source_info=None*, *cam_info=None*, *wind_info=None*, *auto_topo_access=True*)

Class for calculations and management of the measurement geometry.

All calculations are based on provided information about camera (stored in dictionary `_cam`, check e.g. `self._cam.keys()` for valid keys), source (stored in dictionary `_source`, check e.g. `self._source.keys()` for valid keys) and meteorological wind direction (stored in dictionary `_wind`). The keys of these dictionaries (i.e. identifiers for the variables) are the same as the corresponding attributes in the respective classes `pyplis.Camera` and `pyplis.Source`. If you want to change these parameters, it is recommended to use the correpdonding update methods `update_cam_specs()`, `update_source_specs()` and `update_wind_specs()` or use the provided getter / setter methods for each parameter (e.g., `cam_elev` for key `elev` of `_cam` dictionary, `cam_azim` for key `azim` of `_cam` dictionary, `cam_lon` for key `lon` of `_cam` dictionary, `cam_lat` for key `lat` of `_cam` dictionary, `source_lon` for key `lon` of `_source` dictionary, `source_lat` for key `lat` of `_source` dictionary, `wind_dir` for key `dir` of `_dir` dictionary.

Note that in the dictionary based update methods `update_cam_specs()`, `update_source_specs()` and `update_wind_specs()`, the dict keys are supposed to be inserted, e.g.:

```
geom = MeasGeometry()
# either update using valid keywords as **kwargs ...
geom.update_cam_specs(lon=10, lat=20, elev=30, elev_err=0.5)

# ... or update using a dictionary containing camera info, e.g.
# retrieved from an existing camera ...
cam = pyplis.Camera(altitude=1234, azim=270, azim_err=10)
cam_dict = cam.to_dict()

geom.update_cam_specs(cam_dict)

# ... or directly using the getter / setter attributes
print geom.cam_altitude #1234 (value of geom._cam["altitude"])

geom.cam_altitude=111
print geom.cam_altitude #111 (new value of geom._cam["altitude"])

# analogous with source and wind

# This ...
geom.update_wind_specs(dir=180, dir_err=22)

# ... is the same as this:
geom.wind_dir=180
geom.wind_dir_err=22

# load Etna default source info
source = pyplis.Source("etna")
geom.update_source_specs(**source.to_dict())
```

The latter by default also update the most important attribute of this class `geo_setup` which is an instance of the `geonum.GeoSetup` class and which is central for all geometrical calculations (e.g. camera to plume distance).

geo_setup

class containing information about the current measurement setup. Most of the relevant geometrical calculations are performed within this object

Type GeoSetup

_source

dictionary containing information about emission source (valid keys: `name, lon, lat, altitude`)

Type dict

_wind
dictionary containing information about meteorology at source position (valid keys: `dir, dir_err, velo, velo_err`)

Type dict

_cam
dictionary containing information about the camera (valid keys: `cam_id, serno, lon, lat, altitude, elev, elev_err, azim, azim_err, focal_length, pix_width, pix_height, pixnum_x, pixnum_y altitude_offs`

Type dict

Parameters

- **source_info** (*dict*) – dictionary containing source parameters (see *source* for valid keys)
- **cam_info** (*dict*) – dictionary containing camera parameters (see *cam* for valid keys)
- **wind_info** (*dict*) – dictionary conatining meteorology information (see `wind` for valid keys)

__init__(*source_info=None, cam_info=None, wind_info=None, auto_topo_access=True*)
x.__init__(...) initializes x; see help(type(x)) for signature

cam_id
ID of current camera (string).

cam_serno
Return serial number of camera.

cam_lon
Longitude position of camera.

cam_lat
Latitude position of camera.

cam_altitude
Altitude of camera position.

cam_elev
Elevation angle of camera viewing direction (CFOV).

cam_elev_err
Elevation angle error of camera viewing direction (CFOV).

cam_azim
Azimuth of camera viewing direction (CFOV).

cam_azim_err
Azimuth error of camera viewing direction (CFOV).

cam_focal_length
Focal length of camera.

cam_pix_width
Pixel width of camera detector (horizonzal pix-to-pix distance).

cam_pix_height
Pixel height of camera detector (vertical pix-to-pix distance).

cam_pixnum_x
Return Number of camera detector pixels in x-direction (horizontal).

cam_pixnum_y
Return Number of camera detector pixels in y-direction (vertical).

cam_altitude_offs
Camera elevation above topography.

Note: This can be used as offset above the ground, if the camera altitude (`cam_altitude`) is retrieved based on local topography level (e.g. using automatic SRTM access based on camera lat and lon).

source_lon
Longitude position of source.

source_lat
Latitude position of source.

source_altitude
Altitude of source position.

wind_dir
Azimuth of wind direction.

wind_dir_err
Azimuth error of wind direction.

update_cam_specs(*info_dict=None*, *update_geosetup=True*, ***kwargs*)
Update camera settings.

Update dictionary containing geometrical camera information (`cam`) by providing a dictionary containing valid key / value pairs for camera parameters.

Parameters

- **info_dict** (`dict`) – dictionary containing camera information (see `cam` for valid keys)
- **update_geosetup** (`bool`) – If True, the method `update_geosetup()` is called at the end of this method
- ****kwargs** – can be used to directly pass valid key / value pairs

update_source_specs(*info_dict=None*, *update_geosetup=True*, ***kwargs*)
Update source settings.

Update source info dictionary (`source`) either by providing a dictionary containing valid key / value pairs (**:param:‘info_dict‘** or by providing valid key / value pairs directly using **:param:‘kwargs‘**)

Parameters

- **info_dict** (`dict`) – dictionary containing source information (see `source` for valid keys)
- **update_geosetup** (`bool`) – If True, the method `update_geosetup()` is called at the end of this method
- ****kwargs** – alternative way to update the source dictionary using valid keywords directly

update_wind_specs (*info_dict=None*, *update_geosetup=True*, ***kwargs*)
Update meteorological settings.

Update wind info dictionary (`wind`) either by providing a dictionary containing valid key / value pairs (**:param:'info_dict'** or by providing valid key / value pairs directly using **:param:'kwargs'**)

Parameters

- **info_dict** (*dict*) – dictionary containing meterology information (see `wind` for valid keys)
- **update_geosetup** (*bool*) – If True, the method *update_geosetup()* is called at the end of this method
- ****kwargs** – alternative way to update the wind dictionary using valid keywords directly

update_geosetup ()
Update the current GeoSetup object.

Note: The borders of the range are determined considering cam pos, source pos and the position of the cross section of viewing direction with plume

get_coordinates_imgborders ()
Get elev and azim angles corresponding to camera FOV.

horizon_analysis (*skip_cols=30*)
Search pixel coordinates of horizon for image columns.

The algorithm performs a topography analysis for a number of image columns. Elevation profiles are determined for each column (azimuth) and from those, the horizon elevation angle is searched. The retrieved values are returned in pixel coordinates.

Parameters **skip_cols** – distance between pixel columns for which the analysis is performed

Note: This is a Beta version, please report any problems

get_viewing_directions_line (*line*)
Determine viewing direction coords for a line in an image.

Parameters **line** (`LineOnImage`) – line on image object

Returns

4-element tuple containing

- 1-d array containing azimuth angles of pixels on line
- 1-d array containing elevation angles of pixels on line
- 1-d array containing corresponding x-pixel coordinates
- 1-d array containing corresponding y-pixel coordinates

Return type tuple

get_topo_distance_pix (*pos_x_abs*, *pos_y_abs*, *topo_res_m=5.0*, *min_slope_angle=5.0*)
Retrieve distance to topography for a certain image pixel.

The computation of the distance is being done by retriving a elevation profile in the azimuthal viewing direction of tge pixel (i.e. pixel column) and then using this profile and the corresponding camera elevation (pixel row) to find the first intersection of the viewing direction (line) with the topography

Parameters

- **pos_x_abs** (*int*) – x-pixel position of point in image in absolute coordinate (i.e. pyramid level 0 and not cropped)
- **pos_y_abs** (*int*) – y-pixel position of point in image in absolute coordinate (i.e. pyramid level 0 and not cropped)
- **topo_res_m** (*float*) – desired resolution of topographic data (is interpolated)
- **min_slope_angle** (*float*) – mininum required slope (steepness) of topography at pixel position (raises ValueError if topograpy is too flat)

Returns

3-element tuple, containing

- **estimated distance to topography in m based on intersection of** pixel viewing direction with topographic data
- corresponding uncertainty in m
- `GeoPoint` corresponding to intersection position

Return type tuple

get_topo_distances_line(*line*, *skip_pix=30*, *topo_res_m=5.0*, *min_slope_angle=5.0*)
Retrieve distances to topography for a line on an image.

Calculates distances to topography based on pixels on the line. This is being done by retriving a elevation profile in the azimuthal viewing direction of each pixel (i.e. pixel column) and then using this profile and the corresponding camera elevation (pixel row) to find the first intersection of the viewing direction (line) with the topography

Parameters

- **line** (*list*) – list with line coordinates: `[x0, y0, x1, y1]` (can also be `LineOnImage` object)
- **skip_pix** (*int*) – step width for retrieval along line
- **topo_res_m** (*float*) – desired resolution of topographic data (is interpolated)
- **min_slope_angle** (*float*) – mininum angle of slope, pixels pointing into flatter topographic areas are ignored

get_angular_displacement_pix_to_cfov(*pos_x*, *pos_y*)
Get the angular difference between pixel and detector center.

Parameters

- **pos_x** (*int*) – x position on detector
- **pos_y** (*int*) – y position on detector

get_azim_elev(*pos_x*, *pos_y*)
Get values of azimuth and elevation in pixel (x|y).

Parameters

- **pos_x** (*int*) – x position on detector
- **pos_y** (*int*) – y position on detector

get_elevation_profile(*col_num=None*, *azim=None*, *dist_hor=None*, *topo_res_m=5.0*)
Retrieve elev profile from camera into a certain azim direction.

Parameters

- **col_num** (*int*) – pixel column number of profile, if None or not in image detector range then try to use second input parameter azim
- **azim** (*float*) – is only used if input param col_num == None, then profile is retrieved from camera in direction of specified azimuth angle
- **dist_hor** (*float*) – horizontal distance (from camera, in km) up to witch the profile is determined. If None, then use 1.05 times the camera source distance
- **topo_res_m** (*float*) – desired horizontal grid resolution in m ()

get_distance_to_topo(*col_num=None*, *row_num=None*, *azim=None*, *elev=None*, *min_dist=0.2*, *max_dist=None*)

Determine distance to topography based on pixel coordinates.

Parameters

- **col_num** (*int*) – pixel column number for elevation profile, from which the intersection with viewing direction is retrieved. If None or not in image detector range then try to use third input parameter (azim)
- **row_num** (*int*) – pixel row number for which the intersection with elevation profile is is retrieved. If None or not in image detector range then try to use 4th input parameter elev, row_num is only considerd if col_num is valid
- **azim** (*float*) – camera azimuth angle of intersection: is only used if input param col_num == None
- **elev** (*float*) – camera elevation angle for distance estimate: is only used if input param row_num == None
- **min_dist** (*float*) – minimum distance (in km) from camera for retrieval of first intersection. Intersections of viewing direction with topography closer than this distance are disregarded (default: 0.2)
- **max_dist** (*float*) – maximum distance (in km) from camera for which intersections with topography are searched

find_viewing_direction(*pix_x*, *pix_y*, *pix_pos_err=10*, *obj_id=''*, *geo_point=None*, *lon_pt=None*, *lat_pt=None*, *alt_pt=None*, *update=True*, *draw_result=False*)

Retrieve camera viewing direction from point in image.

Uses the geo coordinates of a characteristic point in the image (e.g. the summit of a mountain) and the current position of the camera (Lon / Lat) to determine the viewing direction of the camera (azimuth, elevation).

Parameters

- **pix_x** (*int*) – x position of object on camera detector (measured from left)
- **pix_y** (*int*) – y position of object on camera detector (measured from top)
- **pix_pos_err** (*int*) – radial uncertainty in pixel location (used to estimate and update `self._cam["elev_err"], self._cam["azim_err"]`)
- **update** (*bool*) – if True current data will be updated and `self.geo_setup` will be updated accordingly
- **obj_id** (*str*) – string ID of object, if this object is available as `GeoPoint` in `self.geo_setup` then the corresponding coordinates will be used, if not, please provide the

position of the characteristic point either using **:param:'geo_point'** or by providing its coordinates using params lat_pt, lon_pt, alt_pt

- **geo_point** (*GeoPoint*) – geo point object of characteristic point
- **lon_pt** (*float*) – longitude of characteristic point
- **lat_pt** (*float*) – latitude of characteristic point
- **alt_pt** (*float*) – altitude of characteristic point (unit m)
- **update** – if True, camera azim and elev are updated within this object
- **draw_result** (*bool*) – if True, a 2D map is drawn showing results

Returns

- float, retrieved camera elevation
- float, retrieved camera azimuth
- MeasGeometry, initial state of this object, a deepcopy of

this class, before changes where applied (if they were applied,see also *update*)

pix_dist_err (*col_num, pyrlevel=0*)

Get uncertainty measure for pixel distance of a pixel column.

Parameters

- **colnum** (*int*) – column number for which uncertainty in pix-to-pix distance is computed
- **pyrlevel** (*int*) – convert to pyramid level

Returns pix-to-pix distance in m corresponding to input column number and pyramid level

Return type float

compute_all_integration_step_lengths (*pyrlevel=0, roi_abs=None*)

Determine images containing pixel and plume distances.

Computes and returns three images where each pixel value corresponds to:

1. the horizontal physical integration step length in units of m
2. **the vertical physical integration step length in units of m (is** the same as 1. for standard detectors where the vertical and horizontal pixel pitch is the same)
3. image where each pixel corresponds to the computed plume distance

Parameters

- **pyrlevel** (*int*) – returns images at a given gauss pyramid level
- **roi_abs** (*list*) – ROI `[x0, y0, x1, y1]` in absolute detector coordinates. If valid, then the images are cropped accordingly

Returns

3-element tuple, containing

- **Img: image where each pixel corresponds to pixel** column distances in m
- **Img: image where each pixel corresponds to pixel** row distances in m (same as col_dist_img if pixel width andheight are equal)
- **Img: image where each pixel corresponds to plume** distance in m

Return type tuple

get_plume_direction()
Return the plume direction plus error based on wind direction.

plot_view_dir_pixel(*col_num*, *row_num*)
2D plot of viewing direction within elevation profile.

Determines and plots elevation profile for azimuth angle of input pixel coordinate (column number). The viewing direction line is plotted based on the specified elevation angle (corresponding to detector row number)

Parameters

- **col_num** (*int*) – column number of pixel on detector
- **row_num** (*int*) – row number of pixel on detector (measured from top)

Returns elevation profile

draw_map_2d(*draw_cam=True*, *draw_source=True*, *draw_plume=True*, *draw_fov=True*, *draw_topo=True*, *draw_coastline=True*, *draw_mapscale=True*, *draw_legend=True*, **args*, ***kwargs*)
Draw the current setup in a map.

Parameters

- **draw_cam** (*bool*) – insert camera position into map
- **draw_source** (*bool*) – insert source position into map
- **draw_plume** (*bool*) – insert plume vector into map
- **draw_fov** (*bool*) – insert camera FOV (az range) into map
- **draw_topo** (*bool*) – plot topography
- **draw_coastline** (*bool*) – draw coastlines
- **draw_mapscale** (*bool*) – insert a map scale
- **draw_legend** (*bool*) – insert a legend
- ***args** – additional non-keyword arguments for setting up the base map (see here)
- ****kwargs** – additional keyword arguments for setting up the base map (see here)

draw_azrange_fov_2d(*m*, *fc='lime'*, *ec='none'*, *alpha=0.15*, *poly_id='fov'*)
Insert the camera FOV in a 2D map.

Parameters

- **m** (*geonum.mapping.Map*) – the map object
- **fc** – face color of polygon
- **alpha** (*float*) – alpha value of polygon

Param ec edge color of polygon

draw_map_3d(*draw_cam=True*, *draw_source=True*, *draw_plume=True*, *draw_fov=True*, *cmap_topo='Oranges'*, *contour_color='#708090'*, *contour_antialiased=True*, *contour_lw=0.2*, *ax=None*, ***kwargs*)
Draw the current setup in a 3D map.

Parameters

- **draw_cam** (*bool*) – insert camera position into map
- **draw_source** (*bool*) – insert source position into map

- **draw_plume** (*bool*) – insert plume vector into map
- **draw_fov** (*bool*) – insert camera FOV (az range) into map
- **cmap_topo** (*str*) – string ID of colormap for topography surface plot, defaults to "Oranges"
- **contour_color** (*str*) – string specifying color of contour lines colors of topo contour lines (default: "#708090")
- **contour_antialiased** (*bool*) – apply antialiasing to surface plot of topography, defaults to False
- **contour_lw** – width of drawn contour lines, defaults to 0.5, use 0 if you do not want contour lines inserted
- **ax** (*Axes3D*) – 3D axes object (default: None -> creates new one)
- ***args** – non-keyword arguments for setting up the base map (see here)
- ****kwargs** (*keyword arguments for setting up the basemap*) – (see here)

Returns plotted basemap

Return type Basemap

draw_azrange_fov_3d(*m*, *fc='lime'*, *ec='none'*, *alpha=0.8*)
Insert the camera FOV in a 2D map.

Parameters

- **m** (*geonum.mapping.Map*) – the map object
- **fc** – face color of polygon ("lime")
- **alpha** (*float*) – alpha value of polygon (0.8)

Param ec edge color of polygon ("none")

plume_dir
Return current plume direction angle.

plume_dir_err
Return uncertainty in current plume direction angle.

cam
Camera location (geonum.GeoPoint).

source
Return camera Geopoint.

intersect_pos
Return camera Geopoint.

plume_vec
Return the plume center vector.

source2cam
Return vector pointing camera to source.

cam_view_vec
Return vector corresponding to CFOV azimuth of camera view dir.

haversine(*lon0*, *lat0*, *lon1*, *lat1*, *radius=6371.0*)
Haversine formula.

Approximate horizontal distance between 2 points assuming a spherical earth

Parameters

- **lon0** (*float*) – longitude of first point in decimal degrees
- **lat0** (*float*) – latitude of first point in decimal degrees
- **lon1** (*float*) – longitude of second point in decimal degrees
- **lat1** (*float*) – latitude of second point in decimal degrees
- **radius** (*float*) – average earth radius in km (6371.0)

geo_len_scale()
Return the distance between cam and source in km.

Uses haversine formula (*haversine()*) to determine the distance between source and cam to estimate the geoprahic dimension of this setup

Returns float, distance between source and camera

del_az(*pixcol1=0*, *pixcol2=1*)
Determine the difference in azimuth angle between 2 pixel columns.

Parameters

- **pixcol1** (*int*) – first pixel column
- **pixcol2** (*int*) – second pixel column

Returns azimuth difference in degrees

Return type float

del_elev(*pixrow1=0*, *pixrow2=1*)
Determine the difference in azimuth angle between 2 pixel columns.

Parameters

- **pixrow1** (*int*) – first pixel row
- **pixrow2** (*int*) – second pixel row

Returns elevation difference in degrees

Return type float

plume_dist(*az=None*, *elev=None*)
Return plume distance for input azim and elev angles.

Computes the distance to the plume for the whole image plane, assuming that the horizontal plume propagation direction is given by the meteorological wind direction. The vertical component of the plume distance for each pixel row and column is computed based on the corresponding elevation angle and horizontal distance to the plume.

Parameters

- **az** (float or list, optional) – azimuth value(s) (single val or array of values). If *None*, then the azimuth angle of the camera CFOV is used.
- **elev** (float or list, optional) – elevation angle(s) (single val or array of values). If *None*, then the elevation angle of the camera CFOV is used.

Returns plume distance(s) in m for input azimuth(s) and elevations

Return type `float` or `list`

plume_dist_err(*az=None*)

Compute uncertainty in plume distances.

The computation is based on uncertainties in the camera azimuth and the uncertainty in the horizontal wind direction (i.e. plume propagation direction).

Parameters **az** (`float`, optional) – camera azimuth angle for which the uncertainty is computed (if None then the CFOV azimuth is used).

Returns absolute uncertainty in plume distance in units of m

Return type float

all_elevs_camfov()

Return array containing elevation angles for each image row.

all_azimuths_camfov()

Return array containing azimuth angles for each image row.

col_to_az(*colnum*)

Convert pixel column number (in absolute coords) into azimuth angle.

Note:

- See also *az_to_col()* for the inverse operation
- **Not super efficient, just convenience function which should not** be used if performance is required

---

Parameters **colnum** (*int*) – pixel column number (left column corresponds to 0)

Returns corresponding azimuth angle

Return type float

az_to_col(*azim*)

Convert azimuth into pixel number.

Note: The pixel number is calculated relative to the leftmost column of the image

---

Parameters **azim** (*float*) – azimuth angle which is supposed to be converted into column number

Returns column number

Return type int

Raises **IndexError** – if input azimuth is not within camera FOV

5.4 Image base module

Classes representing image data and corresponding processing features.

The image base class *Img* is a powerful object for image data, containing I/O routines for many data formats, processing classes and keeping track on changes applied to the images. The actual image data is stored as numpy array in the `img` of an instance of the *Img* object.

The *ProfileTimeSeriesImg* class is used to store and process time series of pixel profiles (e.g. along a `LineOnImage`). These are, for instance used when performing a plume velocity cross-correlation analysis (where the optimal lag between a time-series of two plume intersection lines is searched, for details see *pyplis.plumespeed.VeloCrossCorrEngine*).

class pyplis.image.**Img**(*input=None*, *import_method=None*, *dtype=None*, ***meta_info*)

Image base class.

Implementation of image object for `pyplis` library. The image data is represented as `numpy.ndarray` objects and the is stored in the attribute `self.img`.

Supported file formats include those supported by the Python Imaging Library (see here) and the FITS format. Img objects can also be created from numpy arrays directly.

The object includes several loading routines and basic image editing. Image meta information can be provided on creation of this instance by providing valid meta keys and the corresponding values, i.e.:

```
png_image_file = "C:/Test/my_img_file.png"
acq_time = datetime(2016, 10, 10, 13, 15, 12) #10/10/2016, 13:15:12
exposure_time = 0.75 #s
img = Img(png_image_file, start_acq = acq_time, texp = exposure_time)
```

Meta information is stored in the dictionary `self.meta` and can be printed using *print_meta()*. The two most important image meta parameters are the acquisition time (`img.meta["start_acq"]`) and the exposure time (`img.meta["texp"]`). These two parameters have class own access methods (*start_acq()* and *texp()*).

The class provides several image editing routines, of which the most important ones (within this library) are (please see documentation of the individual functions for more information):

1. *subtract_dark_image()* (subtract a dark image)
2. ***correct_dark_offset()*** **(Correct for dark and offset. Models** a dark image based on one dark and one offset image using the exposure time of this image, then uses 1. for subtraction)
3. *crop()* (crop image within region of interest)
4. *apply_median_filter()* (median filtering of image)
5. ***add_gaussian_blurring()*** **(Add blurring to image taking into** account current blurring amount)
6. *apply_gaussian_blurring()* (applies gaussian filter to image)
7. *pyr_down()* (reduce image size using gaussian pyramide)
8. *pyr_up()* (increase image size using gaussian pyramide)

All image editing steps performed using these functions are logged in the dictionary `self.edit_log`, it is therefore recommended to use the class own methods for these image editing routines (and not apply them manually to the image data, e.g. by using `cv2.pyrDown(img.img)` for resizing or `img.img = img.img[y0:y1, x0:x1]` for cropping a ROI `[x0, x1, y0, y1]`) in order to keep track of the changes applied.

The default data accuracy is 32 bit floating point and can be changed on initiation (see *__init__()*).

Parameters

- **input** – image data input (e.g. file path to an image type which can be read or numpy array)
- **import_method** – custom image load method, must return tuple containing image data (2D ndarray) and dictionary containing meta information (can be empty if read routine does not import any meta information)
- **dtype** – datatype for image data (float32)
- ****meta_info** – keyword args specifying meta data (only valid metadata is stored, for valid keys see *meta*)

dtype
data type of image numpy array

vign_mask
vignetting mask used to correct for vignetting (is set in *correct_vignetting()*)

Type ndarray, optional

import_method
custom method used to import image data

Type callable, optional

edid_log
dictionary containing information about editing status

Type dict

meta
dictionary containing meta information.

Type dict

__init__(*input=None*, *import_method=None*, *dtype=None*, ***meta_info*)
x.__init__(...) initializes x; see help(type(x)) for signature

img
Get / set image data.

start_acq
Get image acquisition time.

Returns acquisition time if available (i.e. it deviates from the default 1/1/1900), else, raises ImgMetaError

stop_acq
Return stop time of acquisition (if available).

texp
Get image acquisition time.

Returns acquisition time if available (i.e. it deviates from the default 1/1/1900), else, raises ImgMetaError

gain
Return read gain value from meta info.

shape
Return shape of image data.

xy_aspect
Aspect ratio (delx / dely).

pyr_up_factor
Return coordinates conversion factor.

This factor is used to convert coordinates at current pyramid level into original size coordinates.

is_darkcorr
Boolean specifying whether image is dark corrected.

is_tau
Return boolean whether image is a tau image or not.

is_aa
Return boolean whether current image is AA image.

is_calibrated
Flag for image calibration status.

is_vignetting_corrected
Boolean stating whether image is vignetting corrected or not.

is_gray
Check if image is gray image.

is_binary
Attribute specifying whether image is binary image.

is_inverted
Flag specifying whether image was inverted or not.

is_vigncorr
Bool specifying whether or not image is vignetting corrected.

is_dilcorr
Return whether this image is corrected for signal dilution.

is_cropped
Boolean specifying whether image is cropped.

is_resized
Boolean specifying whether image pyramid level unequals 0.

is_shifted
Boolean specifying whether image was shifted.

This may be e.g. the case for stereo imaging

modified
Check if this image was already modified.

pyrlevel
Return current gauss pyramid level (stored in `self.edit_log`).

roi
Return current roi in consideration of current pyrlevel.

roi_abs
Get / set current ROI in absolute image coordinates.

Note: use *roi()* to get ROI for current pyrlevel

set_data(*input*)
Try load input.

reload()
Try reload from file.

load_input(*input*)
Try to load input as numpy array and additional meta data.

make_histogram()
Make histogram of current image.

get_brightness_range()
Analyses the Histogram to retrieve a suited brightness range.

Note: Currently not in use (was originally used for App)

avg_in_roi(*mask=None*, *roi_rect=None*, *pos_x=None*, *pos_y=None*, *radius=1*)
Get mean value in an ROI.

The ROI can be specified either by providing a mask, an rectangular ROI, or x and y position and a specific radius. The input is dealt with in the specified order, i.e. if **:param:'mask'** is valid, none of the other input parameters is tested.

Parameters

- **mask** (`ndarray` or *Img*) – convolution mask (e.g. DOAS FOV mask).
- **roi_rect** (*list*) – rectangular ROI `[x0, y0, x1, y1]` specifying upper left and lower right corners of region
- **pos_x** (*int*) – detector x-position
- **pos_y** (*int*) – detector y-position
- **radius** (*int*) – radius of ROI

Raises **TypeError** – if none of the provided input works

Returns mean value within specified ROI

Return type float

crop(*roi_abs=[0, 0, 9999, 9999]*, *new_img=False*)
Cut subimage specified by rectangular ROI.

Parameters

- **roi_abs** (*list*) – region of interest (i.e. `[x0, y0, x1, y1]`) in ABSOLUTE image coordinates. The ROI is automatically converted with respect to current pyrlevel
- **new_img** (*bool*) – creates and returns a new image object and leaves this one uncropped

Returns

- Img, cropped image

correct_dark_offset(*dark*, *offset*)
Perform dark frame subtraction, 3 different modi possible.

Parameters

- **dark** (`Img`) – dark image object (dark with long(est) exposure time)
- **offset** (`Img`) – offset image (dark with short(est) exposure time)

Return Img modelled dark image

Uses *model_dark_image()* (in Processing) to model a dark image based on the exposure time of this image object. This is then subtracted from the current image.

Note: This algorithm works only, if no other image processing operations were applied to the input image beforehand, i.e. if *modified()* returns False.

---

subtract_dark_image(*dark*)
Subtracts a dark (+offset) image and updates self.edit_log.

Parameters **dark** (Img) – dark image data

Simple image subtraction without any modifications of input image

correct_vignetting(*mask*, *new_state=True*)
Apply vignetting correction.

Performs either of the following operations:

```
self.img * mask      (if input param ``new_state=False``)
self.img / mask      (if input param ``new_state=True``)
```

Parameters

- **mask** (*ndarray*) – vignetting correction mask
- **reverse** (*bool*) – if False, the inverse correction is applied (img needs to be corrected)

set_roi_whole_image()
Set current ROI to whole image area based on shape of image data.

apply_median_filter(*size_final=3*)
Apply a median filter.

Parameters **shape (3,3)** (*tuple*) – size of the filter

add_gaussian_blurring(*sigma_final=1*)
Add blurring to image.

Parameters **sigma_final** (*int*) – the final width of gauss blurring kernel

apply_gaussian_blurring(*sigma*, ***kwargs*)
Add gaussian blurring.

Uses scipy.ndimage.filters.gaussian_filter

Parameters **sigma** (*int*) – amount of blurring

get_masked_img(*mask*, *fill_value=None*)
Return a np.ma.masked_array of the img array.

Parameters

- **mask** (*numpy.ndarray*) – entries which should be masked (True=invalid entry) has to be same shape as current state of self.img
- **fill_value** (*float*) – (optional, default None) If defined, invalid entries are replaced by fill_value

Returns masked array

Return type numpy.ma.masked_array

get_thresh_mask(*threshold*)
Apply threshold and get binary mask.

to_binary(*threshold=None*, *new_img=False*)
Convert image to binary image using threshold.

Note: The changes are applied to this image object

Parameters **threshold** (*float*) – threshold, if None, use mean value of image data

Returns binary image

Return type *Img*

invert()
Invert image.

Note: Does not yet work for tau images

Returns inverted image object

Return type *Img*

convolve_with_mask(*mask*)
Convolves this image data with input mask and return value.

Note: This is not an image convolution with a kernel that is applied to each image pixel (e.g. blurring, etc.). The input mask is supposed to be of the same shape as this image

Parameters **mask** (*ndarray*) – 2D array of same dimension (height, width) as this image

Returns corresponding value after normalisation and convolution

Return type float

dilate(*kernel=array([[1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1]], dtype=uint8)*)
Apply morphological transformation Dilation to image.

Uses `cv2.dilate()` for dilation. The method requires specification of a smoothing kernel, if unspecified, a 9x9 neighbourhood is used

Note: This operation can only be performed to binary images, use `to_binary()` if applicable.

Parameters **kernel** (*array*) – kernel used for `cv2.dilate()`, default is 9x9 kernel

Returns dilated binary image

Return type *Img*

erode (*kernel=array([[1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1]], dtype=uint8)*)

Apply morphological transformation Erosion to image.

Uses `cv2.erode()` to apply erosion. The method requires specification of a kernel, if unspecified, a 9x9 neighbourhood is used

Note: This operation can only be performed to binary images, use `to_binary()` if applicable.

Parameters **kernel** (*array*) – kernel used for `cv2.dilate()`, default is 9x9 kernel

Returns dilated binary image

Return type *Img*

fit_2d_poly (*mask=None*, *polyorder=3*, *pyrlevel=4*, ***kwargs*)

Fit 2D surface poly to data.

Parameters

- **mask** (*array*) – mask specifying pixels considered for the fit (if None, then all pixels of the image data are considered
- **polyorder** (*int*) – order of polynomial for fit (default=3)
- **pyrlevel** (*int*) – level of Gauss pyramid at which the fit is performed (relative to Gauss pyramid level of input data)
- ****kwargs** – additional optional keyword args passed to `PolySurfaceFit`

Returns new image object corresponding to fit results

Return type *Img*

to_tau (*bg*, *new_img=True*)

Convert into tau image.

Converts this image into a tau image using a provided input background image (which is used without any modifications).

Note: By default, creates and returns new instance of `Img` object (i.e. this object remains unchanged if not other specified using

:param:'new_img')

Parameters

- **bg** (`Img`) – background image used to determin tau image (REMAINS UNCHANGED, NO MODELLING PERFORMED HERE)
- **new_img** (*bool*) – boolean specifying whether this object remains unchanged

Returns new Img object containing tau image data (this object remains unchanged)

Return type *Img*

to_pyrlevel (*final_state=0*)

Down / upscale image to a given pyramid level.

pyr_down(*steps=0*)
Reduce the image size using gaussian pyramide.

Parameters **steps** (*int*) – steps down in the pyramide

Algorithm used: `cv2.pyrDown()`

pyr_up(*steps*)
Increasing the image size using gaussian pyramide.

Parameters **steps** (*int*) – steps down in the pyramide

Algorithm used: `cv2.pyrUp()`

bytescale(*cmin=None*, *cmax=None*, *high=255*, *low=0*)
Convert image to 8 bit integer values.

Parameters

- **cmin** (*float*) – minimum intensity for mapping, if None, the current `self.min()` is used.
- **cmax** (*float*) – maximum intensity for mapping, if None, the current `self.max()` is used.
- **high** (*int*) – mapping value of cmax
- **low** (*int*) – mapping value of cmin

is_8bit()
Flag specifying whether image is 8 bit.

print_meta()
Print current image meta information.

make_info_header_str()
Make header string for image (using image meta information).

duplicate()
Duplicate this image.

normalise(*blur=1*)
Normalise this image.

mean()
Return mean value of current image data.

sum()
Return the sum of all pixel values.

std()
Return standard deviation of current image data.

min()
Return minimum value of current image data.

max()
Return maximum value of current image data.

set_val_below_thresh(*val*, *threshold*)
Set value in all pixels with intensities below threshold.

Note: Modifies this Img object

Parameters

- **val** (*float*) – new value for all pixels below the input threshold
- **threshold** (*float*) – considered intensity threshold

set_val_above_thresh(*val*, *threshold*)

Set value in all pixels with intensities above threshold.

Note: Modifies this Img object

Parameters

- **val** (*float*) – new value for all pixels above the input threshold
- **threshold** (*float*) – considered intensity threshold

blend_other(*other*, *fac=0.5*)

Blends another image to this and returns new Img object.

Uses cv2 `addWeighted()` method"

Parameters **fac** (*float*) – percentage blend factor (between 0 and 1)

meta(*meta_key*)

Return current meta data for input key.

load_file(*file_path*)

Try to import file specified by input path.

load_fits(*file_path*)

Import a FITS file.

This import method assumes, that data and corresponding meta-info is stored in the first HDU of the FITS file (index = 0).

Fits info

save_as_fits(*save_dir=None*, *save_name=None*)

Save this image as FITS file.

Parameters

- **save_dir** (*str*) – optional, if None (default), then the current working directory is used
- **save_name** (*str*) – opional, if None (default), try to use file name of this object (if set) or use default name

Returns name of saved file

Return type str

get_cmap(*vmin=None*, *vmax=None*, ***kwargs*)

Determine and return default cmap for current image.

show(*zlabel=None*, *tit=None*, ***kwargs*)

Plot image.

show_img(*zlabel=None*, *tit=None*, *cbar=True*, *ax=None*, *zlabel_size=18*, ***kwargs*)

Show image using matplotlib method imshow.

show_img_with_histo(***kwargs*)

Show image using plt.imshow.

shift (*dx_abs=0.0, dy_abs=0.0*)
Apply constant image shift to this object.

Parameters

- **dx_abs** (*float*) – shift in x-direction
- **dy_abs** (*float*) – shift in y-direction

show_histogram (*ax=None*)
Plot histogram of current image.

Todo: Needs more edit (i.e. better representation of labels)

---

info ()
Print image info from string representation.

pyplis.image.**model_dark_image** (*texp, dark, offset*)
Model a dark image for input image based on dark and offset images.

Determine a modified dark image (D_mod) from the current dark and offset images. The dark image is determined based on the image exposure time of the image object to be corrected (t_exp,I). D_mod represents dark and offset signal for this image object and is then subtracted from the image data.

Formula for modified dark image:

$$D_{mod} = O+$$
$$frac(D - O) * t_{exp,I}(t_{exp,D} - t_{exp,O})$$

Parameters

- **img** (Img) – the image for which dark and offset is modelled
- **dark** (Img) – dark image object (dark with long(est) exposure time)
- **offset** (Img) – offset image (dark with short(est) exposure time)

Returns

- *Img*, modelled dark image

class pyplis.image.**ProfileTimeSeriesImg** (*img_data=None, time_stamps=array([], dtype=float64), img_id=", dtype=<type 'numpy.float32'>, profile_info_dict=None, **meta_info*)
Image representing time series of line profiles.

The y axis of the profile image corresponds to the actual profiles (retrieved from the individual images) and the x axis corresponds to the image time axis (i.e. the individual frames). Time stamps (mapping of x indices) can also be stored in this object.

Example usage is, for instance to represent ICA time series retrieved along a profile (e.g. using `LineOnImage`) for plume speed cross correlation

__init__ (*img_data=None, time_stamps=array([], dtype=float64), img_id=", dtype=<type 'numpy.float32'>, profile_info_dict=None, **meta_info*)
x.__init__(...) initializes x; see help(type(x)) for signature

img
Get / set image data.

start
Return first datetime from `self.time_stamps`.

stop
Return first datetime from `self.time_stamps`.

save_as_fits (*save_dir=None*, *save_name=None*, *overwrite_existing=True*)
Save stack as FITS file.

Parameters

- **save_dir** (*str*) – directory where image is stored (can also be full file path, then parameter `save_name` is not considered)
- **save_name** (*str*) – name of file
- **overwrite_existing** (*bool*) – if True, an existing file with the same name will be overwritten

load_fits (*file_path*, *profile_type='LineOnImage'*)
Load stack object (fits).

Parameters **file_path** (*str*) – file path of fits image

5.5 Image list objects

Image list objects.

Image list objects (e.g. *BaseImgList*, *ImgList*, *DarkImgList*, *CellImgList*) contain a list of image file paths and are central for the data analysis. Images are loaded as `Img` objects and are loaded and processed iteratively. Typically one list contains all images of a certain type (e.g. onband, offband, see `Dataset` object). *ImgList* objects (inherited from *BaseImgList*) contain powerful pre-processing modes (e.g. load images as dark corrected and calibrated images, compute optical flow between current and next image).

class pyplis.imagelists.**BaseImgList** (*files=None*, *list_id=None*, *list_type=None*, *camera=None*, *geometry=None*, *init=True*, ***img_prep_settings*)
Basic image list object.

Basic class for image list objects providing indexing and image loading functionality

In this class, only the current image is loaded at a time while *ImgList* loads current and next image whenever the index is changed (e.g. required for `optflow_mode`)

This object and all objects inheriting from this are fundamentally based on a list of image file paths, which are dynamically loaded and processed during usage.

Parameters

- **files** (*list, optional*) – list with image file paths
- **list_id** (*str, optional*) – a string used to identify this list (e.g. "second_onband")
- **list_type** (*str, optional*) – type of images in list (please use "on" or "off")
- **camera** (*Camera, optional*) – camera specifications
- **init** (*bool*) – if True, list will be initiated and files loaded (given that image files are provided on input)
- ****img_prep_settings** – additional keyword args specifying image preparation settings applied on image load

__init__ (*files=None*, *list_id=None*, *list_type=None*, *camera=None*, *geometry=None*, *init=True*, ***img_prep_settings*)
x.__init__(. . .) initializes x; see help(type(x)) for signature

start
Acquisistion time of first image.

stop
Start acqusition time of last image.

this
Return current image.

edit_active
Define whether images are edited on image load or not.

If False, images will be loaded as raw, i.e. without any editing or further calculations (e.g. determination of optical flow, or updates of linked image lists). Images will be reloaded.

skip_files
Integer specifying the image iter step in the file list.

Defaults to 1: every file is used, 2 means, that every second file is used.

meas_geometry
Return measurement geometry.

update_cam_geodata
Update measurement geometry whenever list index is changed.

plume_dists
Distance to plume.

Can be an image were each pixel value corresponds to the plume distance at each pixel position (e.g. computed using the MeasGeometry) or can also be a single value, which may be appropriate under certain measurement setups (e.g. distant plume perpendicular to CFOV of camera)

Note: This method checks if a value is accessible in `_plume_dists` and if not tries to compute plume distances by calling `compute_all_integration_step_lengths()` of the `MeasGeometry` object assigned to this ImgList. If this fails, then an AttributeError is raised

Returns Plume distances in m. If plume distances are accessible per image pixel. Note that the corresponding data is converted to pyramid level 0 (required for dilution correction).

Return type float or *Img* or ndarray

vign_mask
Return current vignetting correction mask.

sky_mask
Return sky access mask.

0 for sky, 1 for non-sky (=invalid) (in masked arrays, entries marked with 1 are invalid)

integration_step_length
Return integration step length for emission-rate analyses.

The intgration step length corresponds to the physical distance in m between two pixels within the plume and is central for computing emission-rate. It may be an image were each pixel value corresponds to the integreation step length at each pixel position (e.g. computed using the MeasGeometry) or it can also be a single value, which may be appropriate under certain measurement setups (e.g. distant plume perpendicular to CFOV of camera).

Note: This method checks if a value is accessible in `_integration_step_lengths` and if not tries to compute them by calling `compute_all_integration_step_lengths()` of the `MeasGeometry` object assigned to this ImgList. If this fails, an AttributeError is raised

Returns Integration step lengths in m. If plume distances are accessible per image pixel, then the corresponding data IS converted to the current pyramid level

Return type float or *Img* or ndarray

auto_reload
Activate / deactivate automatic reload of images.

crop
Activate / deactivate crop mode.

pyrlevel
Return current Gauss pyramid level.

Note: images are reloaded on change

gaussian_blurring
Return current blurring level.

Note: images are reloaded on change

roi
Return current ROI (in relative coordinates).

The ROI is returned with respect to the current `pyrlevel`

roi_abs
Return current roi in absolute detector coords (cf. `roi`).

cfn
Return current index (file number in `files`).

nof
Return number of files in this list.

last_index
Return index of last image.

data_available
Return wrapper for `has_files()`.

has_images
Return wrapper for `has_files()`.

start_acq
Array containing all image acq. time stamps of this list.

Note: The time stamps are extracted from the file names

timestamp_to_index(*val=datetime.datetime(1900, 1, 1, 0, 0)*)
Convert a datetime to the list index.

Returns the list index that is closest in time to the input time stamp.

Parameters **val** (*datetime*) – time stamp

Raises **AttributeError** – if time stamps of images in list cannot be accessed from their file names

Returns corresponding list index

Return type int

index_to_timestamp(*val=0*)
Get timestamp of input list index.

Parameters **val** (*index*) – time stamp

Raises **AttributeError** – if time stamps of images in list cannot be accessed from their file names

Returns corresponding list index

Return type int

add_files(*files*, *load=True*)
Add images to this list.

Parameters **file_list** (*list*) – list with file paths

Returns success / failed

Return type bool

init_filelist(*at_index=0*)
Initialize the filelist.

Sets current list index and resets loaded images

Parameters **at_index** (*int*) – desired image index, defaults to 0

iter_indices(*to_index*)
Change the current image indices for previous, this and next img.

Note: This method only updates the actual list indices and does not perform a reload.

load()
Load current image.

Try to load the current file `self.files[self.cfn]` and if remove the file from the list if the import fails

Returns if True, image was loaded, if False not

Return type bool

goto_next()
Goto next index in list.

goto_prev()
Load previous image in list.

goto_img(*to_index*, *reload_here=False*)
Change the index of the list, load and prepare images at new index.

Parameters

- **to_index** (*float*) – new list index
- **reload_here** (*bool*) – applies only if **:param:'to_index'** is the current list index. If True, then the current images are reloaded, if False, nothing is done.

pop (*idx=None*)
Remove one file from this list.

has_files ()
Return boolean whether or not images are available in list.

plume_dist_access ()
Check if measurement geometry is available.

update_img_prep (***settings*)
Update image preparation settings and reload.

Parameters ****settings** – key word args specifying settings to be updated (see keys of `img_prep` dictionary)

clear ()
Empty this list (i.e. `files`).

separate_by_substr_filename (*sub_str, sub_str_pos, delim='_'*)
Separate this list by filename specifications.

The function checks all current filenames, and keeps those, which have a certain sub string at a certain position in the file name after splitting using a provided delimiter. All other files are added to a new image list which is returned.

Parameters

- **sub_str** (*str*) – string identification used to identify the image type which is supposed to be kept within this list object
- **sub_str_pos** (*int*) – position of sub string after filename was split (using input param delim)
- **delim** (*str*) – filename delimiter, defaults to "_"

Returns

2-element tuple containing

- *ImgList*, list contains images matching the requirement
- *ImgList*, list containing all other images

Return type tuple

set_camera (*camera=None, cam_id=None*)
Set the current camera.

Two options:

1. set `Camera` directly
2. provide one of the default camera IDs (e.g. "ecII", "hdcam")

Parameters

- **camera** (`Camera`) – the camera used

- **cam_id** (*str*) – one of the default cameras (use `pyplis.inout.get_all_valid_cam_ids()` to get the default camera IDs)

reset_img_prep()

Init image pre-edit settings.

get_img_meta_from_filename(*file_path*)

Load and prepare img meta input dict for Img object.

Parameters **file_path** (*str*) – file path of image

Returns dictionary containing retrieved values for `start_acq` and `texp`

Return type dict

get_img_meta_all_filenames()

Try to load acquisition and exposure times from filenames.

Note: Only works if relevant information is specified in `self.camera` and can be accessed from the file names

Returns

2-element tuple containing

- list, list containing all retrieved acq. time stamps
- list, containing all retrieved exposure times

Return type tuple

assign_indices_linked_list(*lst*)

Create a look up table for fast indexing between image lists.

Parameters **lst** (`BaseImgList`) – image list supposed to be linked

Returns array contining linked indices

Return type array

same_preedit_settings(*settings_dict*)

Compare input settings dictionary with self.img_prep.

Parameters ****settings_dict** – keyword args specifying settings to be compared

Returns False if not the same, True else

Return type bool

make_stack(*stack_id=None, pyrlevel=None, roi_abs=None, start_idx=0, stop_idx=None, ref_check_roi_abs=None, ref_check_min_val=None, ref_check_max_val=None, dtype=<type 'numpy.float32'>*)

Stack all images in this list.

The stacking is performed using the current image preparation settings (blurring, dark correction etc). Only stack ROI and pyrlevel can be set explicitely.

Note: In case of `MemoryError` try stacking less images (specifying start / stop index) or reduce the size setting a different Gauss pyramid level.

Parameters

- **stack_id** (str, optional) – identification string of the image stack
- **pyrlevel** (int, optional) – Gauss pyramid level of stack
- **roi_abs** (*list*) – build stack of images cropped in ROI
- **start_idx** (int or datetime) – index or timestamp of first considered image. Note that the timestamp option only works if acq. times can be accessed from filenames for all files in the list (using method *timestamp_to_index()*)
- **stop_idx** (int or datetime, optional) – index of last considered image (if None, the last image in this list is used). Note that the timestamp option only works if acq. times can be accessed from filenames for all files in the list (using method *timestamp_to_index()*)
- **ref_check_roi_abs** (list, optional) – rectangular area specifying a reference area which can be specified in combination with the following 2 parameters in order to include only images in the stack that are within a certain intensity range within this ROI (Note that this ROI needs to be specified in absolute coordinate, i.e. corresponding to pyrlevel 0).
- **ref_check_min_val** (float, optional) – if attribute roi_ref_check is a valid ROI, then only images are included in the stack that exceed the specified intensity value (can e.g. be optical density or minimum gas CD in calib mode)
- **ref_check_max_val** (float, optional) – if attribute roi_ref_check is a valid ROI, then only images are included in the stack that are smaller than the specified intensity value (can e.g. be optical density or minimum gas CD in calib mode)
- **dtype** – data type of stack

Returns image stack containing stacked images

Return type *ImgStack*

get_mean_img (*start_idx=0*, *stop_idx=None*)

Determine an average image from a number of list images.

Parameters

- **start_idx** (int or datetime) – index or timestamp of first considered image. Note that the timestamp option only works if acq. times can be accessed from filenames for all files in the list (using method *timestamp_to_index()*)
- **stop_idx** (int or datetime, optional) – index of last considered image (if None, the last image in this list is used). Note that the timestamp option only works if acq. times can be accessed from filenames for all files in the list (using method *timestamp_to_index()*)

Returns average image

Return type *Img*

get_mean_tseries_rects (*start_idx*, *stop_idx*, **rois*)

Similar to *get_mean_value()* but for multiple rects.

Parameters

- **start_idx** (int or datetime) – index or timestamp of first considered image. Note that the timestamp option only works if acq. times can be accessed from filenames for all files in the list (using method *timestamp_to_index()*)

- **stop_idx** (int or datetime) – index of last considered image (if None, the last image in this list is used). Note that the timestamp option only works if acq. times can be accessed from filenames for all files in the list (using method *timestamp_to_index()*)
- ***rois** – non keyword args specifying rectangles for data access

Returns N-element tuple containing PixelMeanTimeSeries objects (one for each ROI specified on input)

Return type tuple

get_mean_value(*start_idx=0, stop_idx=None, roi=[0, 0, 9999, 9999], apply_img_prep=True*)
Determine pixel mean value time series in ROI.

Determines the mean pixel value (and standard deviation) for all images in this list. Default ROI is the whole image and can be set via input param roi, image preparation can be turned on or off.

Parameters

- **start_idx** (int or datetime) – index or timestamp of first considered image. Note that the timestamp option only works if acq. times can be accessed from filenames for all files in the list (using method *timestamp_to_index()*)
- **stop_idx** (int or datetime) – index of last considered image (if None, the last image in this list is used). Note that the timestamp option only works if acq. times can be accessed from filenames for all files in the list (using method *timestamp_to_index()*)
- **roi** (*list*) – rectangular region of interest [x0, y0, x1, y1], defaults to [0, 0, 9999, 9999] (i.e. whole image)
- **apply_img_prep** (*bool*) – if True, img preparation is performed as specified in self.img_prep dictionary, defaults to True

Returns time series of retrieved values

Return type *PixelMeanTimeSeries*

current_edit()
Return edit_log of current image.

edit_info()
Print the current image preparation settings.

add_gaussian_blurring(*sigma=1*)
Increase amount of gaussian blurring on image load.

Parameters **sigma (1)** (*int*) – Add width gaussian blurring kernel

cam_id()
Get the current camera ID (if camera is available).

current_time()
Get the acquisition time of the current image from image meta data.

Raises **IndexError** – if list does not contain images

Returns start acquisition time of currently loaded image

Return type datetime

current_time_str(*format='%H:%M:%S'*)
Return a string of the current acq time.

current_img(*key='this'*)
Get the current image object.

Parameters **key** (*str*) – this" or "next"

Returns currently loaded image in list

Return type *Img*

show_current (***kwargs*)
Show the current image.

append (*file_path*)
Append image file to list.

Parameters **file_path** (*str*) – valid file path

plot_mean_value (*roi=[0, 0, 9999, 9999], yerr=False, ax=None*)
Plot mean value of image time series.

Parameters

- **roi** (*list*) – rectangular ROI in which mean is determined (default is `[0, 0, 9999, 9999]`, i.e. whole image)
- **yerr** (*bool*) – include errorbars (std), defaults to False
- **ax** (`Axes`, optional) – matplotlib axes object

Returns matplotlib axes object

Return type Axes

plot_tseries_vert_profile (*pos_x, start_y=0, stop_y=None, step_size=0.1, blur=4*)
Plot the temporal evolution of a line profile.

Parameters

- **pos_x** (*int*) – number of pixel column
- **start_y** (*int*) – Start row of profile (y coordinate, default: 10)
- **stop_y** (*int*) – Stop row of profile (is set to rownum - 10pix if input is None)
- **step_size** (*float*) – stretch between different line profiles of the evolution (0.1)
- **blur** (*int*) – blurring of individual profiles (4)

Returns figure containing result plot

Return type Figure

class pyplis.imagelists.**DarkImgList** (*files=None, list_id=None, list_type=None, read_gain=0, camera=None, init=True*)
A :class:'BaseImgList'object only extended by read_gain value.

This class is meant for storage of dark and offset images.

Note: It is recommended to perform the dark and offset correction using non-edited raw dark offset images. Therefore, the default edit state of these list (`edit_active`) is set to False. This means, if you have such a list and want to add blurring, cropping, etc., you first have to activate the image edit on image load via the `edit_active`.

__init__ (*files=None, list_id=None, list_type=None, read_gain=0, camera=None, init=True*)
x.__init__(...) initializes x; see help(type(x)) for signature

class pyplis.imagelists.**AutoDilcorrSettings**(*tau_thresh=0.05*, *erosion_kernel_size=0*, *dilation_kernel_size=0*)

Store settings for automatic dilution correction in ImgLists.

tau_thresh

OD threshold for computation of plume pixel mask

Type float

erosion_kernel_size

size of erosion kernel applied to plume pixel mask

Type int

dilation_kernel_size

size of dilation kernel applied to plume pixel mask after erosion was applied

Type int

bg_model

plume background model used to compute tau images (i.e. correction mode 99, is e.g. used in `_apply_edit()` of *ImgList*)

Type *PlumeBackgroundModel*

Parameters

- **tau_thresh** (*float*) – OD threshold for computation of plume pixel mask
- **erosion_kernel_size** (*int*) – size of erosion kernel applied to plume pixel mask
- **dilation_kernel_size** (*int*) – size of dilation kernel applied to plume pixel mask after erosion was applied

__init__(*tau_thresh=0.05*, *erosion_kernel_size=0*, *dilation_kernel_size=0*)

x.__init__(. . .) initializes x; see help(type(x)) for signature

class pyplis.imagelists.**ImgList**(*files=None*, *list_id=None*, *list_type=None*, *camera=None*, *geometry=None*, *init=True*, ***dilcorr_settings*)

Image list object with expanded functionality (cf. *BaseImgList*).

Additional features:

1. Optical flow determination
2. Linking of lists (e.g. on and offband lists)
3. Dark and offset image correction
4. Plume background modelling and tau image determination
5. Methods for dilution correction
6. Automatic vignetting correction
7. Assignment of calibration data and automatic image calibration

Parameters

- **files** (*list*) – list containing image file paths, defaults to `[]` (i.e. empty list)
- **list_id** (str, optional) – string ID of this list, defaults to None
- **list_type** (str, optional) – string specifying type of image data in this list (e.g. on, off)
- **camera** (Camera, optional) – camera specifications, defaults to None

- **geometry** (MeasGeometry, optional) – measurement geometry
- **init** (*bool*) – if True, the first two images in list files are loaded
- ****dilcorr_settings** – additional keyword args corresponding to settings for automatic dilution correction passed to __init__ of :class:´AutoDilcorrSettings‘

__init__ (*files=None*, *list_id=None*, *list_type=None*, *camera=None*, *geometry=None*, *init=True*, ***dilcorr_settings*)
x.__init__(. . .) initializes x; see help(type(x)) for signature

darkcorr_opt
Return the current dark correction mode.

The following modes are available:

0 => no dark correction possible (is e.g. set if camera is unspecified)

1 => individual correction with separate dark and offset (e.g. ECII data)

2 => one dark image which is subtracted (including the offset, e.g. HD cam data)

For details see documentation of CameraBaseInfo

darkcorr_mode
Return current list darkcorr mode.

optflow_mode
Activate / deactivate optical flow calc on image load.

vigncorr_mode
Activate / deactivate vignetting correction on image load.

dilcorr_mode
Activate / deactivate dilution correction on image load.

sensitivity_corr_mode
Activate / deactivate AA sensitivity correction mode.

tau_mode
Return current list tau mode.

shift_mode
Return current list registration shift mode.

aa_mode
Return current list AA mode.

calib_mode
Acitivate / deactivate current list gas calibration mode.

ext_coeff
Return current extinction coefficient.

ext_coeffs
Dilution extinction coefficients.

bg_img
Return background image based on current vignetting corr setting.

dark_img
Return current dark image.

bg_list
Return background image list (if assigned).

senscorr_mask
Get / set AA correction mask.

calib_data
Get set object to perform calibration.

doas_fov
Try access DOAS FOV info (in case cailbration data is available).

init_bg_model(***kwargs*)
Init clear sky reference areas in background model.

activate_darkcorr(*value=True*)
Activate or deactivate dark and offset correction of images.

If dark correction turned on, dark image access is attempted, if that fails, Exception is raised including information what did not work out.

Parameters **val** (`bool`) – new mode

activate_vigncorr(*value=True*)
Activate / deactivate vignetting correction on image load.

Note: Requires `self.vign_mask` to be set or an background image to be available (from which `self.vign_mask` is then determined)

Parameters **value** (`bool`) – new mode

activate_shift_mode(*value=True*)
Activate / deactivate image shift on load.

The shift that is set in the assigned Camera class is used

Parameters **value** (`bool`) – new mode

activate_tau_mode(*value=True*)
Activate tau mode.

In tau mode, images will be loaded as tau images (if background image data is available).

Parameters **value** (`bool`) – new mode

activate_aa_mode(*value=True*)
Activates AA mode (i.e. images are loaded as AA images).

In order for this to work, the following prerequisites need to be fulfilled:

1. This list needs to be an on band list (`self.list_type = "on"`) #. At least one offband list must be linked to this list (if more offband lists are linked and input param off_id is unspecified, then the first offband list found is used) #. The number of images in the off band list must exceed a minimum of 50% of the images in this list

Parameters **val** (`bool`) – Activate / deactivate AA mode

activate_calib_mode(*value=True*)
Activate calibration mode.

activate_dilcorr_mode(*value=True*)
Activate dilution correction mode.

Please see `correct_dilution()` for details.

Parameters **value** (*bool*) – New mode: True or False

activate_optflow_mode (*value=True*, *draw=False*)
Activate / deactivate optical flow calculation on image load.

Parameters

- **val** (*bool*) – activate / deactivate
- **draw** (*bool*) – if True, flow field is plotted into current image

get_dark_image (*key='this'*)
Prepare the current dark image dependent on `darkcorr_opt`.

The code checks current dark correction mode and, if applicable, prepares the dark image.

1. **self.darkcorr_opt == 0 (no dark correction)** return False
2. **self.darkcorr_opt == 1 (model dark image from a sample dark** and offset image) Try to access current dark and offset image from `self.dark_lists` and `self.offset_lists` (so these must exist). If this fails for some reason, set `self.darkcorr_opt = 2`, else model dark image using `model_dark_image()` and return this image
3. **self.darkcorr_opt == 2 (subtract dark image if exposure** times of current image does not deviate by more than 20% to current dark image) Try access current dark image in `self.dark_lists`, if this fails, try to access current dark image in `self.darkImg` (which can be set manually using `set_dark_image()`). If this also fails, set `self.darkcorr_opt = 0` and return False. If a dark image could be found and the exposure time differs by more than 20%, set `self.darkcorr_opt = 0` and raise ValueError. Else, return this dark image.

get_off_list (*list_id=None*)
Search off band list in linked lists.

Parameters **list_id** (`str`, optional) – ID of the list. If unspecified (None), the default off band filter key is attempted to be accessed (`self.camera.filter_setup.default_key_off`) and if this fails, the first off band list found is returned.

Raises **AttributeError** – if not offband list can be assigned

Returns the corresponding off-band list

Return type *ImgList*

set_bg_img (*bg_img*)
Update the current background image object.

Check input background image and, in case a vignetting mask is not available in this list, determine a vignetting mask from the background image. Furthermore, if the input image is not blurred it is blurred using current list blurring factor and in case the latter is 0, then it is blurred with a Gaussian filter of width 1.

Parameters **bg_img** (`Img`) – the background image object used for plume background modelling (modes 1 - 6 in `PlumeBackgroundModel`)

set_bg_list (*lst*, *always_reload=False*)
Assign background image list to this list.

Assigns and links an image list containing background images to this list. Similar to other linked lists, the index of the current BG image is automatically updated such that the current BG image is closest in time to the current image in this list. Please note also, that a single master BG image can be assigned using *bg_img*.

Parameters

- **lst** (`ImgList`) – image list containing background images. Note that the input can also be a string specifying the list_id of an image list that is already linked to this list.
- **always_reload** (*bool*) – if True, the current BG image is always reloaded, whenever the index in this list is changed (not recommended since it is slow). If False, the state of the background list is only changed, if the actual background image index is altered.

set_flow_images()
Update images for optical flow determination.

The images are updated in `optflow` (`OptflowFarneback` object) using method `set_images()`

Raises **IndexError** – object, i.e. *self.loaded_images["this"]* and *self.loaded_images["next"]*

set_optical_flow(*optflow*)
Set the current optical flow object.

Currently only support for type `OptflowFarneback`

Parameters **optflow** (`OptflowFarneback`) – the optical flow engine

set_darkcorr_mode(*mode*)
Update dark correction mode.

Parameters **mode (1)** (*int*) – new mode

add_master_dark_image(*dark, acq_time=datetime.datetime(1900, 1, 1, 0, 0), texp=0.0, read_gain=0*)
Add a (master) dark image data to list.

Sets a dark image, which is used for dark correction in case, no dark / offset image lists are linked to this object or the data extraction from these lists does not work for some reason.

Parameters

- **ndarray) dark** (*(Img,*) – dark image data
- **acq_time** (*datetime*) – image acquisition time (only updated if input image is numpy array or if acqtime in Img object is default), default: (1900, 1, 1)
- **texp** (*float*) – optional input for exposure time in units of s (i.e. is used if img input is ndarray or if exposure time is not set in the input img)

The image is stored at:

```
stored at self.master_dark
```

add_master_offset_image(*offset, acq_time=datetime.datetime(1900, 1, 1, 0, 0), texp=0.0, read_gain=0*)
Add a (master) offset image to list.

Sets a offset image, which is used for dark correction in case, no dark / offset image lists are linked to this object or the data extraction from these lists does not work for some reason.

Parameters

- **ndarray) offset** (*(Img,*) – offset image data
- **acq_time** (*datetime*) – image acquisition time (only used if input image is numpy array or if acqtime in Img object is default)
- **texp** (*float*) – optional input for exposure time in units of s (i.e. is used if img input is ndarray or if exposure time is not set in the input img)

The image is stored at:

```
self.master_offset
```

set_closest_dark_offset()
Update the index of the current dark and offset images.

The index is updated in all existing dark and offset lists.

link_imglist(*other_list*, *list_id=None*, *always_reload=True*)
Link another image list to this list.

Parameters

- **other_list** (ImgList) – image list object that is supposed to be linked to this one
- **always_reload** (*bool*) – if True, the current image in the linked list is always reloaded, whenever the index in this list is changed. This is useful in case an offband list is linked to an onband list, not so much if a list containing BG images is linked to an oband list (see also `set_bg_list()`)

disconnect_linked_imglist(*list_id*)
Disconnect a linked list from this object.

Parameters **list_id** (*str*) – string id of linked list

link_dark_offset_lists(**lists*)
Assign dark and offset image lists to this object.

Assign dark and offset image lists: get "closest-in-time" indices of dark list with respect to the capture times of the images in this list. Then get "closest-in-time" indices of offset list with respect to dark list. The latter is done to ensure, that dark and offset set used for image correction are recorded subsequently and not individual from each other (i.e. only closest in time to the current image)

change_index_linked_lists()
Update current index in all linked lists based on `cfn`.

load()
Try load current and next image.

goto_next()
Load next image in list.

optflow_histo_analysis(*lines=None*, *start_idx=0*, *stop_idx=None*, *intensity_thresh=0*, ***optflow_settings*)
Perform optical flow histogram analysis for list images.

The analysis is performed for all list images within the specified index (or time) range and for an arbitraty number of PCS lines.

Parameters

- **lines** (*list*) – list containing `LineOnImage` instances
- **start_idx** (int or datetime) – index or timestamp of first considered image. Note that the timestamp option only works if acq. times can be accessed from filenames for all files in the list (using method `timestamp_to_index()`)
- **stop_idx** (int or datetime, optional) – index of last considered image (if None, the last image in this list is used). Note that the timestamp option only works if acq. times can be accessed from filenames for all files in the list (using method `timestamp_to_index()`)
- **intensity_thresh** (*float*) – additional intensity threshold that may, e.g. be used to identify plume pixels (e.g. if list is in `tau_mode`).

- ****optflow_settings** – additional keyword args passed to `OptflowFarneback`

Returns list containing the computed time series of optical flow histogram parameters (`LocalPlumeProperties` instances) for each of the provided input `LineOnImage` objects.

Return type list

get_thresh_mask(*thresh=None*, *this_and_next=True*)
Get bool mask based on intensity threshold.

Parameters

- **thresh** (`float`, optional) – intensity threshold
- **this_and_next** (*bool*) – if True, uses the current AND next image to determine mask

Returns mask specifying pixels that exceed the threshold

Return type array

det_vign_mask_from_bg_img()
Determine vignetting mask from current background image.

The mask is determined using a blurred ($\sigma = 3$) background image which is normalised to one.

The mask is stored in `self.vign_mask`

Returns vignetting mask

Return type *Img*

calc_sky_background_mask(*lower_thresh=None*, *apply_movement_search=True*, ***settings_movement_search*)
Retrieve and set background mask for 2D poly surface fit.

Calculates mask specifying sky radiance pixels for background modelling mode 0. The mask is updated in the background model (class attribute `bg_model`).

Parameters

- **lower_thresh** (`float`, optional) – lower intensity threshold. If provided, this value is used, else, the minimum value is derived from the minimum intensity in the plume image within the current 3 sky reference rectangles
- ****settings_movement_search** – additional keyword arguments passed to `find_movement()`. Note that these may include settings for the optical flow calculation which are further passed to the initiation of the `FarnebackSettings` class

Returns 2D-numpy boolean numpy array specifying sky background pixels

Return type array

calc_plumepix_mask(*od_img*, *tau_thresh=0.05*, *erosion_kernel_size=0*, *dilation_kernel_size=0*)
Calculate plume pixel mask from an OD image using a OD thrshold.

The method further allows to close gaps using a suitable combination of an erosion

correct_dilution(*img*, *plume_bg_vigncorr=None*, *plume_pix_mask=None*, *plume_dists=None*, *ext_coeff=None*, *tau_thresh=0.05*, *vigncorr_mask=None*, *erosion_kernel_size=0*, *dilation_kernel_size=0*, *img_check_plumemask=True*)
Correct a plume image for the signal dilution effect.

The provided plume image needs to be in intensity space, meaning the pixel values need to be intensities and not optical densities or calibrated gas-CDs. The correction is based on Campion et al., 2015 and requires knowledge of the atmospheric scattering extinction coefficients (`ext_coeff`) in the viewing direction of the camera. These can be provided using the corresponding input parameter `ext_coeff` or can be assigned to the list beforehand (up to you). See example script no. 11 to check out how you can retrieve the extinction coefficients using dark terrain features in the plume image. The correction furthermore requires knowledge of the plume distance (in the best case on the pixel-level) and a binary mask specifying plume image pixels. If the latter is not provided on input, it is computed within this function by calculating an OD (tau or AA) image (based on current list state) and by applying a specified OD threshold. Thus, in case no mask is provided, it must be possible to compute optical density images in this list, hence the `bg_model` (instance of `PlumeBackgroundModel`) needs to be ready for tau image computation. In addition, a vignetting correction mask must be available.

Parameters

- **img** (`Img`) – the plume image object
- **plume_bg_vigncorr** (`Img`, optional) – vignetting corrected plume background image used for dilution correction
- **plume_pix_mask** (`Img`, optional) – binary mask specifying plume pixels in the image, is retrieved automatically if input is None
- **plume_dists** (`ndarray` or `Img`, optional) – 2D array containing pixel based plume distances. If None, this mask will be attempted to be retrieved from the `MeasGeometry` instance assigned to this list
- **ext_coeff** (`float`, optional) – atmospheric extinction coefficient. If unspecified, try access via *`ext_coeff`* which returns the current extinction coefficient and raises `AttributeError` in case, no coeffs are assigned to this list
- **tau_thresh** (*`float`*) – OD (tau) threshold to compute plume pixel mask (irrelevant if next **:param:'plume_pix_mask'** is provided)
- **vigncorr_mask** (`ndarray` or `Img`, optional) – mask used for vignetting correction
- **erosion_kernel_size** (*`int`*) – if not zero, the morphological operation erosion is applied to the plume pixel mask (e.g. to remove noise outliers) using an appropriate quadratic kernel corresponding to the input size
- **dilation_kernel_size** (*`int`*) – if not zero, the morphological operation dilation is applied to the plume pixel mask (e.g. to slightly extend the borders of the detected plume) using an appropriate quadratic kernel corresponding to the input size
- **img_check_plumemask** (*`bool`*) – if True, the current dark and vignetting correction states of plume and BG images are checked before computation of the plume background and, if applicable, the plume pixel mask

Returns

3-element tuple containing

- `Img`, dilution corrected input image (vignetting corrected)
- `Img`, vignetting corrected plume background used for the

correction (is computed within this function body if not providedon input - `array`, mask specifying plume pixels

Return type tuple

correct_dilution_all(*tau_thresh=0.05*, *ext_on=None*, *ext_off=None*, *add_off_list=True*, *save_dir=None*, *save_masks=False*, *save_bg_imgs=False*, *save_tau_prev=False*, *vmin_tau_prev=None*, *vmax_tau_prev=None*, ***kwargs*)

Correct all images for signal dilution.

Correct and save all images in this list for the signal dilution effect. See `correct_dilution()` and `prep_data_dilutioncorr()` for details about requirements and additional input options.

Note: The vignetting and dilution corrected images are stored with all additional image preparation settings applied (e.g. dark correction, blurring)

Parameters

- **tau_thresh** (`float`, optional) – tau threshold applied to determine plume pixel mask (retrieved using `tau_mode`, not `aa_mode`)
- **ext_on** (`float`, optional) – atmospheric extinction coefficient at on-band wavelength, if None (default), try access via `ext_coeff`
- **ext_off** (`float`, optional) – atmospheric extinction coefficient at off-band wavelength. Only relevant if input param `add_off_list` is True. If None (default) and `add_off_list=True` try access via `ext_coeff` in off band list.
- **add_off_list** (`bool`) – if True, also the images in a linked off-band image list (using `get_off_list()`) are corrected as well. For the correction of the off-band images, the current plume pixel mask of this list is used.
- **save_dir** (`str`, optional) – base directory for saving the corrected images. If None (default), then a new directory `dilcorr` is created at the storage location of the first image in this list
- **save_masks** (`bool`) – if True, a folder *plume_pix_masks* is created within **:param:'save_dir'** in which all plume pixel masks are stored as FITS
- **save_bg_imgs** (`bool`) – if True, a folder *bg_imgs* is created which is used to store modelled plume background images for each image in this list. This folder can be used on re-import of the data in order to save background modelling time using background modelling mode 99.
- **save_tau_prev** (`bool`) – if True, png previews of dilution corrected tau images are saved
- **vmin_tau_prev** (`float`, optional) – lower tau value for tau image preview plots
- **vmax_tau_prev** (`float`, optional) – upper tau value for tau image preview plots
- ****kwargs** – additional keyword args for dilution correction functions `correct_dilution()` and `prep_data_dilutioncorr()`

import_ext_coeffs_csv(*file_path*, *header_id=None*, ***kwargs*)

Import extinction coefficients from csv.

The text file requires datetime information in the first column and a header which can be used to identify the column. The import is performed using `pandas.read_csv()`

Parameters

- **file_path** (`str`) – the csv data file
- **header_id** (`str`) – header string for column containing ext. coeffs
- ****kwargs** – additionald keyword args passed to `pandas.read_csv()`

Returns pandas Series containing extinction coeffs

Return type Series

Todo: This is a Beta version, insert try / except block after testing

has_bg_img()
Return boolean whether or not background image is available.

update_index_dark_offset_lists()
Check and update current dark image (if possible / applicable).

class pyplis.imagelists.**CellImgList**(*files=None*, *list_id=None*, *list_type=None*, *camera=None*, *geometry=None*, *cell_id=”*, *gas_cd=0.0*, *gas_cd_err=0.0*)

Image list object for cell images.

Whenever cell calibration is performed, one calibration cell is put in front of the lense for a certain time and the camera takes one (or ideally) a certain amount of images.

This image list corresponds to such a list of images with one specific cell in the camera FOV. It is a *BaseImgList* only extended by the variable `self.gas_cd` specifying the amount of gas (column density) in this cell.

__init__(*files=None*, *list_id=None*, *list_type=None*, *camera=None*, *geometry=None*, *cell_id=”*, *gas_cd=0.0*, *gas_cd_err=0.0*)
x.__init__(...) initializes x; see help(type(x)) for signature

update_cell_info(*cell_id*, *gas_cd*, *gas_cd_err*)
Update cell_id and gas_cd amount.

class pyplis.imagelists.**ImgListLayered**(*files=None*, *meta=None*, *list_id=None*, *list_type=None*, *camera=None*, *geometry=None*, *init=True*)

Image list object able to deal with multi layered fits files.

Additional features:

1. Indexing using double index: Filename and image layer
2. Function which returns a DataFrame of all available data

Parameters

- **files** (*list*) – list containing image file paths, defaults to `[]` (i.e. empty list)
- **meta** (*DataFrame*) – meta data from a previous load of the same image list. Can be used alternatively for a faster initialisation
- **list_id** (`str`, optional) – string ID of this list, defaults to None
- **list_type** (`str`, optional) – string specifying type of image data in this list (e.g. on, off)
- **camera** (`Camera`, optional) – camera specifications, defaults to None
- **geometry** (`MeasGeometry`, optional) – measurement geometry
- **init** (*bool*) – if True, the first two images in list `files` are loaded

Note: Initialise with a list of n files each containing a (variable) number of image layers) m_n. The file header needs to be read in for every file in order to get the right amount of total images. The attribute *self.files* will contain m_n copies of the file n. If the list has been loaded before, the ImgListLayered can also be initialised with a DataFrame containing all meta information.

__init__(*files=None, meta=None, list_id=None, list_type=None, camera=None, geometry=None, init=True*)
x.__init__(...) initializes x; see help(type(x)) for signature

get_img_meta_from_filename(*file_path*)
Load and prepare img meta input dict for Img object.

Note: Convenience method only rewritten in order to not break the code. Loads meta data of first image plane in fits file_path

Parameters **file_path** (`str`) – file path of image

Returns dictionary containing retrieved values for `start_acq` and `texp`

Return type dict

get_img_meta_all_filenames()
Return the same data as ImgList.get_img_meta_all_filenames.

Note: Convenience method only rewritten in order to not break the code

Returns

2-element tuple containing

- list, list containing all retrieved acq. time stamps
- list, containing all retrieved exposure times

Return type tuple

get_img_meta_one_fitsfile(*file_path*)
Load all meta data from all image layers of one fits file.

In this form it is custom for the comtessa files TODO: Make general for multilayered fits

get_img_meta_all()
Load all available meta data from fits files.

Returns containing all metadata

Return type dataFrame

5.6 Plume background analysis

Pyplis module containing features related to plume background analysis.

class pyplis.plumebackground.**PlumeBackgroundModel**(*bg_raw=None, plume_init=None, init_surf_fit_mask=True, **kwargs*)
Class for plume background modelling and tau image determination.

Parameters

- **bg_raw** (`Img`) – sky background radiance raw image data

- **plume_init** (Img) – initial plume image data (is used to estimate default clear sky areas for bg modelling)
- ****kwargs** – additional class attributes (e.g. for modelling, valid keys are all keys in self.__dict__.keys())

__init__(*bg_raw=None, plume_init=None, init_surf_fit_mask=True, **kwargs*)
x.__init__(…) initializes x; see help(type(x)) for signature

scale_rect = None
Rectangle for scaline of background image corr modes: 1 - 6

ygrad_rect = None
Rectangle for linear based correction of vertical gradient corr modes: 2, 4

ygrad_line_colnum = None
Settings for quadratic correction of vertical gradient (along line) corr modes: 3, 5, 6

xgrad_rect = None
Rectangle for linear based correction of horizontal gradient (applied before ygradient correction is performed) corr modes: 4, 5

xgrad_line_rownum = None
Settings for quadratic correction of horizontal gradient (along line) corr modes: 6

all_modes
List containing valid modelling modes.

mode
Return current modelling mode.

CORR_MODE
Return current background modelling mode.

surface_fit_mask
fit 2D polynomial.

Type Mask for retrieval mode 0

check_settings()
Check if any of the modelling settings is not specified.

mean_in_rects(*img*)
Determine `(mean, min, max)` intensity in reference rectangles.

Parameters **img** (*array*) – image data array (can also be `Img`)

Returns 2-element tuple containing mean value and error

Return type tuple

update(***kwargs*)
Update class attributes.

Parameters ****kwargs** –

guess_missing_settings(*plume_img*)
Call and return *set_missing_ref_areas()*.

Note: This is the previous name of the method *set_missing_ref_areas()*

set_missing_ref_areas(*plume_img*)
Find and set missing default sky reference areas for modelling.

Based on the input plume image, the clear sky reference areas for sky radiance image based tau modelling are estimated, i.e.:

1. **The rectangle areas for scaling and linear gradient corrections:** `self.scale_rect, self.ygrad_rect, self.xgrad_rect`

2. Coordinates of horizontal and vertical profile lines for quadratic gradient corrections: `self.ygrad_line_colnum, self.xgrad_line_rownum` (i.e. positions and start / stop pixel coordinates)

The estimation is performed based on a brightness analysis for left and right image area.

Parameters **plume_img** (`Img`) – exemplary plume image (should be representative for a whole dataset)

calc_sky_background_mask(*plume_img*, *next_img=None*, *lower_thresh=None*, *apply_movement_search=True*, ***settings_movement_search*)

Retrieve and set background mask for 2D poly surface fit.

Wrapper for method *find_sky_background()*

Calculates mask specifying sky radiance pixels for modelling mode 0 (plume background retrieval directly from plume image without an additional I0-image, using a 2D polynomial surface fit). The mask is stored in *surface_fit_mask*.

Parameters

- **plume_img** (`Img`) – the plume image for which the sky background pixels are supposed to be detected
- **next_img** (`Img`, optional) – second image used to compute the optical flow in **:param:‘plume_img‘**
- **lower_thresh** (`float`, optional) – lower intensity threshold. If provided, this value is used, else, the minimum value is derived from the minimum intensity in the plume image within the current 3 sky reference rectangles
- ****settings_movement_search** – additional keyword arguments passed to `find_movement()`. Note that these may include settings for the optical flow calculation which are further passed to the initiation of the `FarnebackSettings` class

Returns 2D-numpy boolean numpy array specifying sky background pixels

Return type array

bg_from_poly_surface_fit(*plume*, *mask=None*, *polyorder=2*, *pyrlevel=4*)

Apply poly surface fit to plume image for bg retrieval.

Parameters

- **plume** (`Img`) – plume image
- **mask** (*ndarray*) – mask specifying gas free areas (if None, use all pixels). Note that the mask needs to be computed within the same ROI and at the same pyrlevel as the input plume image
- **polyorder** (*int*) – order of polynomial used for fit, defaults to 4
- **pyrlevel** (*int*) – pyramid level in which fit is performed (e.g. 4 => image size for fit is reduced by factor 2^4 = 16). Note that the

:param : second: `PolySurfaceFit` object :type : return tuple: 1st entry: fitted background image

Returns fitted sky background

Return type ndarray

Note: The `PolySurfaceFit` object used to retrieve the background is stored in the `_last_surffit`.

get_tau_image (*plume_img*, *bg_img=None*, *check_state=True*, ***kwargs*)
Determine current tau image for input plume image.

Parameters

- **plume_img** (`Img`) – plume image in intensity space
- **bg_img** (`Img`, optional) – sky radiance image (for `self.CORR_MODE = 1 - 6`)
- **check_state** (*bool*) – if True and current mode != 0, it is checked whether the input images (plume and bg) have the same darkcorrection and vignetting state
- ****kwargs** – additional keyword arguments for updating current settings (valid input keywords (strings): mode, ygrad_rect, ygrad_line_colnum, ygrad_line_startrow, ygrad_line_stoprow

Returns plume tau image

Return type *Img*

Raises **AttributeError** – if input image is already a tau or AA image or if input plume image and the current background image have different states with regard to vignetting or dark correction.

get_aa_image (*plume_on*, *plume_off*, *bg_on=None*, *bg_off=None*, *check_state=True*, ***kwargs*)
Retrieve apparent absorbance image from on and off imgs.

Determines an initial AA image based on input plume and background images and

Parameters

- **plume_on** (`Img`) – on-band plume image
- **plume_off** (`Img`) – off-band plume image
- **bg_on** (`Img`, optional) – on-band sky radiance image (for `self.CORR_MODE = 1 - 6`)
- **bg_off** (`Img`, optional) – off-band sky radiance image (for `self.CORR_MODE = 1 - 6`)
- **check_state** (*bool*) – if True and current mode != 0, it is checked whether the input images (plume and bg) have the same darkcorrection and vignetting state
- ****kwargs** – additional keyword arguments for updating current settings (valid input keywords (strings), e.g. `surface_fit_mask` if `mode == 0`

Returns plume AA image

Return type *Img*

correct_tau_curvature_ref_areas (*tau_init*)
Scale and correct curvature in initial tau image.

The method used is depends on the current `CORR_MODE`. This method only applies for correction modes 1-6.

Parameters **tau_init** (`array`, `Img`) – inital tau image

Returns modelled tau image

Return type array

plot_sky_reference_areas(*plume*)
Plot the current sky ref areas into a plume image.

plot_tau_result(*tau_img=None, tau_min=None, tau_max=None, edit_profile_labels=True, legend_loc=3, figheight=8, add_mode_info=False, fsize_legend=12, fsize_labels=16, **add_lines*)
Plot current tau image including all reference areas.

Parameters

- **tau_img** (Img) – the tau image to be displayed
- **tau_min** (float, optional) – lower tau boundary to be displayed
- **tau_max** (float, optional) – upper tau boundary for colormap
- **edit_profile_labels** (*bool*) – beta version of smart layout for axis labels from profile subplots
- **legend_loc** (*int*) – number ID for specifying legend position
- **figheight** (*int*) – figure height in inches (dpi=matplotlib default)
- **add_mode_info** (*bool*) – if True, information about the used correction mode is included in the plot
- ****kwargs** –

  additional lines to be plotted, e.g.:: pcs = [300, 400, 500, 600]

settings_dict()
Write current sky reference areas and masks into dictionary.

mode_info_dict
Return information on available bg modelling modes.

mode_info(*mode_num*)
Return short information about one of the available modelling modes.

Parameters **mode_num** (*int*) – the background modelling mode for which information is required

Returns short description of mode

Return type str

print_mode_info()
Print information about the different correction modes.

pyplis.plumebackground.**scale_tau_img**(*tau, rect*)
Scale tau image such that it fulfills tau==0 in reference area.

pyplis.plumebackground.**scale_bg_img**(*bg, plume, rect*)
Normalise background image to plume image intensity in input rect.

Parameters

- **bg** (*ndarray*) – background image
- **plume** (*ndarray*) – plume image
- **rect** (*list*) – list containing rectangle coordinates

Returns the scaled background image

Return type ndarray

pyplis.plumebackground.**corr_tau_curvature_vert_two_rects**(*tau0*, *r0*, *r1*)

Apply vertical linear background curvature correction to tau img.

Retrieves pixel mean value from two rectangular areas and determines linear offset function based on the vertical positions of the rectangle center coordinates. The corresponding offset for each image row is then subtracted from the input tau image

Parameters

- **Img) tau0** (*(ndarray,)*) – inital tau image
- **r0** (*list*) – 1st rectanglular area "[x0, y0, x1, y1]'
- **r1** (*list*) – 2nd rectanglular area "[x0, y0, x1, y1]'

Return ndarray modified tau image

pyplis.plumebackground.**corr_tau_curvature_hor_two_rects**(*tau0*, *r0*, *r1*)

Apply horizonzal linear background curvature correction to tau img.

Retrieves pixel mean values from two rectangular areas and determines linear offset function based on the horizontal positions of the rectangle center coordinates. The corresponding offset for each image row is then subtracted from the input tau image.

Parameters

- **Img) tau0** (*(ndarray,)*) – inital tau image
- **r0** (*list*) – 1st rectanglular area "[x0, y0, x1, y1]'
- **r1** (*list*) – 2nd rectanglular area "[x0, y0, x1, y1]'

Return ndarray modified tau image

pyplis.plumebackground.**corr_tau_curvature_vert_line**(*tau0*, *pos_x*, *start_y=0*, *stop_y=None*, *row_mask=None*, *polyorder=2*)

Correct vertical tau curvature using selected row indices of vertical line.

Parameters

- **Img) tau0** (*(ndarray,)*) – inital tau image
- **pos_x** (*int*) – x position of line (column number)
- **start_y** (*int*) – first considered vertical index for fit (0)
- **stop_y** (*int*) – last considered vertical index for fit (is set to last row number if unspecified)
- **row_mask** (*ndarray*) – boolean mask specifying considered row indices (if valid, params start_y, stop_y are not considered)
- **polyorder** (*int*) – order of polynomial to fit curvature

return tuple: 1st entry: modified tau image, second: fitted polynomial

pyplis.plumebackground.**corr_tau_curvature_hor_line**(*tau0*, *pos_y*, *start_x=0*, *stop_x=None*, *col_mask=None*, *polyorder=2*)

Correct vertical tau curvature using selected row indices of vertical line.

Parameters

- **Img) tau0** (*(ndarray,)*) – inital tau image
- **pos_y** (*int*) – y position of line (row number)

- **start_x** (*int*) – first considered horizontal index for fit (0)
- **stop_y** (*int*) – last considered horizontal index for fit (is set to last col number if unspecified)
- **col_mask** (*ndarray*) – boolean mask specifying considered column indices (if valid, params start_x, stop_x are not considered)
- **polyorder** (*int*) – order of polynomial to fit curvature

return tuple: 1st entry: modified tau image, second: fitted polynomial

pyplis.plumebackground.**find_sky_reference_areas**(*plume_img*, *sigma_blur=2*, *plot=False*)

Take an input plume image and identify suitable sky reference areas.

pyplis.plumebackground.**plot_sky_reference_areas**(*plume_img*, *settings_dict*, *ax=None*)

Plot provided sky reference areas into a plume image.

Parameters

- **Img) plume_img** (*(ndarray,*) – plume image data
- **settings_dict** (*dict*) – dictionary containing settings (e.g. retrieved using *find_sky_reference_areas()*)

pyplis.plumebackground.**find_sky_background**(*plume_img*, *next_img=None*, *bgmodel_settings_dict=None*, *lower_thresh=None*, *apply_movement_search=True*, ***settings_movement_search*)

Prepare mask for background fit based on analysis of current image.

The mask is determined by applying an intensity threshold to a plume image based on the intensities in 3 sky reference rectangles of an instance of the *PlumeBackgroundModel* object. If not specified on input from an exisiting instance of the class (using **:param:'bgmodel_settings_dict'**), the 3 required reference areas are calculated automatically using *find_sky_reference_areas()*. Alternatively, the lower threshold can be provided on input using **:param:'lower_thresh'**. Furthermore, pixels showing movement between the input image **:param:'plume_img'and a second provided image (i.e. :param:'next_img'**) can be excluded. The detection of movement between the two frames is performed using the movement detection algorithm `find_movement()` (see `plumespeed`).

Note:

1. This is a Beta version
2. **The method requires images in radiances space (i.e. plume and** terrain pixels appear darker than the sky background).
3. **The input plume image should not contain clouds. If it does, it is** highly recommended to make use of the movement detection algorithm

---

Parameters

- **plume_img** (`Img`) – the plume image for which the sky background pixels are supposed to be detected
- **next_img** (`Img`, optional) – second image used to compute the optical flow in **:param:'plume_img'**

- **bgmodel_settings_dict** (*dict*) – dictionary containing information about sky reference areas (e.g. created using `settings_dict()` from an existing instance of the *PlumeBackgroundModel*). If not specified, the required reference rectangles (`scale_rect`, `ygrad_rect`, `xgrad_rect`) are determined automatically using *find_sky_reference_areas()*.
- **lower_thresh** (`float`, optional) – lower intensity threshold. If provided, this value is used rather than the value derived from the 3 sky reference rectangles (see **:param:'bgmodel_settings_dict'**)
- ****settings_movement_search** – additional keyword arguments passed to `find_movement()`. Note that these may include settings for the optical flow calculation which are further passed to the initiation of the `FarnebackSettings` class

Returns 2D-numpy boolean numpy array specifying sky background pixels

Return type array

5.7 Plume velocity analysis

Pyplis module containing features related to plume velocity analysis.

`pyplis.plumespeed.`**`get_veff`**(*normal_vec, dir_mu, dir_sigma, len_mu, len_sigma, pix_dist_m=1.0, del_t=1.0, sigma_tol=2*)

Calculate effective velocity through line element with normal n.

The velocity is estimated based on a provided displacement angle and magnitude which is converted into dx and dy displacement and projected to n using the dot product. Uneceertainties are estimated

Parameters

- **normal_vec** (*array*) – 2D normal vector relative to which the effective velocity is retrieved
- **dir_mu** (*float*) – expectation value of displacement orientation angle in degrees (e.g. retrieved using histogram analysis)
- **dir_sigma** (*float*) – uncertainty of prev. angle
- **len_mu** (*float*) – expectation value of displacement magnitude in units of pixels (e.g. retrieved using histogram analysis)
- **len_sigma** (*float*) – uncertainty of prev. magnitude
- **pix_dist_m** (*float*) – pixel-to-pixel distance in m
- **del_t** (*float*) – time difference corresponding to above displacement information
- **sigma_tol** (*int*) – sigma tolerance factor for uncertainty estimate, defaults to 2

Returns 2-element tuple containing effective velocity and corresponding error

Return type tuple

`pyplis.plumespeed.`**`find_signal_correlation`**(*first_data_vec, next_data_vec, time_stamps=None, reg_grid_tres=None, freq_unit='S', itp_method='linear', max_shift_percent=20, sigma_smooth=1, plot=False, **kwargs*)

Determine cross correlation from two ICA time series.

Parameters

- **first_data_vec** (*array*) – first data vector (i.e. left or before `next_data_vec`)
- **next_data_vec** (*array*) – second data vector (i.e. behind `first_data_vec`)
- **time_stamps** (*array*) – array containing time stamps of the two data vectors. If default (None), then the two vectors are assumed to be sampled on a regular grid and the returned lag corresponds to the index shift with highest correlation. If `len(time_stamps) == len(first_data_vec)` and if entries are datetime objects, then the two input time series are resampled and interpolated onto a regular grid, for resampling and interpolation settings, see following 3 parameters.
- **reg_grid_tres** (*int*) – sampling resolution of resampled time series data in units specified by input parameter `freq_unit`. If None, then the resolution is determined automatically based on the mean time resolution of the data
- **freq_unit** (*str*) – pandas frequency unit (use S for seconds, L for ms)
- **itp_method** (*str*) – interpolation method, choose from `["linear", "quadratic", "cubic"]`
- **max_shift_percent** (*float*) – percentage maximum shift applied to index of **:arg:'first_data_vec'** to find pearson correlation with **:arg:'next_data_vec'**.
- **sigma_smooth** (*int*) – specify width of gaussian blurring kernel applied to data before correlation analysis (default=1)
- **plot** (*bool*) – if True, result is plotted

Returns

5-element tuple containing

- *float*: lag (in units of s or the index, see input specs)
- *array*: retrieved correlation coefficients for all shifts
- *Series*: analysis signal 1. data vector
- *Series*: analysis signal 2. data vector
- *Series*: analysis signal 2. data vector shifted using **''**lag'

Return type tuple

`pyplis.plumespeed.`**`find_signal_correlation_old`**(*first_data_vec, next_data_vec, time_stamps=None, reg_grid_tres=None, freq_unit='S', itp_method='linear', cut_border_idx=0, sigma_smooth=1, plot=False, **kwargs*)

Determine cross correlation from two ICA time series.

This is

Parameters

- **first_data_vec** (*array*) – first data vector (i.e. left or before `next_data_vec`)
- **next_data_vec** (*array*) – second data vector (i.e. behind `first_data_vec`)
- **time_stamps** (*array*) – array containing time stamps of the two data vectors. If default (None), then the two vectors are assumed to be sampled on a regular grid and the returned lag corresponds to the index shift with highest correlation. If `len(time_stamps) == len(first_data_vec)` and if entries are datetime objects, then the two input time series are resampled and interpolated onto a regular grid, for resampling and interpolation settings, see following 3 parameters.

- **reg_grid_tres** (*int*) – sampling resolution of resampled time series data in units specified by input parameter `freq_unit`. If None, then the resolution is determined automatically based on the mean time resolution of the data
- **freq_unit** (*str*) – pandas frequency unit (use S for seconds, L for ms)
- **itp_method** (*str*) – interpolation method, choose from `["linear", "quadratic", "cubic"]`
- **cut_border_idx** (*int*) – number of indices to be removed from both ends of the input arrays (excluded datapoints for cross correlation analysis)
- **sigma_smooth** (*int*) – specify width of gaussian blurring kernel applied to data before correlation analysis (default=1)
- **plot** (*bool*) – if True, result is plotted

Returns

5-element tuple containing

- *float*: lag (in units of s or the index, see input specs)
- *array*: retrieved correlation coefficients for all shifts
- *Series*: analysis signal 1. data vector
- *Series*: analysis signal 2. data vector
- *Series*: analysis signal 2. data vector shifted using ‘‘lag‘

Return type tuple

class pyplis.plumespeed.**VeloCrossCorrEngine**(*imglist=None*, *pcs=None*, *pcs_offset=None*, *meas_geometry=None*, ***settings*)

Class for plume velocity retrieval using cross-correlation method.

This class can be used to calculate the plume velocity based on a cross-correlation analysis of two ICA time-series retrieved from a set of plume images between two (ideally parallel) plume intersection lines.

profile_images

dictionary containing PCS profile images, which are `ProfileTimeSeriesImg` and where the x-axis corresponds to the image index (time) and the y-axis the corresponding line profiles for each of the two PCS lines. These images can be saved as FITS and reload, which can accelarate a reanalysis process (the bottleneck of the cross correlation analysis is loading all images in the list and retrieving the actual PCS profiles along both lines). Valid keys: `pcs` and `pcs_offset`.

Type dict

results

results of cross-correlation analysis, will be filled in `get_velocity()`.

Type dict

Parameters

- **imglist** – Image list used to retrieve time series of integrated column amounts along 2 suitable plume intersections used for velocity retrieval
- **pcs** (`LineOnImage`) – Plume cross section line used for velocity retrieval
- **pcs_offset** (`LineOnImage`) – Second (shifted to `pcs`) PCS line used for velocity retrieval

- **meas_geometry** – optional: `MeasGeometry` object used to calculate in-plume pix-to-pix distances. If None, then the `MeasGeometry` object of the assigned `ImgList` (`imglist`) is used.
- ****settings** – optional keyword args passed to `find_signal_correlation()` (used in class method `get_velocity()`).

__init__(*imglist=None, pcs=None, pcs_offset=None, meas_geometry=None, **settings*)
x.__init__(...) initializes x; see help(type(x)) for signature

update_settings(*settings_dict*)
Update valid settings for cross correlation retrieval.

Parameters **settings_dict** (`dict`) – dictionary containing new settings

velocity
Retrieve plume velocity.

Raises `ValueError` – if velocity is nan (default).

correlation_lag
Time lag showing highest correlation between two ICA time-series.

Raises `ValueError` – if velocity is nan (default).

pcs
Return the PCS line used for the velocity retrieval.

pcs_offset
Return the PCS offset line used for the velocity retrieval.

imglist
Return the image list supposed to be used for the analysis.

meas_geometry
Return measurement geometry from image list.

pcs_profile_pics
Check and, if applicable, return current PCS profile images.

get_pcs_tseries_from_imgstack(*stack, start_idx=0, stop_idx=None*)
Load PCS profile time series pictures from image stack.

Parameters

- **stack** (`ImgStack`) – stack containing OD or AA images
- **start_idx** (`int`) – index of first considered image in list
- **stop_idx** (`int`) – last considered index in image list

Returns 2-element tuple containing `ProfileTimeSeriesImg` for both PCS lines

Return type tuple

get_pcs_tseries_from_list(*start_idx=0, stop_idx=None*)
Load profile time series pictures from AA img list.

Loop over images in image list and extract cross section profiles for each of the 2 provided plume cross section lines. The profiles are written into a `ProfileTimeSeriesImg` which can be stored as FITS file.

Parameters

- **start_idx** (`int`) – index of first considered image in list

- **stop_idx** (*int*) – last considered index in image list

Returns 2-element tuple containing `ProfileTimeSeriesImg` for both PCS lines

Return type tuple

run (***settings*)
Apply correlation algorithm to ICA time series of both lines.

Parameters ****settings** – optional keyword args passed to *find_signal_correlation()*

create_parallel_pcs_offset (*offset_pix=50, color='lime', linestyle='–'*)
Create an offset line to the current PCS used for retrieval.

Parameters **offset_pix** (*int*) – Distance of new line to current PCS line (in normal direction). The distance is calculated in detector coordinates on pyramid level 0.

Returns Translated PCS line

Return type *LineOnImage*

check_list (*lst*)
Check if imglist is ready for velocity analysis.

Parameters **lst** (`BaseImgList`) – the image list object supposed to be checked

Returns True, if list is okay, False if not

Return type bool

get_pix_dist_img (*pyrlevel=0*)
Image specifying pix-to-pix distances for each pixel.

The image is loaded from the current `MeasGeometry` object assigned to the image list.

load_pcs_profile_img (*file_path, line_id='pcs'*)
Try to load ICA profile time series image from FITS file.

Parameters

- **file_path** (`ProfileTimeSeriesImg`) – valid file path to profile time-series image
- **line_id** (*str*) – specify to which line the image belongs

save_pcs_profile_images (*save_dir=None, fname1='profile_tseries_pcs.fts', fname2='profile_tseries_offset.fts'*)
Save current ICA profile time series images as FITS file.

Note: Existing files will be overwritten without warning

Parameters

- **save_dir** (*str*) – Directory where images are saved (if None, use current directory)
- **fname1** (*str*) – name of first profile image
- **fname2** (*str*) – name of second profile image

plot_pcs_lines (*img=None, **kwargs*)
Plot current PCS retrieval lines into image.

Parameters

- **img** – optional: example plume image (Img object). If None, then the current image of *imglist* is used
- ****kwargs** – additional keyword arguments passed to show() of Img object. This can also be used to pass an axes instance using keyword ax, for instance if this is supposed to be plotted into a subplot.

Returns matplotlib axes instance

Return type ax

plot_ica_tseries_overlay(*ylabel=None*, *ax=None*)
Plot the ICA time-series of the analysed signals.

Note: Only works after cross-correlation analysis is performed

plot_corrcoeff_tseries(*add_lag=True*, *ax=None*, ***kwargs*)
Plot time series of correlation coefficients.

Parameters

- **add_lag** (*bool*) – if True, a vertical line is added at x-position showing maximum correlation
- **ax** – matplotlib axes object

class pyplis.plumespeed.**LocalPlumeProperties**(*roi_id=''*, ***kwargs*)
Class to store results about local properties of plume displacement.

This class represents statistical (local) plume (gas) displacement information (e.g. retrieved using an optical flow algorithm). These include the predominant local displacement direction (orientation of displacement vectors) and the corresponding displacement length both including uncertainties (e.g. retrieved from Gauss fits applied to histogram distribution). Further, the time difference between the two frames used to estimate the displacement parameters is stored. This class is for instance used for plume displacement properties derived using local_flow_params() from *OptflowFarneback* which is based on a statistical analysis of histograms derived from a dense optical flow algorithm.

__init__(*roi_id=''*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

start
Acquisistion time of first image.

stop
Start acqusition time of last image.

len_mu
Array containing displacement lengths (unit [pix/del_t]).

len_sigma
Array with errors of displacement lengths (unit [pix/del_t]).

len_mu_norm
Array containing normalised displacement lengths (unit [pix/s]).

len_sigma_norm
Array with errors of normalised displ. lens (unit [pix/s]).

dir_mu
Return current displacement orientation vector.

dir_sigma
Return current displacement orientation std vector.

significance
Significancy of data point.

This array is filled in *get_and_append_from_farneback()*, which calls `local_flow_params()` of *OptflowFarneback* object. The number corresponds to the fraction of pixels used to determine the displacement parameters, relative to the total number of pixels available in the corresponding ROI used. The latter can, for instance, be a rotated ROI around a retrieval line (`LineOnImage` class).

fit_success
Array containing flags whether or not multi-gauss fit was successful.

pyrlevel
Array containing pyramid levels used to determine displ. params.

del_t
Return current del_t vector.

Corresponds to the difference between frames for time series

start_acq
Return current displacement length std vector.

Corresponds to the start acquisition times of the time series

displacement_vectors
All displacement vectors (unit [pix / del_t]).

displacement_vector(*idx=-1*)
Get displacement vector for given index.

The vector is returned in unit [pix / del_t].

Parameters **idx** (*int*) – index

Returns 2-element array containing displacement in x and y direction, i.e. `(dx, dy)`

Return type array

to_pyrlevel(*pyrlevel=0*)
Convert data to a given pyramid level.

apply_significance_thresh(*thresh=0.5*)
Remove all datapoints with significance val below thresh.

Datapoints with significance lower than the provided threshold are converted into NaN. Can be combined with interpolation and clean up.

Parameters **thresh** (*float*) – significance threshold supposed to be applied to data

Returns new object excluding nan values in any of the data arrays

Return type *LocalPlumeProperties*

dropna(***kwargs*)
Drop all indices containing nans.

Remove all indices for which any of the data arrays `len_mu`, `len_sigma`, `dir_mu`, `dir_sigma`, `del_t` contains NaN values using the method *dropna()* of pandas `DataFrame` object.

Parameters ****kwargs** – additional keyword arguments passed to *dropna()* of pandas `DataFrame` object.

Returns new object excluding nan values in any of the data arrays

Return type *LocalPlumeProperties*

interpolate(*time_stamps=None*, ***kwargs*)

Interpolate missing.

Remove all indices for which any of the data arrays `len_mu`, `len_sigma`, `dir_mu`, `dir_sigma`, `del_t` contains NaN values using the method `dropna()` of pandas `DataFrame` object.

Parameters

- **time_stamps** (*array*) – array containing datetime indices supposed to be used for interpolation
- ****kwargs** – additional keyword arguments passed to `dropna()` of pandas `DataFrame` object.

Returns new object excluding nan values in any of the data arrays

Return type *LocalPlumeProperties*

apply_median_filter(*width=5*)

Apply median filter to data.

The filter is only applied to `len_mu` and `dir_mu`, and the corresponding uncertainty arrays `len_sigma` and `dir_sigma`

Note: Creates and returns new `LocalPlumeProperties` instance, the data in this object remains unchanged

Parameters **width** (*int*) – width of 1D median filter

Returns new data object

Return type *LocalPlumeProperties*

apply_gauss_filter(*width=5*)

Apply Gaussian blurring filter to data.

The filter is only applied to `len_mu` and `dir_mu`, and the corresponding uncertainty arrays `len_sigma` and `dir_sigma`

Note: Creates and returns new `LocalPlumeProperties` instance, the data in this object remains unchanged

Parameters **width** (*int*) – width of Gaussian blurring kernel

Returns new data object

Return type *LocalPlumeProperties*

get_and_append_from_farneback(*optflow_farneback*, ***kwargs*)

Retrieve main flow field parameters from Farneback engine.

Calls `local_flow_params()` from `OptflowFarneback` engine and appends the results to the current data

Parameters

- **optflow_farneback** (`OptflowFarneback`) – optical flow engine used for analysis
- ****kwargs** – additional keyword args passed to `local_flow_params()`

get_velocity(*idx=-1*, *pix_dist_m=1.0*, *pix_dist_m_err=None*, *normal_vec=None*, *sigma_tol=2*)
Determine plume velocity from displacements.

Parameters

- **idx** (*int*) – index of results for which velocity is determined
- **pix_dist_m** (*float*) – pixel to pixel distance in m (default is 1.0), e.g. determined using `MeasGeometry` object
- **pix_dist_m_err** (`float`, optional) – uncertainty in pixel distance, if None (default), then 5% of the actual pixel distance is assumed
- **normal_vec** (`tuple`, optional) – normal vector used for scalar product to retrieve effective velocity (e.g. `normal_vector` of a `LineOnImage`) object. If None (default), the normal direction is assumed to be aligned with the displacement direction, i.e. the absolute magnitude of the velocity is retrieved
- **sigma_tol** (*int*) – sigma tolerance level for expectation intervals of orientation angle and displ. magnitude

Returns

2-element tuple containing

- `float`: magnitude of effective velocity
- `float`: uncertainty of effective velocity

Return type tuple

get_orientation_tseries()
Get time series (and uncertainties) of movement direction.

Returns

3-element tuple containing

- `Series`: time series of orientation angles
- `Series`: time series of lower vals (using `dir_sigma`)
- `Series`: time series of upper vals (using `dir_sigma`)

Return type tuple

get_magnitude_tseries(*normalised=True*)
Get time series (and uncertainties) of displacement lengths.

Note: The time series are absolute magnitudes of the retrived displacement lengths and are not considered relative to a certain normal direction.

Parameters **normalised** (*bool*) – if True, the lengths are normalised to a time difference of 1s

Returns

3-element tuple containing

- `Series`: time series of displacement lengths
- `Series`: time series of lower vals (using `len_sigma`)
- `Series`: time series of upper vals (using `len_sigma`)

Return type tuple

plot_directions(*ax=None*, *date_fmt=None*, *yerr=True*, *ls=None*, ***kwargs*)
Plot time series of displacement orientation.

Parameters

- **ax** – optional, matplotlib axes object
- **date_fmt** (*str*) – optional, x label datetime formatting string, passed to `DateFormatter` (e.g. "%H:%M")
- ****kwargs** – additional keyword args passed to plot function of `Series` object

Returns matplotlib axes object

Return type Axes

plot_magnitudes(*normalised=True*, *ax=None*, *date_fmt=None*, *yerr=True*, *ls=None*, ***kwargs*)
Plot time series of displacement magnitudes.

Parameters

- **normalised** (*bool*) – normalise magnitudes to time difference intervals of 1s
- **ax** – optional, matplotlib axes object
- **date_fmt** (*str*) – optional, x label datetime formatting string, passed to `DateFormatter` (e.g. "%H:%M")
- ****kwargs** – additional keyword args passed to plot function of `Series` object

Returns matplotlib axes object

Return type Axes

plot(*date_fmt=None*, *fig=None*, ***kwargs*)
Plot showing detailed information about this time series.

Parameters

- **date_fmt** (*str*) – date string formatting for x-axis
- **fig** (*figure*) – matplotlib figure containing 3 subplots

plot_velocities(*pix_dist_m=None*, *pix_dist_m_err=None*, *ax=None*, *normal_vec=None*, *date_fmt=None*, ***kwargs*)
Plot time series of velocity evolution.

Parameters

- **pix_dist_m** – detector pixel distance in m, if unspecified, then velocities are plotted in units of pix/s
- **pix_dist_m_err** – uncertainty in pixel to pixel distance in m

to_dict()
Write all data attributes into dictionary.

Keys of the dictionary are the private class names

Returns Dictionary containing results

Return type OrderedDict

from_dict (*d*)
Read valid attributes from dictionary.

Parameters **d** (*dict*) – dictionary containing data

Returns this object

Return type *LocalPlumeProperties*

to_pandas_dataframe ()
Convert object into pandas dataframe.

This can, for instance be used to store the data as csv (cf. *from_pandas_dataframe()*)

from_pandas_dataframe (*df*)
Import results from pandas `DataFrame` object.

Parameters **df** (*DataFrame*) – pandas dataframe containing emisison rate results

Returns this object

Return type *LocalPlumeProperties*

default_save_name
Return default name for txt export.

save_txt (*path=None*)
Save this object as text file.

load_txt (*path*)

class pyplis.plumespeed.**FarnebackSettings** (***settings*)
Settings for optical flow Farneback calculations and visualisation.

Todo: Finish docs

This object contains settings for the opencv implementation of the optical flow Farneback algorithm `calcOpticalFlowFarneback()`. For a detailed description of the input parameters see OpenCV docs (last access: 07.03.2017).

Furthermore, it includes attributes for image preparation which are applied to the input images before `calcOpticalFlowFarneback()` is called. Currently, these include contrast changes specified by *i_min* and *i_max* which can be used to specify the range of intensities to be considered.

In addition, post analysis settings of the flow field can be specified, which are relevant, e.g. for a histogram analysis of the retrieved flow field:

1. ***roi_abs*****: specifiy ROI for post analysis of flow field** (`abs` indicates that the input is assumed to be in absolute image coordinates and not in coordinates set based on a cropped or size reduced image). Default corresponds to whole image.

2. ***min_length*****: minimum length of optical flow vectors to be** considered for statistical analysis, default is 1 (pix)

3. ***hist_sigma_tol*****: parameter for retrieval of mean flow** field parameters. It specifies the range of considered orientation angles based on mu and sigma of the main peak of flow field orientation histogram. All vectors falling into this angular range are considered to determine the flow length histogram used to estimate the average displacement length.

4. ***hist_dir_gnum_max*****: maximum allowed number of gaussians for** multi gauss fit of orientation histogram (default = 5).

Parameters `**settings` – valid keyword arguments for class attributes, e.g.:

```
stp = FarnebackSettings(i_min=0, i_max=3500,
                        iterations=8)
```

`__init__`(***settings*)
x.__init__(...) initializes x; see help(type(x)) for signature

`update`(***settings*)
Update current settings.

Parameters `**settings` – keyword args specifying new settings (only valid keys are considered, i.e. class attributes)

`i_min`
Lower intensity limit for image contrast preparation.

`i_max`
Upper intensity limit for image contrast preparation.

`roi_rad`
Old name of *`roi_rad_abs`*.

`roi_rad_abs`
ROI used for measuring min / max intensities for contrast settings.

ROI (in absolute image coords) for updating the intensity range `i_min` / `i_max` (only relevant if *`auto_update`* is True).

`auto_update`
Contrast is automatically updated based on min / max intensities.

If active, then *`i_min`* and *`i_max`* are updated automativally whenever new images are assigned to a *`OptflowFarneback`* using method `set_images()`. The update is performed based on min / max intensities of the images in the current ROI

`pyr_scale`
scale space parameter for pyramid levels.

pyplis default = 0.5

Type Farneback algo input

`levels`
number of pyramid levels.

pyplis default = 4

Type Farneback algo input

`winsize`
width of averaging kernel.

The larger, the more stable the results are, but also more smoothed

pyplis default = 20

Type Farneback algo input

`iterations`
number of iterations.

pyplis default = 5

Type Farneback algo input

poly_n
size of pixel neighbourhood for poly exp.

default = 5

Type Farneback algo input

poly_sigma
std of Gaussian to smooth poly derivatives.

pyplis default = 1.1

Type Farneback algo input

roi_abs
Get ROI for analysis of flow field (in absolute image coords).

min_length
Get / set minimum flow vector length for post analysis.

min_count_frac
Minimum fraction of significant vectors required for histo analysis.

hist_dir_sigma
Old name of `hist_sigma_tol`.

hist_sigma_tol
Sigma tolerance value for mean flow analysis.

hist_dir_gnum_max
Max number of gaussians for multigauss fit of orientation histo.

hist_dir_binres
Angular resolution of orientation histo (bin width, in deg).

disp_skip
Return current pixel skip value for displaying flow field.

disp_len_thresh
Return current pixel skip value for displaying flow field.

duplicate()
Return deepcopy of this object.

class pyplis.plumespeed.**OptflowFarneback**(*first_img=None*, *next_img=None*, ***settings*)
Implementation of Optical flow Farneback algorithm of OpenCV library.

Engine for autmatic optical flow calculation, for settings see `FarnebackSettings`. The calculation of the flow field is performed for two consecutive images.

Includes features for histogram based post analysis of flow field which can be used to estimate flow vectors in low contrast image regions.

Parameters

- **first_img** (`Img`, optional) – first of two consecutive images
- **next_img** (`Img`, optional) – second of two consecutive images

images_input
Dictionary containing the current images used to determine flow field. The images can be updated using `set_images()`. Keys: `this`, `next`

Type dict

images_prep
Dictionary containing modified input images prepared for determining the optical field using `calcOpticalFlowFarneback()` (e.g. contrast changed, converted to 8 bit). Keys: `this`, `next`

Type dict

flow
this attribute contains the flow field (i.e. raw output of `calcOpticalFlowFarneback()`).

Type array

settings
settings class including input specifications for flow calculation (i.e. input args for `calcOpticalFlowFarneback()`) and further, settings for image preparation (before the flow field is calculated, cf. *images_prep*) as well as settings for post analysis of the optical flow field (e.g. for histogram analysis).

Type *FarnebackSettings*

__init__(*first_img=None*, *next_img=None*, ***settings*)
x.__init__(. . .) initializes x; see help(type(x)) for signature

auto_update_contrast
Get / set mode for automatically update contrast range.

If True, the contrast parameters `self.settings.i_min` and `self.settings.i_max` are updated when `set_images`()` is called, based on the min / max intensity of the two images. The latter intensities are retrieved within the current ROI for the flow field analysis (`self.roi_abs`)

reset_flow()
Reset flow field.

roi_abs
Get / set current ROI (in absolute image coordinates).

roi
Get ROI converted to current image preparation settings.

pyrlevel
Return pyramid level of current image.

del_t
Return time difference in s between both images.

current_time
Return acquisition time of current image.

set_mode_auto_update_contrast_range(*value=True*)
Activate auto update of image contrast range.

If this mode is active (the actual parameter is stored in `self._img_prep_modes["update_contrast"]`), then, whenever the optical flow is calculated, the input contrast range is updated based on minimum / maxium intensity of the first input image within the current ROI.

Parameters **value (True)** (*bool*) – new mode

check_contrast_range(*img_data*)
Check input contrast settings for optical flow calculation.

current_contrast_range()
Get min / max intensity values for image preparation.

update_contrast_range()
Update contrast range using min/max vals of current images in ROI.

set_images(*this_img*, *next_img*)
Update the current image objects.

Parameters

- **this_img** (*ndarray*) – the current image
- **next_img** (*ndarray*) – the next image

prep_images()
Prepare images for optical flow input.

calc_flow(*this_img=None*, *next_img=None*)
Calculate the optical flow field.

Uses `cv2.calcOpticalFlowFarneback()` to calculate optical flow field between two images using the input settings specified in `self.settings`.

Parameters

- **this_img** (`Img`) – the first of two successive images (if unspecified, the current images in `self.images_prep` are used, else, they are updated)
- **next_img** (`Img`) – the second of two successive images (if unspecified, the current images in `self.images_prep` are used, else, they are updated)

Returns 3D numpy array containing flow displacement field (is also assigned to *flow*)

Return type array

get_flow_in_roi(*roi_rel=None*)
Get the flow field within in a ROI.

Parameters **roi_rel** (*list*) – rectangular ROI aligned with image axis (`[x0, y0, x1, y1]`).

Note: The ROI is used as is, i.e. it needs to be defined for current Gauss pyramid level.

Returns 3D numpy array containing flow displacement field in ROI

Return type array

prep_flow_for_analysis(*mask=None*)
Get flow field data from all pixels in a certain ROI.

This function provides access to the flow field in a certain region of interest. In the default case the currently set roi *roi* is used (which is a rectangle aligned with the image x / y axis). Alternatively, a pixel access mask can be provided (e.g. specifying pixels in a rotated rectangle) which is then be used.

Parameters **mask** (*array*) – boolean mask specifying all pixels used to retrieve displacement information (True pixels in mask)

Returns

2-element tuple containing

- `array`, vector containing all x displacement lengths
- `array`, vector containing all y displacement lenghts

Return type tuple

to_plume_speed(*col_dist_img*, *row_dist_img=None*)
Convert the current flow field to plume speed array.

Parameters

- **col_dist_img** (`Img`) – image, where each pixel corresponds to horizontal pixel distance in m
- **row_dist_img** – optional, image where each pixel corresponds to vertical pixel distance in m (if None, `col_dist_img` is also used for vertical pixel distances)

get_flow_orientation_img(*in_roi=False*, *roi_rel=None*)
Return flow angle image.

The pixel values correspond to the orientation angles of the vectors of the current flow field, where the values correspond to:

- 0 -> upwards (**-y** direction)
- 90 -> to the right (**+x** direction)
- -90 -> to the left (**-x** direction)
- -180, 180 -> down (**+y** direction)

Parameters

- **in_roi** (*`bool`*) – get the image for a certain ROI
- **roi_rel** (`list`, optional,) – the ROI supposed to be used if `in_roi` is True. If None (default) then the current ROI is used (*`roi`*).

Returns 2D numpy array corresponding to flow orientation image

Return type array

get_flow_vector_length_img(*in_roi=False*, *roi=None*)
Return flow vector length image.

The pixel values correspond to the magnitude of the vectors of the current flow field.

Parameters

- **in_roi** (*`bool`*) – get the image for a certain ROI
- **roi** (`list`, optional,) – the ROI supposed to be used if `in_roi` is True. If None (default) then the current ROI is used (*`roi`*).

Returns 2D numpy array corresponding to flow orientation image

Return type array

all_len_angle_vecs_roi(*mask=None*)
Get lengths and angles for all pixels in a ROI.

Parameters **mask** (`array`, optional) – boolean mask specifying all pixels supposed to be used for data access, defaults to None, in which case the current ROI is used (i.e. *`roi`*)

Returns

2-element tuple containing

- `array`, vector with all displacement lengths in ROI / mask
- `array`, vector with all displacement angles in ROI / mask

Return type tuple

flow_orientation_histo (*pix_mask=None*, *bin_res_degrees=None*, *min_length=1.0*, ***kwargs*)
Get histogram of orientation distribution of current flow field.

Parameters

- **pix_mask** (`array`, optional) – boolean mask specifying image pixels supposed to be considered for the analysis. Is passed to `all_len_angle_vecs_roi()`, i.e. if this mask is unspecified the histogram data is retrieved using the current ROI (`roi`) for specifying the considered image region.

  Note: This is ignored if two arrays containing lengths and angles are provided using `**kwargs` (for details see below)

- **bin_res_degrees** (`int`) – bin width of histogram (is rounded to nearest integer if not devisor of 360), if unspecified use `hist_dir_binres` of settings class
- **min_length** (`float`) – minimum length of vectors in order to be considered for histogram, defaults to 1.0
- ****kwargs** – additional key word args that can be used to pass lens and angles arrays (see e.g. `local_flow_params()`). Use keywords `lens` and `angles` to pass this information.

Returns

3-element tuple containing

- `array`: histogram counts
- `array`: histogram bins
- `array`: all angles used to determine the histogram

Return type tuple

flow_length_histo (*pix_mask=None*, *bin_res_pix=1*, *min_length=1.0*, ***kwargs*)
Get histogram of displacement length distribution of flow field.

Parameters

- **pix_mask** (`array`, optional) – boolean mask specifying image pixels supposed to be considered for the analysis. Is passed to `all_len_angle_vecs_roi()`, i.e. if this mask is unspecified the histogram data is retrieved using the current ROI (`roi`) for specifying the considered image region.

  Note: This is ignored if two arrays containing lengths and angles are provided using `**kwargs` (for details see below)

- **bin_res_pix** (`int`) – bin width in units of pixels, defaults to 2
- **min_length** (`float`) – minimum length of vectors in order to be considered for histogram, defaults to 1.0
- ****kwargs** – additional key word args that can be used to pass lens and angles arrays (see e.g. `local_flow_params()`). Use keyword `lens` to pass this information.

Returns

3-element tuple containing

- `array`: histogram counts

- `array`: histogram bins
- `array`: all lengths used to determine the histogram

Return type tuple

fit_multigauss_to_histo(*count*, *bins*, *noise_amp=None*, *max_num_gaussians=None*)
Fit multi gauss distribution to histogram.

Parameters

- **count** (*array*) – array containing histogram counts
- **bins** (*array*) – array containing bins corresponding to counts
- **noise_amp** (*float*) – noise amplitude of the histogram data (you don't want to fit all the noise peaks). If None, then it is estimated automatically within `MultiGaussFit`.
- **max_num_gaussians** (*int*) – Maximum allowed number of Gaussians for `MultiGaussFit`, if None, then default of `MultiGaussFit` is used

Returns

2-element tuple containing

- *MultiGaussFit*: fit object
- *bool*: success True / False

Return type tuple

fit_orientation_histo(*count*, *bins*, *noise_amp=None*, *max_num_gaussians=None*, ***kwargs*)
Fit multi gauss distribution to flow orientation histogram.

Parameters

- **count** (*array*) – histogram counts (see `flow_orientation_histo()`)
- **bins** (*array*) – histogram bins (see `flow_orientation_histo()`)
- **noise_amp** (`float`, optional) – minimum amplitude required for peaks in histogram in order to be considered for multi gauss fit, if None (default) use 5% of max count
- **max_num_gaussians** (`int`, optional) – maximum number of Gaussians fitted to the distributions, if None (default) then use `self.settings.hist_dir_gnum_max`

Returns

2-element tuple containing

- `MultiGaussFit`, the fit object
- bool, fit success

Return type tuple

mu_sigma_from_moments(*count*, *bins*)
Get mean and sigma of histogram distr. using 1. and 2nd moment.

Parameters

- **count** (*array*) – array with counts per bin
- **bins** (*array*) – array containing bins

Returns

2-element tuple, containing

- `float`: expectation value mu
- `float`: corresponding standard deviation

Return type tuple

fit_length_histo(*count*, *bins*, *noise_amp=None*, *max_num_gaussians=4*, ***kwargs*)
Apply multi gauss fit to length distribution histogram.

Parameters

- **count** (*array*) – histogram counts (see *flow_orientation_histo()*)
- **bins** (*array*) – histogram bins (see *flow_orientation_histo()*)
- **noise_amp** (`float`, optional) – minimum amplitude required for peaks in histogram in order to be considered for multi gauss fit, if None (default) use 5% of max count
- **max_num_gaussians** (`int`, optional) – maximum number of Gaussians fitted to the distributions, if None (default) then use `self.settings.hist_dir_gnum_max`

Returns

2-element tuple containing

- `MultiGaussFit`, the fit object
- bool, fit success

Return type tuple

get_main_flow_field_params(***kwargs*)
Old name of *local_flow_params()*.

local_flow_params(*line=None*, *pix_mask=None*, *noise_amp=None*, *min_count_frac=None*, *min_length=None*, *dir_multi_gauss=True*)
Histogram based statistical analysis of flow field in current ROI.

This function analyses histograms of the current flow field within a ROI in order to find the predominant movement direction (within the ROI) and the corresponding predominant displacement length.

Parameters

- **line** (`LineOnImage`, optional) – if provided, then the ROI corresponding to the line orientation is used (see `get_rotated_roi_mask()` in `LineOnImage` objects). If unspecified the current roi (*roi*) is used.
- **pix_mask** (`array`, optional) – boolean mask specifying image pixels supposed to be considered for the analysis, e.g. only plume pixels (determined applying a tau threshold to a tau image).
- **noise_amp** (`float`, optional) – this number specifies the minimum amplitude for individual peaks in the histograms (for multiple Gaussian fit). If unspecified here it will be set automatically in the corresponding methods *fit_length_histo()* and *fit_orientation_histo()*.
- **min_count_frac** (`float`, optional) – determines the minimum required number of significant vectors in current ROI for histogram analysis (i.e. if ROI is NxM pixels and `min_count_frac=0.1`, then at least (MxN)*0.1 pixels need to remain after applying `cond_mask_flat` and exclusion of vectors shorter than current minimum length `self.settings.min_length`)
- **min_length** (`float`, optional) – minimum length of vectors required in order to be considered for historgram analysis

- **dir_multi_gauss** (*bool*) – if True, a multi Gauss analysis (see `MultiGaussFit`) is applied to orientation histogram to separate the main peak from potential other peaks. Note that the optimisation slows down the analysis a bit.

Returns dictionary containing results of the analysis

Return type dict

apply_median_filter(*shape=(3, 3)*)
Apply median filter to flow field, i.e. to both flow images individually.

dx, dy is stored in self.flow.

Parameters **shape (3,3)** (*tuple*) – size of the filter

replace_trash_vecs(*displ_vec=(0.0, 0.0)*, *min_len=1.0*, *dir_low=-180.0*, *dir_high=180.0*)
Replace all vectors that do not match certain constraints.

Returns a new *OptflowFarneback* object with all vectors in attr. *flow* replaced by provided displacement vector.

Parameters

- **displ_vec** (*iterable*) – 2-element vector (list, tuple, array) containing diplacement information `(dx, dy)` supposed to be used to replace vectors not matching provided constraints related to minimum length and expectation direction range
- **min_len** (*float*) – minimum required length of vectors to be considered reliable
- **dir_low** (*float*) – lower end of accepted displacement direction in order to be considered reliable
- **dir_high** (*float*) – upper end of accepted displacement direction in order to be considered reliable

Returns duplicate of this class with *flow* containing `displ_vec` at indices not matching constraints

Return type *OptflowFarneback*

get_img_acq_times()
Return acquisition times of current input images.

Returns

2-element tuple, containing

- `datetime`: acquisition time of first image
- `datetime`: acquisition time of next image

Return type tuple

plot_orientation_histo(*pix_mask=None*, *min_length=None*, *bin_res_degrees=None*, *apply_fit=True*, *ax=None*, *tit='Orientation histo'*, *color='b'*, *label='Histo data'*, *bar_plot=True*, ***fit_settings*)
Plot flow orientation histogram.

Plots a histogram of the orientation angles of the flow vectors w within a certain ROI. By default, vectors shorter then `self.settings.min_length` are excluded from the histogram, if you want a histogram including the short vectors, provide input parameter `min_length=0.0`.

Todo: Finish docs . . .

---

plot_length_histo (*pix_mask=None, dir_low=-180, dir_high=180, min_length=None, bin_res_pix=1, apply_fit=False, apply_stats=True, ax=None, tit='Length histo', label='Histo', color='b', bar_plot=True, **fit_settings*)

Plot flow vector length histogram including some options.

Todo: Write docs . . .

---

plot_flow_histograms (*line=None, pix_mask=None, dir_multi_gauss=True*)

Plot detailed information about optical flow histograms.

Parameters

- **line** (`LineOnImage`, optional) – retrieval line used to calculate historgrams only in line specific ROI
- **pix_mask** (`ndarray`, optional) – 2D numpy array specifying pixels for histogram retrieval, if unspecified, all image pixels are used, if specified and **:param:'line'** is specified too, then the union of valid pixels between both parameters is used
- **dir_multi_gauss** (*bool*) – if True, then the orientation direction histogram is fitted using MultiGauss regression

Returns

Return type figure

calc_flow_lines (*in_roi=True, roi=None, extend_len_fac=1.0, include_short_vecs=False*)

Determine line objects for visualisation of current flow field.

Parameters

- **in_roi** (*bool*) – if True (default), then the lines are calculated for pixels within ROI (either specified by 2. input param and else `roi_abs` is used).
- **roi** (*list*) – Region of interest supposed to be displayed
- **extend_len_fac** (*float*) – factor by which length of vectors are extended
- **include_short_vecs** (*bool*) – if True, lines for short vectors are calculated as well

Returns the line coordinates

Return type tuple

plot (***kwargs*)

Draw current flow field onto image.

Wrapper for `draw_flow()`

Parameters ****kwargs** – key word args (see `draw_flow()`)

draw_flow (*in_roi=False, roi_abs=None, add_cbar=False, include_short_vecs=False, extend_len_fac=1.0, linewidth=1, color=None, ax=None*)

Draw the current optical flow field.

Parameters

- **in_roi** (*bool*) – if True, the flow field is plotted in a cropped image area else, the whole image is drawn
- **roi_abs** (`list`, optional) – region of interest for which the flow field is drawn (in absolute image coordinates, i.e. is converted to current pyrlevel). If None, then the `roi_abs` is used.

- **add_cbar** (*bool*) – if True, a colorbar is added to the plot (note that the images are converted into 8 bit before the flow is calculated, therefore the intensity range of the displayed image is between 0 and 256).
- **include_short_vecs** (*bool*) – if True, also vectors shorter than `self.settings.min_length` are drawn
- **extend_len_fac** (*float*) – factor by which length of vectors are extended
- **linewidth** (*int*) – with of flow vector lines
- **color** – Color of vectors if flow field is plotted onto an already plotted image.
- **ax** (*Axes*) – matplotlib axes object

Returns the plot axes

Return type Axes

live_example()
Show live example using webcam.

connect_histo(*canvasWidget*)

`pyplis.plumespeed.`**find_movement**(*first_img*, *next_img*, *pyrlevel=2*, *num_contrast_ivals=8*, *ival_overlap=0.05*, *imin=None*, *imax=None*, *apply_erosion=True*, *erosion_kernel_size=20*, *apply_dilation=True*, *dilation_kernel_size=20*, *verbose=False*, ***optflow_settings*)
Search for movement using an iterative optical flow algorithm.

This algorithm searches for pixels containing movement between two consecutive images. This is done by using an optical flow algorithm which computes the optical flow field between the two images for a series of different input contrast intervals (imin, imax).

Note: This is a Beta version

Parameters

- **first_img** (`Img`) – first image for computation of optical flow
- **next_img** (`Img`) – next image for computation of optical flow
- **pyrlevel** (*int*) – pyramid level for iterative analysis, default: 2
- **num_contrast_ivals** (*int*) – number of iterations (contrast intervals). The lower and upper values for each interval are determined based on the brightness range of the first image (or alternatively, the specified total considered brightness range using input parameters :param:ìmin‘ and **:param:‘imax‘**) divided by the number of specified intervals and the desired overlap between each interval (see **:param:‘ival_overlap‘**).
- **ival_overlap** (*float*) – percentage overlap between each of the intervals used to calculate the optical flow, default is 0.05 (corresponding to 5%)
- **imin** (`float`, optional) – lower limit for considered intensity range, if not specified, the minimum intensity of the first input image is used (at the specified pyramid level)
- **imax** (`float`, optional) – upper limit for considered intensity range, if not specified, the maximum intensity of the first input image is used (at the specified pyramid level)
- **apply_erosion** (*bool*) – if True, the OpenCV erosion algorithm is applied to the computed mask specifying pixels containing movement

- **erosion_kernel_size** (*int*) – size of the erosion kernel applied to movement mask if **:param:'apply_erosion'** is True. Note that the erosion is applied to the mask at the specified input pyramid level (if e.g. a size of 20 pixels is used at pyramid level 2, this corresponds to 80 pixels in the original image resolution)
- **apply_dilation** (*bool*) – if True, the OpenCV dilation algorithm is applied to the computed mask specifying pixels containing movement (this is done after the erosion is applied)
- **dilation_kernel_size** (*int*) – size of the dilation kernel applied to movement mask if **:param:'apply_dilation'** is True. Note that the dilation is applied to the mask at the specified input pyramid level (if e.g. a size of 20 pixels is used at pyramid level 2, this corresponds to 80 pixels in the original image resolution)
- **verbose** (*bool*) – print search information
- ****optflow_settings** – additional keyword arguments specifying settings for optical flow computation

Returns 2D-numpy boolean numpy array specifying pixels where movement was detected during the iterative search in the different brightness ranges. The pyramid level of the mask corresponds to the pyramid level of the input images and not the the pyramid level, where the computation was performed

Return type ndarray

class pyplis.plumespeed.**OpticalFlowFarnebackSettings**(**args, **kwargs*)

Old name of `FarnebackSettings`.

__init__(**args, **kwargs*)

x.__init__(...) initializes x; see help(type(x)) for signature

class pyplis.plumespeed.**OpticalFlowFarneback**(**args, **kwargs*)

Old name of `OptflowFarneback`.

__init__(**args, **kwargs*)

x.__init__(...) initializes x; see help(type(x)) for signature

5.8 Camera calibration base class

Pyplis module containing the `CalibData`.

This is the base class for storing calibration data, fitting calibration curves, and corresponding I/O routines (e.g storage as FITS or text file).

class pyplis.calib_base.**CalibData**(*tau_vec=None, cd_vec=None, cd_vec_err=None, time_stamps=None, calib_fun=None, calib_coeffs=None, senscorr_mask=None, polyorder=1, calib_id=", camera=None*)

Base class representing calibration data and optimisation parameters.

The default calibration curve is a polynomial of first order. Calibration data is represneted by two arrays `cd_vec` and `tau_vec` and optionally, a vector containing errors in the column densities `cd_vec_err` (note that errors in the optical densities are not supported). Furthermore, an array of `time_stamps` can be provided and If you want to use a custom calibration function you can provide the function using **:param:'calib_fun'**.

Parameters

- **tau_vec** (*ndarray*) – tau data vector for calibration data

- **cd_vec** (*ndarray*) – DOAS-CD data vector for calibration data
- **cd_vec_err** (*ndarray*) – Fit errors of DOAS-CDs
- **time_stamps** (*ndarray*) – array with datetime objects containing time stamps (e.g. start acquisition) of calibration data
- **calib_fun** (*function*) – optimisation function used for fitting of calibration data
- **calib_coeffs** (;obj:*list*, optional) – optimisation parameters for calibration curve.
- **senscorr_mask** (Img, optional) – sensitivity correction mask that was normalised relative to the pixel position where the calibration data was retrieved (i.e. position of DOAS FOV in case of DOAS calibration data, or image pixel position, where cell calibration data was retrieved)
- **calib_id** (*str*) – calibration ID (e.g. "aa", "tau_on", "tau_off")
- **camera** (Camera) – camera object (not necessarily required). A camera can be assigned in order to convert the FOV extend from pixel coordinates into decimal degrees

__init__(*tau_vec=None*, *cd_vec=None*, *cd_vec_err=None*, *time_stamps=None*, *calib_fun=None*, *calib_coeffs=None*, *senscorr_mask=None*, *polyorder=1*, *calib_id=''*, *camera=None*)
x.__init__(. . .) initializes x; see help(type(x)) for signature

num_optargs_fun(*fun*)
Return number of optimisation args of a function.

senscorr_mask
Get current sensitivity correction mask (Img instance).

calib_coeffs
List containing calibration coefficients for *calib_fun*.

calib_fun
Mathematical function used for retrieval of calibration curve.

Note: The function can be defined on class initiation and may be updated using the setter method. If not explicitely specified, a polynomial is used with order *polyorder*.

start
Start time of calibration data (datetime).

stop
Stop time of calibration data (datetime).

polyorder
Get current order of fit polynomial.

cov
Get covariance matrix of calibration polynomial.

y_offset
Y-axis offset of calib curve.

cd_tseries
Pandas Series object of doas data.

tau_tseries
Pandas Series object of tau data.

tau_range
Range of tau values extended by 5%.

Returns

2-element tuple, containing

- float, tau_min: lower end of tau range
- float, tau_max: upper end of tau range

Return type tuple

cd_range
Range of DOAS cd values extended by 5%.

has_calib_data()
Check if calibration data is available.

fit_calib_data(*calib_fun=None, guess=None, polyorder=None, weighted=True, weights_how='abs', through_origin=False, param_bounds=None, normalise_cds=False, plot=False*)
Fit calibration polynomial to current data.

The calibration data is fitted using a least squares optimisation. Be careful with cusomised optimisation functions that are not linear in all their optimisation parameters (especially, with using input argument **:arg:'normalise_cds'**).

Parameters

- **calib_fun** (`function`, optional) – if specified, the current calibration function is updated
- **guess** (`list`, optional) – initial guess for optimisation (is only considerd)
- **polyorder** (`int`, optional) – if specified, the current polyorder is updated (only relevant for polynomial optimisation functions, i.e. if no custom calibration function has been provided)
- **weighted** (*bool*) – performs weighted fit based on DOAS errors in `cd_vec_err` (if available), defaults to True
- **weights_how** (*str*) – use "rel" if relative errors are supposed to be used (i.e. w=CD_sigma / CD) or "abs" if absolute error is supposed to be used (i.e. w=CD_sigma).
- **through_origin** (*bool*) – only relevant for polynomial fits (i.e. if no custom fit function has been provided). If True, the polynomial fit is forced to cross the coordinate origin
- **param_bounds** (*tuple*) – 2-element tuple containing two lists (or tuples) specifying lower (param_borders[0]) and upper (param_borders[1]) borders for the fit parameters. If unspecified (None), the borders will automatically be set to -/+ infinity
- **normalise_cds** (*bool*) – if True, the CD vector is normalised by its exponential magnitude before applying the fit.
- **plot** (*bool*) – If True, the calibration curve and the polynomial are plotted

Returns list containing optimised parameters

Return type list

save_as_fits(*save_dir=None, save_name=None, overwrite_existing=True*)
Save calibration data as FITS file.

Save all relevant information in an HDU list as FITS. The first HDU (`PrimaryHDU`) contains the sensitivity correction mask (*senscorr_mask*) and the second HDU is of type `BinTableHDU` and contains the calibration data, which contains the following 4 columns in the specified order:

1. `time_stamps` (as strings, format: %Y%m%d%H%M%S%f)
2. `tau_vec`
3. `cd_vec`
4. `cd_vec_err`

Parameters

- **save_dir** (*str*) – save directory, if None, the current working directory is used
- **save_name** (*str*) – filename of the FITS file (if None, use pyplis default naming)
- **overwrite_existing** (*bool*) – if True, an existing calibration file with the same name will be overwritten

to_csv()

Store calibration data as tab delimited text file.

load_from_fits(*file_path*)

Load stack object (fits).

Parameters **file_path** (*str*) – file path of calibration data

Returns opened HDU object (e.g. to access potential further data in a function that is calling this method)

Return type HDUList

plot(*add_label_str=''*, *ax=None*, ***kwargs*)

Plot calibration data and fit result.

Parameters

- **add_label_str** (*str*) – additional string added to label of plots for legend
- **ax** – matplotlib axes object, if None, a new one is created

plot_calib_fun(*add_label_str=''*, *shift_yoffset=False*, *ax=None*, ***kwargs*)

Plot calibration fit result.

Parameters

- **add_label_str** (*str*) – additional string added to label of plots for legend
- **shift_yoffset** (*bool*) – if True, the data is plotted without y-offset
- **ax** – matplotlib axes object, if None, a new one is created

err()

Return standard deviation of fit residual.

calibrate(*value*)

Apply calibration to input.

Parameters **value** – optical density (can be float, n-dimensional numpy array, `Img`, `ImgStack`)

Returns

Return type calibrated input

5.9 Cell calibration

Pyplis module containing features related to cell calibration.

class pyplis.cellcalib.**CellSearchInfo**(*filter_id*, *add_id*, *y_max*)
Class for for storage cell search from automatic cell search engine.

Parameters

- **filter_id** (*str*) – string ID of filter / Image type
- **add_id** (*str*) – additional identifier (e.g. "bg", "cell1")
- **y_max** (*float*) – a reference intensity

__init__(*filter_id*, *add_id*, *y_max*)
x.__init__(. . .) initializes x; see help(type(x)) for signature

start
Get time stamp of first detected image.

Returns First time stamp, i.e. `start_acq[0]`

Return type datetime

Raises **IndexError** – if `start_acq` is empty

stop
Get time stamp of last detected image.

Returns Last time stamp, i.e. `self.start_acq[-1]`

Return type datetime

Raises **IndexError** – if `start_acq` is empty

tot_num
Get total number of detected images.

Returns length of `self.mean_vals`

Return type int

from_img_list(*img_list*)
Fill values using all images from a specific image list.

Note: Old beta version, not tested, currently not in use

Parameters **img_list** (ImgList) – image list from which pixel mean value time series is supposed to be determined

mean_err
Return average std of mean value time series.

Returns Error of mean value

Return type float

mid_point_val
Get mean value of middle image of the time series.

Returns Mean intensity of middle image

Return type float

point_ok(*idx*)
Check data point at given index.

Checks if intensity value at given index is within acceptance intensity range with respect to the middle intensity value of the time series

Parameters **idx** (*int*) – index of datapoint

Returns True, if ok, False if not

Return type bool

create_image_list(*camera*)
Create image list containing all valid cell images.

Creates a `CellImgList` which includes all detected data points that fulfill condition *point_ok()*.

Note: If successful, the list is assigned to `img_list`

Parameters **camera** (`Camera`) – the camera used

Returns image list containing all (valid) cell images

Return type *CellImgList*

offs
Get array containing offset values.

The offset values are determined from `mean_vals`` with respect to `y_max`.

Returns array containing offset values for each intensity in `mean_vals`

Return type ndarray

class pyplis.cellcalib.**CellAutoSearchResults**
Helper class collecting results from auto-cell detection algorithm.

This object is included in *CellCalibEngine* object and will be filled with *CellSearchInfo* objects if the cell autodetection is used (`find_cells()`)

Note: This class is normally not intended to be used directly

---

cell_info
Ordered dictionary containing dictionaries for all filter_ids for which cell search was performed (e.g. on, off). These dictionaries contain *CellSearchInfo* ordered based on the acq. time of the detected cell dip

Type OrderedDict

bg_info
Ordered dictionary containing *CellSearchInfo* objects that include detected background images for each filter (e.g. on off)

Type OrderedDict

rest_info
Ordered dictionary containing all images for each filter (e.g. on off) that could not be identified as Cell or BG image

Type OrderedDict

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

add_cell_search_result(*filter_id*, *cell_info*, *bg_info*, *rest_info*)

Add a collection of *CellSearchInfo* objects.

Parameters

- **filter_id** (*str*) – image type ID (e.g. on, off)
- **cell_info** (*dictlike*) – dictonary containing *CellSearchInfo* objects containing information about images belonging to one cell
- **bg_info** (CellSearchInfo) – object containing information about detected background images for image type specified by `filter_id`
- **rest_info** (CellSearchInfo) – object containing information about images that could not be assigned to a detected cell nor to the background images

class pyplis.cellcalib.**CellCalibData**(*tau_vec=None*, *cd_vec=None*, *cd_vec_err=None*, *time_stamps=None*, *calib_fun=None*, *calib_coeffs=None*, *senscorr_mask=None*, *polyorder=1*, *calib_id="*, *camera=None*, *pos_x_abs=None*, *pos_y_abs=None*)

Class representing cell calibration data.

This class inherits from the `CalibData` base class.

Parameters

- **tau_vec** (*ndarray*) – tau data vector for calibration data
- **cd_vec** (*ndarray*) – DOAS-CD data vector for calibration data
- **cd_vec_err** (*ndarray*) – Fit errors of DOAS-CDs
- **time_stamps** (*ndarray*) – array with datetime objects containing time stamps (e.g. start acquisition) of calibration data
- **calib_fun** (*function*) – optimisation function used for fitting of calibration data
- **calib_coeffs** (;obj:*list*, optional) – optimisation parameters for calibration curve.
- **senscorr_mask** (`ndarray`or :obj:`Img`, optional) – sensitivity correction mask that was normalised relative to the pixel position where the calibration data was retrieved (i.e. position of DOAS FOV in case of DOAS calibration data, or image pixel position, where cell calibration data was retrieved)
- **calib_id** (*str*) – calibration ID (e.g. "aa", "tau_on", "tau_off")
- **camera** (Camera) – camera object (not necessarily required). A camera can be assigned in order to convert the FOV extend from pixel coordinates into decimal degrees
- **pos_x_abs** (*int*) – x-position of image pixel for which the data was retrieved
- **pos_y_abs** (*int*) – y-position of image pixel for which the data was retrieved

__init__(*tau_vec=None*, *cd_vec=None*, *cd_vec_err=None*, *time_stamps=None*, *calib_fun=None*, *calib_coeffs=None*, *senscorr_mask=None*, *polyorder=1*, *calib_id="*, *camera=None*, *pos_x_abs=None*, *pos_y_abs=None*)

x.__init__(...) initializes x; see help(type(x)) for signature

load_from_fits (*file_path*)
Load calibration data from FITS file.

Note: See methods of base class `pyplis.calib_base.CalibData.save_as_fits()` and `pyplis.calib_base.CalibData.load_from_fits()` for more details

file_path [str] path of FITS file

class pyplis.cellcalib.**CellCalibEngine** (*setup=None*, *init=True*)
Class for performing automatic cell calibration.

This class is designed to define datasets related to time windows, where cell calibration was performed, i.e. the camera pointing into a gas (and cloud) free area of the sky with a number of calibration cells are put in front of the lense consecutively (ideally, the cells should cover the whole FOV of the camera in order to be able to retrieve calibration polynomials for each image pixel individually). Individual time windows for each cell are extracted by analysing the time series of pixel mean intensities for all images that fall into the start / stop interval. Cells can be identified by dips of decreased intensities in the time series. The individual cells are then assigned automatically based on the depth of each dip (in the on band) and the column densities of the cells used (the latter need to be provided).

Is initialised as `pyplis.Datasets.Dataset` object, i.e. normal setup is like plume data using a `MeasSetup` object (make sure that `cell_info_dict` is set in the setup class).

Parameters

- **setup** (`MeasSetup`) – see `Dataset` for details
- **init** (*bool*) – if True, the image lists are initiated and filled (if possible)

__init__ (*setup=None*, *init=True*)
x.__init__(. . .) initializes x; see help(type(x)) for signature

cell_lists_ready
Call `check_all_lists`()`.

set_cell_images (*img_paths*, *cell_gas_cd*, *cell_id*, *filter_id*, *img_import_method=None*)
Add cell images corresponding to one cell and image type.

Creates `CellImgList` containing input cell images and adds them calling `add_cell_img_list()`

Parameters

- **img_paths** (*list*) – list containing image file paths (can also be a single image path)
- **cell_gas_cd** (*float*) – column amount of gas in cell
- **cell_id** (*str*) – string identification of cell
- **filter_id** (*str*) – filter ID for images (e.g. "on", "off")

set_bg_images (*img_paths*, *filter_id*)
Set background images for a certain filter type.

Creates `CellImgList` containing input background images and adds them calling `add_bg_img_list()`

Parameters

- **img_paths** (*list*) – list containing image file paths of bg images (can also be a single image file path)
- **filter_id** (*str*) – image type (e.g. "on", "off")

add_cell_img_list (*lst*)
Add a cell image list for calibration.

Parameters lst (CellImgList) – if, valid, the list is added to cell_lists using it's ID (lst.list_id) as first key and lst.cell_id as second, e.g. self.cell_lists["on"]["a53"]

add_bg_img_list (*lst*)
Add an image list containing background images for calibration.

Parameters lst (CellImgList) – if valid input, the list is added to dictionary self.bg_lists using lst.list_id as key

det_bg_mean_pix_timeseries (*filter_id*)
Determine (or get) pixel mean values of background image list.

Gets the average pixel intenisty (considering the whole image) for all images in specified background image list and stores it within a PixelMeanTimeSeries object. The latter is then stored in bg_tseries and can be used to interpolate background intensities for cell image time stamps (this might be important for large SZA measurements where the background radiance changes fastly, cf. prepare_tau_calib()).

Parameters filter_id (*str*) – ID of background image list (must be valid key of dict bg_lists

Returns time series object containing background mean intensities

Return type *PixelMeanTimeSeries*

find_cells (*filter_id='on', threshold=0.1, accept_last_in_dip=False*)
Autodetection of cell images and bg images using mean value series.

This algorithm tries to separate individual cell images and background images by analysing the 1st derivative of the mean pixel intensity of each image in the time span specified in this object (self.start, self.stop).

The separation of the individual cell images is performed by identifying dips in the mean intensity evolution and assignment of all image files belonging to each dip.

Parameters

- **filter_id** (*str*) – filter ID (e.g. on, off, uses *get_list()* with filter_id as input)
- **threshold** (*float*) – threshold in percent by which intensity decreases are identified
- **accept_last_in_dip** (*bool*) – if True, also the last image in one of the Cell intensity dips is considered a valid cell image (by default, the first and the last images of a dip are not considered)

add_search_results ()
Add results from automatic cell detection to calibration.

This method analyses self.search_results for valid cell image lists (i.e. lists that contain images and have the gas column assigned)

set_bg_closest (*cell_id=None*)
Set the current background image closest to one of the cells.

Parameters cell_id (*str*) – cell ID supposed to be used, if None, then the first cell list in cell_lists is used

find_and_assign_cells_all_filter_lists (*threshold=0.1*)
High level function for automatic cell and BG image search.

This method basically calls the following functions:

1. `find_cells()` (for all filter IDs, e.g. on/off)
2. `_assign_calib_specs()`
3. `add_search_results()`
4. `check_all_lists()`

and sets flag `cell_search_performed=True`.

Parameters **threshold** (*float*) – percentage threshold for identification of regions of decreased intensity in time series

bg_img_available(*filter_id*)
Check if a background image is available.

Parameters **filter_id** (*str*) – filter ID of image list (e.g. on / off)

check_image_list(*lst*)
Check if image list contains files and has images ready (loaded).

Parameters **lst** (`ImgList`) – image list object

Raises

- **IndexError** – If list does not contain images
- **Exception** – If images cannot be loaded in list (unexpected error) or if `lst.gas_cd` is not a float

check_all_lists()
Check if all image lists are ready for analysis.

Returns True (if it makes it to the return statement)

Return type bool

check_cell_info_dict_autosearch(*cell_info_dict*)
Check if dict including cell gas column info is right format.

Parameters **cell_info_dict** (*dict*) – keys: cell ids (e.g. "a57"), values: list of gas column density and uncertainty in cm-2, format: `[value, error]`

Raises **Exception** – If any of the specs in `cell_info_dict` is invalid

set_cell_info_dict_autosearch(*cell_info_dict*)
Set attribute `self._cell_info_auto_search` (dictionary).

Parameters **cell_info_dict** (*dict*) – dictionary containing cell information

prep_tau_stacks(*on_id='on', off_id='off', darkcorr=True, blurring=2*)
Prepare image stacks for on, off and AA calibration data.

Parameters

- **on_id** (*str*) – ID of onband filter
- **off_id** (*str*) – ID of offband filter
- **darkcorr** (*bool*) – Use dark corrected images
- **blurring** (*int*) – sigma of Gaussian blurring kernel
- **pyrlevel** (*int*) – pyramid level of calibration stack

prepare_calib_data(*pos_x_abs=None*, *pos_y_abs=None*, *radius_abs=1*, *on_id='on'*, *off_id='off'*, *darkcorr=True*, *blurring=1*, ***kwargs*)

Prepare calib data for onband, offband and AA.

This function creates 3 `CellCalibData` objects for each OD type (on, off and from that, AA). If not differently specified using the input parameters `pos_x_abs` and `pos_y_abs` the corresponding cell optical densities are retrieved at the image center coordinate.

The 3 `CellCalibData` instances for each type (on, off, AA) can be accessed via the `calib_data` attribute of this class.

Parameters

- **pos_x_abs** (`int`, optional) – x-position for which the calibration data is retrieved
- **pos_y_abs** (`int`, optional) – y-position for which the calibration data is retrieved
- **radius_abs** (*int*) – radius specifying the disk size around `pos_x_abs` and `pos_y_abs` used to retrieve the cell-ODs
- **on_id** (*str*) – ID of onband filter used to determine calib curve
- **off_id** (*str*) – ID of offband filter used for calibration
- **darkcorr** (*bool*) – perform dark correction before determining cell tau images
- **blurring** (*int*) – apply gaussian blurring to cell tau images
- **pyrlevel** (*int*) – downscale factor (Gauss pyramid)

get_sensitivity_corr_mask(*calib_id='aa'*, *pos_x_abs=None*, *pos_y_abs=None*, *radius_abs=1*, *cell_cd_closest=0*, *surface_fit_pyrlevel=2*)

Get sensitivity correction mask.

Prepares a sensitivity correction mask to correct for filter transmission shifts. These shifts result in increasing optical densities towards the image edges for a given gas column density.

The mask is determined for original image resolution, i.e. pyramid level 0 and for a specific cell optical density image (aa, tau_on, tau_off). The latter is normalised with respect to the input pixel position (e.g. center position of DOAS FOV or pixel position where cell calibration data was retrieved).

Plume AA (or tau_on, tau_off) images can then be corrected for sensitivity variations by division with the mask. If DOAS calibration is used, the calibration function can then be used for all image pixels. If only cell calibration is used, the mask is normalised with respect to the image center, the corresponding cell calibration polynomial should then be retrieved in the center coordinate which is the default calibration position when using creating calibration data if not explicitely specified. You may then calibrate a given aa image (`aa_img`) as follows with using a `CellCalibData` object (denoted with `cellcalib`):

```
mask = cellcalib.get_sensitivity_corr_mask()
aa_corr = aa_img.duplicate()
aa_corr.img = aa_img.img / mask
#this is retrieved in the image center if not other specified
gas_cd_img = cellcalib(aa_corr)
gas_cd_img.show()
```

Parameters

- **calib_id** (*str*) – the mask is determined from the corresponding calib data (e.g. "on", "off", "aa")
- **pos_x_abs** (*int*) – x-pixel position of normalisation mask, if None the image center position is used (which is also the default pixel used to retrieve the vector of calibration optical densities from the cell OD images)

- **pos_y_abs** (*int*) – y-pixel position of normalisation mask, if None the image center position is used (which is also the default pixel used to retrieve the vector of calibration optical densities from the cell OD images)
- **radius_abs** (*int*) – radius specifying the disk size around `pos_x_abs` and `pos_y_abs` used to normalise the mask (i.e. uses average OD of cell image in this OD)
- **filter_id** (*str*) – mask is determined from the corresponding calib data (e.g. "on", "off", "aa")
- **cell_cd_closest** (*float*) – use the cell which is closest to the provided column density
- **surface_fit_pyrlevel** (*int*) – additional downscaling factor for 2D polynomial surface fit

Raises **ValueError** – if the corresponding `ImgStack` is cropped, from which the cell OD image is supposed to be retrieved

Returns the sensitivity correction mask

Return type *Img*

Note: This function was only tested for AA images and not for on / off cell tau images

get_list (*list_id*, *cell_id=None*)
Expand functionality of this method from `Dataset`.

Parameters

- **list_id** (*str*) – filter ID of list (e.g. on, off). If parameter `cell_id` is None, then this function returns the initial Dataset list (containing all images, not the ones separated by cells / background).
- **cell_id** (*str*) – if input is specified (type str) and valid (available cell img list), then the corresponding list is returned which only contains images from this cell. The string "bg" might be used to access the background image list of the filter specified with parameter `list_id`

Returns the actual list object

Return type *ImgList*

plot_cell_search_result (*filter_id='on'*, *for_app=False*, *include_tit=True*, *cell_cmap='Oranges'*, *ax=None*)
High level plotting function for results from auto-cell search.

Parameters

- **filter_id** (*str*) – filter ID (e.g. "on", "off")
- **for_app** (*bool*) – currently irrelevant (default is False)
- **include_tit** (*bool*) – if True, include default title
- **cell_cmap** (*str*) – string specifying matplotlib colormap used to plot cell time windows
- **ax** – matplotlib axes object

Returns matplotlib axes object

Return type axes

plot_calib_curve(*calib_id*, ***kwargs*)
Plot calibration curve.

Parameters

- **filter_id** (*str*) – image type ID (e.g. "aa")
- ****kwargs** – additional keyword arguments for plot passed to `plot()` of corresponding *CellCalibData* object

Returns matplotlib axes object

Return type axes

plot_all_calib_curves(*ax=None*, ***kwargs*)
Plot all available calibration curves in a certain pixel region.

Parameters

- **ax** (*axes*) – matplotlib axes instance
- ****kwargs** – additional keyword arguments passed to `get_calibration_polynomial()` of corresponding *CellCalibData* objects

Returns matplotlib axes object

Return type axes

5.10 DOAS calibration

Pyplis module for DOAS calibration including FOV search engines.

class pyplis.doascalib.**DoasCalibData**(*tau_vec=None*, *cd_vec=None*, *cd_vec_err=None*, *time_stamps=None*, *calib_fun=None*, *calib_coeffs=None*, *senscorr_mask=None*, *polyorder=1*, *calib_id=''*, *camera=None*, *fov=None*)
Class containing DOAS calibration data.

Parameters

- **tau_vec** (*ndarray*) – tau data vector for calibration data
- **cd_vec** (*ndarray*) – DOAS-CD data vector for calibration data
- **cd_vec_err** (*ndarray*) – Fit errors of DOAS-CDs
- **time_stamps** (*ndarray*) – array with datetime objects containing time stamps (e.g. start acquisition) of calibration data
- **calib_fun** (*function*) – optimisation function used for fitting of calibration data
- **calib_coeffs** (;obj:*list*, optional) – optimisation parameters for calibration curve.
- **senscorr_mask** (`ndarray`or :obj:`Img`, optional) – sensitivity correction mask that was normalised relative to the pixel position where the calibration data was retrieved (i.e. position of DOAS FOV in case of DOAS calibration data, or image pixel position, where cell calibration data was retrieved)
- **calib_id** (*str*) – calibration ID (e.g. "aa", "tau_on", "tau_off")

- **camera** (Camera) – camera object (not necessarily required). A camera can be assigned in order to convert the FOV extend from pixel coordinates into decimal degrees
- **fov** (DoasFOV) – information about position and shape of the FOV of the DOAS within the camera images

__init__(*tau_vec=None*, *cd_vec=None*, *cd_vec_err=None*, *time_stamps=None*, *calib_fun=None*, *calib_coeffs=None*, *senscorr_mask=None*, *polyorder=1*, *calib_id=”*, *camera=None*, *fov=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

save_as_fits(*save_dir=None*, *save_name=None*, *overwrite_existing=True*)
Save calibration data as FITS file.

Parameters

- **save_dir** (*str*) – save directory, if None, the current working directory is used
- **save_name** (*str*) – filename of the FITS file (if None, use pyplis default naming)

load_from_fits(*file_path*)
Load stack object (fits).

Parameters **file_path** (*str*) – file path of calibration data

plot_data_tseries_overlay(*date_fmt=None*, *ax=None*)
Plot overlay of tau and DOAS time series.

class pyplis.doascalib.**DoasFOV**(*camera=None*)
Class for storage of FOV information.

__init__(*camera=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

method
Return search method.

pyrlevel
Return pyramide level at which FOV search was performed.

cx_rel
Return center x coordinate of FOV (in relative coords).

cy_rel
Return center x coordinate of FOV (in relative coords).

radius_rel
Return radius of FOV (in relative coords).

Raises TypeError if method == “ifr”

popt
Return super gauss optimisation parameters (in relative coords).

Raises TypeError if method == “pearson”

x_abs

y_abs

sigma_x_abs

sigma_y_abs

pos_abs
Return center coordinates of FOV (in absolute detector coords).

pixel_extend(*abs_coords=True*)
Return pixel extend of FOV on image.

Parameters abs_coords (*bool*) – return value in absolute or relative coordinates (considering pyrlevel and roi)

pixel_position_center(*abs_coords=False*)
Return pixel position of center of FOV.

Parameters abs_coords (*bool*) – return position in absolute or relative coordinates (considering pyrlevel and roi)

Returns

- tuple, `(cx, cy)`

fov_mask_abs(*img_shape_orig=(), cam_id=''*)
Convert the FOV mask to absolute detector coordinates.

The shape of the FOV mask (and the represented pixel coordinates) depends on the image preparation settings of the `ImgStack` object which was used to identify the FOV.

Parameters

- **img_shape_orig** (*tuple*) – image shape of original image data (can be extracted from an unedited image)
- **cam_id** (*str*) – string ID of pyplis default camera (e.g. "ecII")

import_from_hdulist(*hdu, first_idx=0*)
Import FOV information from FITS HDU list.

Parameters

- **hdu** (*HDUList*) – HDU list containing a list of HDUs created using `prep_hdulist()` starting at index **:param:'first_idx'** (e.g. first_idx==2 if the method `save_as_fits()` from the `DoasCalibData` class is used, since the first 2 indices are used for saving the acutal calibration data)
- **first_idx** (*int*) – index specifying the first entry of the FOV info in the provided HDU list

prep_hdulist()
Prepare and return `HDUList` object for saving as FITS.

save_as_fits(***kwargs*)
Save the fov as fits file.

Saves this object as DoasCalibData:

```
d = DoasCalibData(fov = self)
d.save_as_fits(**kwargs)
```

plot(*ax=None*)
Draw the current FOV position into the current correlation img.

class pyplis.doascalib.**DoasFOVEngine**(*img_stack=None, doas_series=None, method='pearson', **settings*)
Engine to perform DOAS FOV search.

__init__(*img_stack=None, doas_series=None, method='pearson', **settings*)
x.__init__(...) initializes x; see help(type(x)) for signature

maxrad
maximum expected disk radius of FOV.

Note: this radius is considered independent of the current pyramid level of the image stack, hence, if it is set 20 and the pyramid level of the stack is 2, then, the FOV disk radius (in detector coords) may be 80.

Type For Pearson method

ifrlbda
allow asymmetric 2d gauss fit.

Type For IFR method

g2dasym
allow asymmetric 2d gauss fit.

Type For IFR method

g2dsuper
use supergauss parametrisation.

Type For IFR method

g2dcrop
crop gaussian FOV parametrisation at sigma.

Type For IFR method

g2dtilt
allow supergauss-fit to be tilted.

Type For IFR method

blur
Sigma of gaussian blurring filter applied to correlation image.

The filter is applied to the correlation image before finding the position of the maximum correlation. This is only relevant for method IFR, since this method parameterises the FOV by fitting a 2D Gaussian to the correlation image. Defaults to 4.

mergeopt
Option for temporal merging of stack and DOAS vector.

Choose from `average, nearest, interpolation`

update_search_settings(***settings*)
Update current search settings.

Parameters ****settings** – keyword args to be updated (only valid keys will be updated)

doas_data_vec
Return DOAS CD vector (values of `self.doas_series`).

method
Return current FOV search method.

perform_fov_search(***settings*)
High level method for automatic FOV search.

Uses the current settings (`self._settings`) to perform the following steps:

1. Call `merge_data()`: Time merging of stack and DOAS vector. This step is skipped if data was already merged within this engine, i.e. if `self.data_merged == True`

#. Call `det_correlation_image()`: Determination of correlation image using `self.method` ('ifr' or 'pearson')

#. Call `get_fov_shape()`: Identification of FOV shape / extend on image detector either using circular disk approach (if `self.method == 'pearson'`) or 2D (super) Gauss fit (if ``self.method == 'ifr').

All relevant results are written into `self.calib_data` (which includes *DoasFOV* object)

run_fov_fine_search (*img_list*, *doas_series*, *extend_fac=3*, ***settings*)
Get FOV position in full resolution.

Note: 1. Only works if FOV search (i.e. `perform_fov_search()`) was already performed. #. This method requires some time as it needs to recompute a cropped image stack in full resolution from the provided img_list. #. This method deletes the current image stack in this objects. #. Uses the same search settings as set in this class (i.e. method, etc.)

Parameters

- **img_list** (`BaseImgList`) – image list used to calculate cropped stack
- **doas_series** (*DoasResults*) – original DOAS time series (i.e. not merged in time with image data, needs to be provided since the one stored within this class is modified during the first FOV search)
- **extend_fac** (*int*) – factor determining crop ROI based on the current pixel extend of the FOV

Returns new instance containing results from fine search

Return type *DoasFOVEngine*

merge_data (*merge_type=None*)
Merge stack data and DOAS vector in time.

Wrapper for `merge_with_time_series()` of `ImgStack`

Parameters merge_type (*str*) – choose between ``average, interpolation,

nearest``

Note: Current data (i.e. `self.img_stack` and `self.doas_series`) will be overwritten if merging succeeds.

Parameters merge_type (*str*, optional,) – one of the available merge types, see *mergeopt* for valid options

Raises **RuntimeError** – if merging of data fails

det_correlation_image (*search_type='pearson'*, ***kwargs*)
Determine correlation image.

Determines correlation image either using IFR or Pearson method. Results are written into `self.calib_data.fov` (*DoasFOV*)

Parameters **search_type** (*str*) – updates current search type, available types `["pearson", "ifr"]`

get_fov_shape(***settings*)
Find shape of FOV based on correlation image.

Search pixel coordinate of highest correlation in `self.calib_data.fov.corr_img` (using `get_img_maximum()`) and based on that finds FOV shape either using disk approach (if `self.method == 'pearson'`) calling *fov_radius_search()* or using 2D Gauss fit (if `self.method == 'ifr'`) calling `fov_gauss_fit()`. Results are written into `self.calib_data.fov` (*DoasFOV* object)

Parameters ****settings** – update current settings (keyword args passed to *update_search_settings()*)

fov_radius_search(*cx*, *cy*)
Search the FOV disk radius around center coordinate.

The search varies the radius around the center coordinate and extracts image data time series from average values of all pixels falling into the current disk. These time series are correlated with spectrometer data to find the radius showing highest correlation.

Parameters

- **cx** (*int*) – pixel x coordinate of center position
- **cy** (*int*) – pixel y coordinate of center position

convolve_stack_fov(*fov_mask*)
Normalize fov image and convolve stack.

Returns

- stack time series vector within FOV

5.11 Emission rate retrieval

Pyplis module containing methods and classes for emission-rate retrievals.

class pyplis.fluxcalc.**OutputERA**(*out_dir=None*, *overlay_optflow=True*, *img_vmin=None*, *img_vmax=None*)
Class for specifying default output for emission rate analyses.

Note: This class is under development and not intended to be used currently

__init__(*out_dir=None*, *overlay_optflow=True*, *img_vmin=None*, *img_vmax=None*)
x.__init__(…) initializes x; see help(type(x)) for signature

init_output()
Create all relevant objects based on settings.

init_axes()
Init figure based on current settings.

class pyplis.fluxcalc.**EmissionRateSettings**(*pcs_lines=None*, *velo_glob=nan*, *velo_glob_err=nan*, *bg_roi_abs=None*, *ref_check_lower_lim=None*, *ref_check_upper_lim=None*, ***settings*)
Class for management of settings for emission rate retrievals.

Parameters

- **pcs_lines** – `LineOnImage` object or list containing `LineOnImage` objects along which emission rates are retrieved.
- **velo_glob** (*float*) – optional, global velocity estimate (e.g. retrieved from cross correlation analysis). Please note, that global velocities can also be assigned directly to `LineOnImage` objects (see prev. inp. param), hence, this input velocity is only used for lines, which do not have an explicit global velocity assigned. In any case, these velocities (whether assigned in `LineOnImage`objects or here) are only used if ``self.velo_mode["glob"] is True`.
- **velo_glob_err** (*float*) – optional, error on prev. parameter
- **bg_roi_abs** (*list*) – background region of interest used for logging of retrieved CDs in an area out of the plume (can later be used for an assessment of the performance of the plume background retrieval for each image) since the CDs are expected to be zero.
- **ref_check_lower_lim** (*float*) – lower required limit for CDs in `bg_roi_abs` area in case `ref_check_mode` is active. All images which show average CDs lower than this thresh within `bg_roi_abs` are disregarded for the analysis
- **ref_check_upper_lim** (*float*) – upper required limit for CDs in `bg_roi_abs` area in case `ref_check_mode` is active. All images which show average CDs larger than this thresh within `bg_roi_abs` are disregarded for the analysis

__init__(*pcs_lines=None*, *velo_glob=nan*, *velo_glob_err=nan*, *bg_roi_abs=None*, *ref_check_lower_lim=None*, *ref_check_upper_lim=None*, ***settings*)
x.__init__(…) initializes x; see help(type(x)) for signature

bg_roi_abs
Return current background reference ROI.

ref_check_mode
Activate / deactivate reference area control mode.

velo_mode_glob
Attribute velo_glob for velocity analysis retrieval.

velo_mode_flow_raw
Attribute velo_glob for velocity analysis retrieval.

velo_mode_flow_histo
Attribute for velocity analysis retrieval.

velo_mode_flow_hybrid
Attribute for velocity analysis retrieval.

velo_glob
Global velocity in m/s, assigned to this line.

Raises **AttributeError** – if current value is not of type float

velo_glob_err
Error of global velocity in m/s, assigned to this line.

add_pcs_line(*line*)
Add one analysis line to this list.

Parameters **line** (`LineOnImage`) – emission rate retrieval line

class pyplis.fluxcalc.**EmissionRates**(*pcs_id*, *velo_mode='glob'*, *settings=None*, *color='b'*)
Class to store results from emission rate analysis.

__init__(*pcs_id*, *velo_mode='glob'*, *settings=None*, *color='b'*)
x.__init__(...) initializes x; see help(type(x)) for signature

start
Acquisistion time of first image.

stop
Start acqusition time of last image.

start_acq
Array containing acquisition time stamps.

phi
Array containing emission rates.

phi_err
Array containing emission rate errors.

velo_eff
Array containing effective plume velocities.

velo_eff_err
Array containing effective plume velocitie errors.

as_series
Emission rates as pandas Series.

meta_header
Return string containing available meta information.

Returns string containing relevant meta information (e.g. for txt export)

Return type str

default_save_name
Return default name for txt export.

mean()
Mean of emission rate time series.

nanmean()
Mean of emission rate time series excluding NaNs.

std()
Mean of emission rate time series.

nanstd()
Mean of emission rate time series excluding NaNs.

min()
Minimum value of emission rate time series.

nanmin()
Minimum value of emission rate time series excluding NaNs.

max()
Maximum value of emission rate time series.

nanmax()
Maximum value of emission rate time series excluding NaNs.

get_date_time_strings()
Return string reprentations of date and start / stop times.

Returns

3-element tuple containing

- date string
- start acq. time string
- stop acq. time string

Return type tuple

to_dict()

Write all data attributes into dictionary.

Keys of the dictionary are the private class names

Returns Dictionary containing results

Return type dict

to_pandas_dataframe()

Convert object into pandas dataframe.

This can, for instance be used to store the data as csv (cf. *from_pandas_dataframe()*)

from_pandas_dataframe(*df*)

Import results from pandas `DataFrame` object.

Parameters **df** (*DataFrame*) – pandas dataframe containing emisison rate results

Returns this object

Return type *EmissionRates*

plot_velo_eff(*yerr=True*, *label=None*, *ax=None*, *date_fmt=None*, ***kwargs*)

Plot emission rate time series.

Parameters

- **yerr** (*bool*) – Include uncertainties
- **label** (*str*) – optional, string argument specifying label
- **ax** – optional, matplotlib axes object
- **date_fmt** (*str*) – optional, x label datetime formatting string, passed to `DateFormatter` (e.g. "%H:%M")
- ****kwargs** – additional keyword args passed to plot function of `Series` object

Returns matplotlib axes object

Return type axes

plot(*yerr=True*, *label=None*, *ax=None*, *date_fmt=None*, *ymin=None*, *ymax=None*, *alpha_err=0.1*, *in_kg=True*, ***kwargs*)

Plot emission rate time series.

Parameters

- **yerr** (*bool*) – Include uncertainties
- **label** (*str*) – optional, string argument specifying label
- **ax** – optional, matplotlib axes object
- **date_fmt** (*str*) – optional, x label datetime formatting string, passed to `DateFormatter` (e.g. "%H:%M")

- **ymin** (float, optional) – lower limit of y-axis
- **ymax** (float, optional) – upper limit of y-axis
- **alpha_err** (*float*) – transparency of uncertainty range
- **in_kg** (*bool*) – if True, emission rates are plotted in units of kg / s
- ****kwargs** – additional keyword args passed to plot call

Returns matplotlib axes object

Return type axes

save_txt (*path=None*)
Save this object as text file.

load_txt (*path*)
Load results from text file.

Parameters **path** (*str*) – valid file location

Returns loaded result data class

Return type *EmissionRates*

class pyplis.fluxcalc.**EmissionRateRatio** (**args*, ***kwargs*)
Time series ratio of two emission rates.

This class is new and still in Beta status

__init__ (**args*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

dphi
Return attr. phi, as this class represents ratios.

dphi_err
Return for attr. phi_err, as this class represents ratios.

plot (*yerr=False*, *label=None*, *ax=None*, *date_fmt=None*, *ymin=None*, *ymax=None*, *alpha_err=0.1*, ***kwargs*)
Plot emission rate time series.

Parameters

- **yerr** (*bool*) – Include uncertainties
- **label** (*str*) – optional, string argument specifying label
- **ax** – optional, matplotlib axes object
- **date_fmt** (*str*) – optional, x label datetime formatting string, passed to DateFormatter (e.g. "%H:%M")
- **ymin** (float, optional) – lower limit of y-axis
- **ymax** (float, optional) – upper limit of y-axis
- **alpha_err** (*float*) – transparency of uncertainty range
- **in_kg** (*bool*) – if True, emission rates are plotted in units of kg / s
- ****kwargs** – additional keyword args passed to plot call

Returns matplotlib axes object

Return type axes

class pyplis.fluxcalc.**EmissionRateAnalysis**(*imglist*, ***settings*)

Class to perform emission rate analysis.

The analysis is performed by looping over images in an image list which is in `calib_mode`, i.e. which loads images as gas CD images. Emission rates can be retrieved for an arbitrary amount of plume cross sections (defined by a list of `LineOnImage` objects which can be provided on init or added later). The image list needs to include a valid measurement geometry (`MeasGeometry`) object which is used to determine pixel to pixel distances (on a pixel column basis) and corresponding uncertainties.

Parameters

- **imglist** (ImgList) – onband image list prepared such, that at least `aa_mode` and `calib_mode` can be activated. If emission rate retrieval is supposed to be performed using optical flow, then also `optflow_mode` needs to work. Apart from setting these modes, no further changes are applied to the list (e.g. dark correction, blurring or choosing the pyramid level) and should therefore be set before. A warning is given, in case dark correction is not activated.
- **pcs_lines** (*list*) – python list containing `LineOnImage` objects supposed to be used for retrieval of emission rates (can also be a `LineOnImage` object directly)
- **velo_glob** (*float*) – global plume velocity in m/s (e.g. retrieved using cross correlation algorithm)
- **velo_glob_err** (*float*) – uncertainty in global plume speed estimate
- **bg_roi** (*list*) – region of interest specifying gas free area in the images. It is used to extract mean, max, min values from each of the calibrated images during the analysis as a quality check for the performance of the plume background retrieval or to detect disturbances in this region (e.g. due to clouds). If unspecified, the `scale_rect` of the plume background modelling class is used (i.e. `self.imglist.bg_model.scale_rect`).
- ****settings** – analysis settings (passed to *EmissionRateSettings*)

Todo: 1. Include light dilution correction - automatic correction for light dilution is currently not supported in this object. If you wish to perform light dilution, for now, please calculate dilution corrected on and offband images first (see example script ex11) and save them locally. The new set of images can then be used normally for the analysis by creating a `Dataset` object and an AA image list from that (see example scripts 1 and 4).

__init__(*imglist*, ***settings*)

x.__init__(. . .) initializes x; see help(type(x)) for signature

imglist_optflow

Image list supposed to be used for optical flow retrieval.

Is required to have the same number of images than analysis list. If this list is not set explicitely, then the optical flow is calculated from the analysis list (default setting).

This feature was introduced, since it was empirically found, that images that are dilution corrected, often cause problems with the optical flow retrieval, due to the applied threshold

pcs_lines

Return dict containing PCS retrieval lines assigned to settings class.

velo_glob

Global velocity.

velo_glob_err

Return error of current global velocity.

flow_required
Check if current velocity mode settings require flow algo.

get_results(*line_id=None*, *velo_mode=None*)
Return emission rate results (if available).

Parameters

- **line_id** (*str*) – ID of PCS line
- **velo_mode** (*str*) – velocity retrieval mode (see also *EmissionRateSettings*)

Returns

- EmissionRateResults, class containing emission rate

results for specified line and velocity retrieval

Raises

- KeyError, if result for the input constellation cannot be

found

check_and_init_list()
Check if image list is ready and include all relevant info.

get_pix_dist_info_all_lines()
Retrieve pixel distances and uncertainty for all pcs lines.

Returns

2-element tuple containing

- `dict`, keys are line ids, vals are arrays with pixel dists
- `dict`, keys are line ids, vals are distance uncertainties

Return type tuple

init_results()
Reset results.

Returns

2-element tuple containing

- `dict`, keys are line ids, vals are empty result classes
- **dict, keys are line ids, vals are empty** `LocalPlumeProperties` objects

Return type tuple

check_pcs_plume_props()
Check if plume displacement information is available for all PCS.

Tries to access `LocalPlumeProperties` objects in each of the assigned plume cross section retrieval lines (*pcs_lines*). If so and if a considerable datetime index overlap is given in the corresponding object (with datetime indices in `imglist`), then the object is interpolated onto the time stamps of the list and the corresponding displacement information is used (and not re-calculated) while performing emission rate retrieval when using `velo_mode = flow_histo`. If no significant overlap can be detected, the `LocalPlumeProperties` object in the corresponding `LineOnImage` object is initiated and filled while performing the analysis.

calc_emission_rate(***kwargs*)
Old name of *run_retrieval()*.

run_retrieval (*start_index=0*, *stop_index=None*, *check_list=True*)
Calculate emission rates of image list.

Performs emission rate analysis for each line in `self.pcs_lines` and for all plume velocity retrievals activated in `self.settings.velo_modes`. The results for each line and velocity mode are stored within *EmissionRates* objects which are saved in `self.results[line_id][velo_mode]`, e.g.:

```
res = self.results["bla"]["flow_histo"]
```

would yield emission rate results for line with ID "bla" using histogram based plume speed analysis.

The results can also be easily accessed using *get_results()*.

Parameters

- **start_index** (*int*) – index of first considered image in `self.imglist`, defaults to 0
- **stop_index** (*int*) – index of last considered image in `self.imglist`, defaults to last image in list
- **check_list** (*bool*) – if True, *check_and_init_list()* is called before analysis

Returns

2-element tuple containing

- `dict`, keys are line ids, vals are corresponding results
- **dict, keys are line ids, vals are** `LocalPlumeProperties` objects

Return type tuple

add_pcs_line (*line*)
Add one analysis line to this list.

Parameters **line** (`LineOnImage`) – the line object

plot_pcs_lines (*ax=None*, ***kwargs*)
Plot all current PCS lines onto current list image.

plot_bg_roi_rect (*ax=None*, *to_pyrlevel=0*)
Plot rectangular area used for background check.

plot_bg_roi_vals (*ax=None*, *date_fmt=None*, *labelsize=None*, ***kwargs*)
Plot emission rate time series.

Parameters

- **ax** – optional, matplotlib axes object
- **date_fmt** (*str*) – optional, x label datetime formatting string, passed to `DateFormatter` (e.g. "%H:%M")
- ****kwargs** – additional keyword args passed to plot function of `Series` object

Returns ax, matplotlib axes object

Return type axes

`pyplis.fluxcalc.`**det_emission_rate** (*cds*, *velo*, *pix_dists*, *cds_err=None*, *velo_err=None*, *pix_dists_err=None*, *mmol=64.0638*)
Determine emission rate.

Parameters

- **cds** – column density in units cm-2 (float or ndarray)
- **velo** – effective plume velocity in units of m/s (float or ndarray) Effective means, that it is with respect to the direction of the normal vector of the plume cross section used (e.g. by performing a scalar product of 2D velocity vectors with normal vector of the PCS)
- **pix_dists** – pixel to pixel distances in units of m (float or ndarray)

class pyplis.fluxcalc.**EmissionRateResults**(**args*, ***kwargs*)
Old name of OptflowFarneback.

__init__(**args*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

5.12 Signal dilution correction

Pyplis module for image based correction of the signal dilution effect.

class pyplis.dilutioncorr.**DilutionCorr**(*lines=None*, *meas_geometry=None*, ***settings*)
Class for management of dilution correction.

The class provides functionality to retrieve topographic distances from meas geometry, to manage lines in the image used for the retrieval, to perform the actual dilution fit (i.e. retrieval of atmospheric scattering coefficients) and to apply the dilution correction.

This class does not store any results related to individual images.

Parameters

- **lines** (*list*) – optional, list containing LineOnImage objects used to retrieve terrain distances for the dilution fit
- **meas_geometry** (MeasGeometry) – optional, measurement geometry (required for terrain distance retrieval)
- ****settings** – settings for terrain distance retrieval:
  - **skip_pix: specify pixel step on line for which topo** intersections are searched
  - **min_slope_angle: minimum slope of topography in order to be** considered for topo distance retrieval
  - **topo_res_m: interpolation resolution applied to** ElevationProfile objects used to find intersections of pixel viewing direction with topography

__init__(*lines=None*, *meas_geometry=None*, ***settings*)
x.__init__(...) initializes x; see help(type(x)) for signature

line_ids
Get IDs of all LineOnImage objects for distance retrieval.

update_settings(***settings*)
Update settings dict for topo distance retrieval.

add_retrieval_line(*line*)
Add one topography retrieval line.

add_retrieval_point(*pos_x_abs*, *pos_y_abs*, *dist=None*)
Add a distinct pixel with known distance to image.

Parameters

- **pos_x_abs** (*int*) – x-pixel position of point in image in absolute coordinate (i.e. pyramid level 0 and not cropped)
- **pos_y_abs** (*int*) – y-pixel position of point in image in absolute coordinate (i.e. pyramid level 0 and not cropped)
- **dist** (`float`, optional) – distance to feature in image in m. If None (default), the distance will be estimated

det_topo_dists_all_lines (***settings*)
Estimate distances to topo distances to all assigned lines.

Parameters ****settings** – keyword args passed to update search settings (`settings`) and passed to `get_topo_distances_line()` in `MeasGeometry`

det_topo_dists_line (*line_id*, ***settings*)
Estimate distances to pixels on current lines.

Retrieves distances to all `LineOnImage` objects in `self.lines` using `self.meas_geometry` (i.e. camera position and viewing direction).

Parameters

- **line_id** (*str*) – ID of line
- ****settings** – additional key word args used to update search settings (passed to `get_topo_distances_line()` in `MeasGeometry`)

Returns retrieved distances

Return type array

get_radiances (*img*, *line_ids=None*)
Get radiances for dilution fit along terrain lines.

The data is only extracted along specified input lines. The terrain distance retrieval `det_topo_dists_lines_line()` must have been performed for that.

Parameters

- **img** (`Img`) – vignetting corrected plume image from which the radiances are extracted
- **line_ids** (*list*) – if desired, the data can also be accessed for specified line ids, which have to be provided in a list. If empty (default), all lines assigned to this class are considered

apply_dilution_fit (*img*, *rad_ambient*, *i0_guess=None*, *i0_min=0*, *i0_max=None*, *ext_guess=0.0001*, *ext_min=0*, *ext_max=0.001*, *line_ids=None*, *plot=True*, ***kwargs*)
Perform dilution correction fit to retrieve extinction coefficient.

Uses `dilution_corr_fit()` of `optimisation` which is a bounded least square fit based on the following model function

$$I_{meas}(\lambda) = I_0(\lambda)e^{-\epsilon(\lambda)d} + I_A(\lambda)(1 - e^{-\epsilon(\lambda)d})$$

Parameters

- **img** (`Img`) – vignetting corrected image for radiance extraction
- **rad_ambient** (*float*) – ambient intensity (I_A in model)
- **i0_guess** (*float*) – optional: guess value for initial intensity of topographic features, i.e. the reflected radiation before entering scattering medium (I_0 in model, if None, then it is set 5% of the ambient intensity `rad_ambient`)

- **i0_min** (*float*) – optional: minimum initial intensity of topographic features
- **i0_max** (*float*) – optional: maximum initial intensity of topographic features
- **ext_guess** (*float*) – guess value for atm. extinction coefficient (ϵ in model)
- **ext_min** (*float*) – minimum value for atm. extinction coefficient
- **ext_max** (*float*) – maximum value for atm. extinction coefficient
- **line_ids** (*list*) – if desired, the data can also be accessed for specified line ids, which have to be provided in a list. If empty (default), all lines are considered
- **plot** (*bool*) – if True, the result is plotted
- ****kwargs** – additional keyword args passed to plotting function (e.g. to pass an axes object)

Returns

4-element tuple containing

- retrieved extinction coefficient
- retrieved initial intensity
- fit result object
- axes instance or None (dependent on **:param:‘plot‘**)

Return type tuple

get_ext_coeffs_imglist (*lst*, *roi_ambient=None*, *apply_median=5*, ***kwargs*)
Apply dilution fit to all images in an `ImgList`.

Parameters

- **lst** (`ImgList`) – image list for which the coefficients are supposed to be retrieved
- **roi_ambient** (*list*) – region of interest used to estimage ambient intensity, if None (default), usd `scale_rect` of `PlumeBackgroundModel` of the input list
- **apply_median** (*int*) – if > 0, then a median filter of provided width is applied to the result time series (ext. coeffs and initial intensities)
- ****kwargs** – additional keyword args passed to dilution fit method `apply_dilution_fit()`.

Returns

pandas data frame containing time series of retrieved extinction coefficients and initial intensities as well as the ambient intensities used, access keys are:

- `coeffs`: retrieved extinction coefficients
- `i0`: retrieved initial intensities
- `ia`: retrieved ambient intensities

Return type DataFrame

correct_img (*plume_img*, *ext*, *plume_bg_img*, *plume_dists*, *plume_pix_mask*)
Perform dilution correction for a plume image.

Note: See `correct_img()` for description

Returns dilution corrected image

Return type *Img*

plot_fit_result (*dists*, *rads*, *rad_ambient*, *i0*, *ext*, *ax=None*)
Plot result of dilution fit.

get_extinction_coeffs_imglist (*imglist*, *ambient_roi_abs*, *darkcorr=True*, *line_ids=None*, ***fit_settings*)
Retrieve extinction coefficients for all imags in list.

Note: Alpha version: not yet tested

plot_distances_3d (*draw_cam=1*, *draw_source=1*, *draw_plume=0*, *draw_fov=0*, *cmap_topo='Oranges'*, *contour_color='#708090'*, *contour_antialiased=True*, *contour_lw=0.2*, *axis_off=True*, *line_ids=None*, ***kwargs*)
Draw 3D map of scene including geopoints of distance retrievals.

Parameters

- **draw_cam** (*bool*) – insert camera position into map
- **draw_source** (*bool*) – insert source position into map
- **draw_plume** (*bool*) – insert plume vector into map
- **draw_fov** (*bool*) – insert camera FOV (az range) into map
- **cmap_topo** (*str*) – string specifying colormap for topography surface plot defaults to "Oranges"
- **contour_color** (*str*) – string specifying color of contour lines colors of topo contour lines (default: "#708090")
- **contour_antialiased** (*bool*) – apply antialiasing to surface plot of topography, defaults to False
- **contour_lw** – width of drawn contour lines, defaults to 0.5, use 0 if you do not want contour lines inserted
- **axis_off** (*bool*) – if True, then the rendering of axes is excluded
- **line_ids** (*list*) – if desired, the data can also be accessed for specified line ids, which have to be provided in a list. If empty (default), all topo lines are drawn

Returns plotted map instance (is of type Basemap)

Return type Map

pyplis.dilutioncorr.**correct_img** (*plume_img*, *ext*, *plume_bg_img*, *plume_dists*, *plume_pix_mask*)
Perform dilution correction for a plume image.

Corresponds to Eq. 4 in in Campion et al., 2015.

Parameters

- **plume_img** (Img) – vignetting corrected plume image
- **ext** (*float*) – atmospheric extinction coefficient
- **plume_bg_img** (Img) – vignetting corrected plume background image (can be, for instance, retrieved using `plumebackground`)

- **plume_dists** (*array*, Img, *float*) – plume distance(s) in m. If input is numpy array or Img then, it must have the same shape as **:param:'plume_img'**
- **plume_pix_mask** (*ndarray*) – mask specifying plume pixels (only those are corrected), can also be type Img

Returns dilution corrected image

Return type *Img*

pyplis.dilutioncorr.**get_topo_dists_lines**(*lines*, *geom*, *img=None*, *skip_pix=5*, *topo_res_m=5.0*, *min_slope_angle=5.0*, *plot=False*, *line_color='lime'*)

pyplis.dilutioncorr.**perform_dilution_correction**(*plume_img*, *ext*, *plume_bg_img*, *plume_dist_img*, *plume_pix_mask*)

pyplis.dilutioncorr.**get_extinction_coeff**(*rads*, *dists*, *rad_ambient*, *plot=True*, ***kwargs*)

Perform dilution correction fit to retrieve extinction coefficient.

Parameters

- **rads** (*ndarray*) – radiances retrieved for topographic features
- **dists** (*ndarray*) – distances corresponding to rads
- **rad_ambient** – ambient sky intensity
- **plot** (*bool*) – if True, the result is plotted
- ****kwargs** – additional keyword arguments for fit settings (passed to dilution_corr_fit() of module optimisation)

5.13 Low level utils

Pyplis module containing low level utilitiy methods and classes.

pyplis.utils.**identify_camera_from_filename**(*filepath*)

Identify camera based on image filepath convention.

Parameters **filepath** (*str*) – valid image file path

Returns ID of Camera that matches best

Return type str

Raises **IOError** – Exception is raised if no match can be found

class pyplis.utils.**LineOnImage**(*x0=0*, *y0=0*, *x1=1*, *y1=1*, *normal_orientation='right'*, *roi_abs_def=[0, 0, 9999, 9999]*, *pyrlevel_def=0*, *line_id="*, *color='lime'*, *linestyle='-'*)

Class representing a line on an image

Main purpose is data extraction along this line on a discrete image grid. This is done using spline interpolation.

Parameters

- **x0** (*int*) – start x coordinate
- **y0** (*int*) – start y coordinate
- **x1** (*int*) – stop x coordinate
- **y1** (*int*) – stop y coordinate

- **normal_orientation** (*str*) – orientation of normal vector, choose from left or right (left means in negative x direction for a vertical line)
- **roi_abs_def** (*list*) – ROI specifying image sub coordinate system in which the line coordinates are defined (is used to convert to other image shape settings)
- **pyrlevel_def** (*int*) – pyramid level of image for which start /stop coordinates are defined
- **line_id** (*str*) – string for identification (optional)

Note: The input coordinates correspond to relative image coordinates with respect to the input ROI (`roi_def`) and pyramid level (`pyrlevel_def`)

__init__ (*x0=0, y0=0, x1=1, y1=1, normal_orientation='right', roi_abs_def=[0, 0, 9999, 9999], pyrlevel_def=0, line_id='', color='lime', linestyle='-'*)
x.__init__(...) initializes x; see help(type(x)) for signature

start
x, y coordinates of start point (`[x0, y0]`).

stop
x, y coordinates of stop point (`[x1, y1]`).

center_pix
Return coordinate of center pixel.

normal_orientation
Get / set value for orientation of normal vector.

line_frame
ROI framing the line (in line coordinate system).

line_frame_abs
ROI framing the line (in absolute coordinate system).

roi_def
ROI in which line is defined (at current `pyrlevel`).

roi_abs_def
Return current ROI (in absolute detector coordinates).

pyrlevel
Pyramid level at which line coords are defined.

roi_abs
Return current ROI (in absolute detector coordinates).

pyrlevel_def
Pyramid level at which line coords are defined.

coords
Return coordinates as ROI list.

rect_roi_rot
Rectangle specifying coordinates of ROI aligned with line normal.

velo_glob
Global velocity in m/s, assigned to this line.

Raises `AttributeError` – if current value is not of type float

velo_glob_err
Error of global velocity in m/s, assigned to this line.

Raises **AttributeError** – if current value is not of type float

plume_props
`LocalPlumeProperties` object assigned to this list.

dist_other (*other*)
Determine the distance to another line.

Note: 1. The offset is applied in relative coordinates, i.e. it does not consider the pyramide level or ROI.

1. The two lines need to be parallel

Parameters **other** (`LineOnImage`) – the line to which the distance is retrieved

Returns retrieved distance in pixel coordinates

Return type float

Raises **ValueError** – if the two lines are not parallel

offset (*pixel_num=20, line_id=None*)
Return a new line shifted within normal direction.

Note:

1. **The offset is applied in relative coordinates, i.e. it does not** consider the pyramide level or ROI
2. **The determined required displacement (dx, dy) is converted into** integers

Parameters

- **pixel_num** (*int*) – shift length in pixels
- **line_id** (*str*) – string ID of new line, if None (default) it is set automatically

Returns shifted line

Return type *LineOnImage*

convert (*to_pyrlevel=0, to_roi_abs=[0, 0, 9999, 9999]*)
Convert to other image preparation settings.

check_coordinates ()
Check line coordinates.

Checks if coordinates are in the right order and exchanges start / stop points if not

Raises **ValueError** – if any of the current coordinates is smaller than zero

in_image (*img_array*)
Check if this line is within the coordinates of an image array.

Parameters **img_array** (*array*) – image data

Returns True if point is in image, False if not

Return type bool

point_in_image (*x*, *y*, *img_array*)
Check if a given coordinate is within image.

Parameters

- **x** (*int*) – x coordinate of point
- **y** (*int*) – y coordinate of point
- **img_array** (*array*) – image data

Returns True if point is in image, False if not

Return type bool

get_roi_abs_coords (*img_array*, *add_left=5*, *add_right=5*, *add_bottom=5*, *add_top=5*)
Get a rectangular ROI covering this line.

Parameters

- **add_left** (*int*) – expand range to left of line (in pix)
- **add_right** (*int*) – expand range to right of line (in pix)
- **add_bottom** (*int*) – expand range to bottom of line (in pix)
- **add_top** (*int*) – expand range to top of line (in pix)

Returns ROI around this line

Return type list

integrate_profile (*input_img*, *pix_step_length=None*)
Integrate the line profile on input image.

Parameters **input_img** (Img) – input image data for

set_rect_roi_rot (*depth=None*)
Get rectangle for rotated ROI based on current tilting.

Note: This function also changes the current `roi_abs` attribute

Parameters **depth** (*int*) – depth of rotated ROI (in normal direction of line) in pixels

Returns rectangle coordinates

Return type list

get_rotated_roi_mask (*shape*)
Return pixel access mask for rotated ROI.

Parameters **shape** (*tuple*) – shape of image for which the mask is supposed to be used

Returns bool array that can be used to access pixels within the ROI

Return type array

check_roi_borders (*roi*, *img_array*)
Check if all points of ROI are within image borders.

Parameters

- **roi** (*list*) – ROI rectangle `[x0,y0,x1,y1]`
- **img_array** (*array*) – exemplary image data for which the ROI is checked

Returns roi within image coordinates (unchanged, if input is ok, else image borders)

Return type list

prepare_coords()

Prepare the analysis mesh.

Note: The number of analysis points on this object correponds to the physical length of this line in pixel coordinates.

length()

Determine the length in pixel coordinates.

get_line_profile(*array*, *order=1*, ***kwargs*)

Retrieve the line profile along pixels in input array.

Parameters

- **array** (*array*) – 2D data array (e.g. image data). Color images are converted into gray scale using `cv2.cvtColor()`.
- **order** (*int*) – order of spline interpolation used to retrieve the values along input coordinates (passed to `map_coordinates()`)
- ****kwargs** – additional keword args passed to interpolation method `map_coordinates()`

Returns profile

Return type array

plot_line_on_grid(*img_arr=None*, *ax=None*, *include_normal=False*, *include_roi_rot=False*, *include_roi=False*, *annotate_normal=False*, ***kwargs*)

Draw this line on the image.

Parameters

- **img_arr** (*ndarray*) – if specified, the array is plotted using `imshow()` and onto that axes, the line is drawn
- **ax** – matplotlib axes object. Is created if unspecified. Leave **:param:'img_arr'** empty if you want the line to be drawn onto an already existing image (plotted in ax)
- **include_normal** (*bool*) – if True, the line normal vector is drawn
- **include_roi_rot** (*bool*) – if True, a line-orientation specific ROI is drawn
- **include_roi** (*bool*) – if True, an ROI is drawn which spans the i,j range of the image covered by the line
- **annotate_normal** (*bool*) – if True, the normal vector is annotated (only if include_normal is set True)
- ****kwargs** – additional keyword arguments for plotting of line (please use following keys: marker for marker style, mec for marker edge color, c for line color and ls for line style)

Returns matplotlib axes instance

Return type Axes

plot_rotated_roi(*color=None*, *ax=None*)

Plot current rotated ROI into axes.

Parameters

- **color** – optional, color information. If None (default) then the current line color is used
- **ax** (Axes, optional) – matplotlib axes object, if None, a figure with one subplot will be created

Returns axes instance

Return type Axes

plot_line_profile(*img_arr*, *ax=None*)
Plot the line profile.

plot(*img_arr*)
Create two subplots showing line on image and corresponding profile.

Parameters **img_arr** (*array*) – the image data

Returns figure containing the supblots

Return type Figure

norm
Return length of line in pixels.

det_normal_vecs()
Get both normal vectors.

normal_vector
Get normal vector corresponding to current orientation setting.

complex_normal
Return current normal vector as complex number.

normal_theta
Return orientation of normal vector in degrees.

The angles correspond to:

1. 0 => to the top (neg. y direction)
2. 90 => to the right (pos. x direction)
3. 180 => to the bottom (pos. y direction)
4. 270 => to the left (neg. x direction)

to_list()
Return line coordinates as 4-list.

to_dict()
Write relevant parameters to dictionary.

from_dict(*settings_dict*)
Load line parameters from dictionary.

Parameters **settings_dict** (*dict*) – dictionary containing line parameters (cf. *to_dict()*)

orientation_info
Return string about orientation of line and normal.

class pyplis.utils.**Filter**(*id=None*, *type='on'*, *acronym='default'*, *meas_type_acro=None*, *center_wavelength=nan*)
Object representing an interference filter.

A low level helper class to store information of interference filters.

__init__(*id=None*, *type='on'*, *acronym='default'*, *meas_type_acro=None*, *center_wavelength=nan*)
Initialize of object.

Parameters

- **id ("on")** (*str*) – string identification of this object for working environment
- **type ("on")** (*str*) – Type of object (choose from "on" and "off")
- **acronym ("")** (*str*) – acronym for identification in filename
- **meas_type_acro ("")** (*str*) – acronym for meastype identification in filename
- **center_wavelength (nan)** (*str*) – center transmission wavelength of filter

to_list()
Return filter info as list.

set_trans_curve(*data*, *wavelengths=None*)
Assign transmission curve to this filter.

Parameters

- **data** (*ndarray*) – transmission data
- **wavelengths** (*ndarray*) – corresponding wavelength array

Returns `pandas.Series` object

Note: Also accepts `pandas.Series` as input using input param data and leaving wavelength empty, in this case, the Series index is assumed to be the wavelenght data

print_specs()
Print __str__.

class pyplis.utils.**DarkOffsetInfo**(*id='dark'*, *type='dark'*, *acronym="*, *meas_type_acro=None*, *read_gain=0*)
Base class for storage of dark offset information.

Similar to *Filter*. This object can be used to store relevant information of different types of dark and offset images. The attribute "read_gain" is set 0 by default. For some camera types (e.g. Hamamatsu c8484 16c as used in the ECII SO2 camera), the signal can be enhancened with an electronic read_gain (measured in dB) on read. This can be helpful in low light conditions. However, it significantly increases the noise in the images and therefore also the dark image signal.

__init__(*id='dark'*, *type='dark'*, *acronym="*, *meas_type_acro=None*, *read_gain=0*)
Initialize object.

Parameters

- **id** (*str*) – string identification of this object for working environment (default: "dark")
- **type** (*str*) – Type of object (e.g. dark or offset, default: "dark")
- **acronym** (*str*) – acronym for identification in filename
- **meas_type_acro** (*str*) – acronym for meastype identification in filename
- **read_gain** (*str*) – string specifying read_gain mode of this object (use 0 or 1, default is 0)

to_list()
Return parameters as list.

5.14 Further processing classes

Pyplis module contains the following processing classes and methods.

1. *ImgStack*: Object for storage of 3D image data

#. *PixelMeanTimeSeries*: storage and post analysis of timeseries of average pixel intensities

class pyplis.processing.**ImgStack**(*height=0*, *width=0*, *img_num=0*, *dtype=<type 'numpy.float32'>*, *stack_id=''*, *img_prep=None*, *camera=None*, ***stack_data*)

Image stack object.

The images are stacked into a 3D numpy array, note, that for large datasets this may cause MemoryErrors. This object is for instance used to perform a DOAS field of view search (see also `doascalib`).

It provides basic image processing functionality, for instance changing the pyramid level, time merging with other time series data (e.g. DOAS CD time series vector).

The most important attributes (data objects) are:

1. `self.stack`: 3D numpy array containing stacked images. The first axis corresponds to the time axis, allowing for easy image access, e.g. `self.stack[10]` would yield the 11th image in the time series.

2. `self.start_acq`: 1D array containing acquisition time stamps (datetime objects)

3. `self.texps`: 1D array conaining exposure times in s for each image

4. `self.add_data`: 1D array which can be used to store additional data for each image (e.g. DOAS CD vector)

Todo:

1. Include optical flow routine for emission rate retrieval

Parameters

- **height** (*int*) – height of images to be stacked
- **width** (*int*) – width of images to be stacked
- **num** (*int*) – number of images to be stacked
- **dtype** –

  numerical data type (e.g. uint8, makes the necessary space smaller, default: float32)
- **stack_id** (*str*) – string ID of this object ("")
- **img_prep** (*dict*) – additional information about the preparation state of the images (e.g. roi, gauss pyramid level, dark corrected?, blurred?)
- ****stack_data** – can be used to pass stack data directly

__init__(*height=0*, *width=0*, *img_num=0*, *dtype=<type 'numpy.float32'>*, *stack_id=''*, *img_prep=None*, *camera=None*, ***stack_data*)

x.__init__(. . .) initializes x; see help(type(x)) for signature

init_stack_array(*height=0*, *width=0*, *img_num=0*)

Initialize the actual stack data array.

Note: All current data stored in `stack`, `start_acq`, `texps`, `add_data` will be deleted.

Parameters

- **height** (*int*) – height of images to be stacked
- **width** (*int*) – width of images to be stacked
- **num** (*int*) – number of images to be stacked

last_index
Return last index.

start
Return start time stamp of first image.

stop
Return start time stamp of first image.

time_stamps
Acq. time stamps of all images.

pyrlevel
Gauss pyramid level of images in stack.

camera
Camera object assigned to stack.

num_of_imgs
Depth of stack.

check_index(*idx=0*)

insert_img(*pos*, *img_arr*, *start_acq=datetime.datetime(1900*, *1*, *1*, *0*, *0)*, *texp=0.0*, *add_data=0.0*)
Insert an image into the stack at provided index.

Parameters

- **pos** (*int*) – Insert position of img in stack
- **img_arr** (*array*) – image data (must have same dimension than `self.stack.shape[:2]`, can also be of type `Img`)
- **start_acq** (*datetime*) – acquisition time stamp of image, defaults to datetime(1900, 1, 1)
- **texp** (*float*) – exposure time of image (in units of s), defaults to 0.0
- **add_data** – arbitrary additional data appended to list `add_data`

add_img(*img_arr*, *start_acq=datetime.datetime(1900*, *1*, *1*, *0*, *0)*, *texp=0.0*, *add_data=0.0*)
Add image at current index position.

The image is inserted at the current index position `current_index` which is increased by 1 afterwards. If the latter exceeds the dimension of the actual stack data array `stack`, the stack shape will be extended by 1.

Parameters

- **img_arr** (*array*) – image data (must have same dimension than `self.stack.shape[:2]`)

- **start_acq** (*datetime*) – acquisition time stamp of image, defaults to datetime(1900, 1, 1)
- **texp** (*float*) – exposure time of image (in units of s), defaults to 0.0
- **add_data** – arbitrary additional data appended to list `add_data`

make_circular_access_mask (*cx*, *cy*, *radius*)
Create a circular mask for stack.

Parameters

- **cx** (*int*) – x position of centre
- **cy** (*nint*) – y position of centre
- **radius** (*int*) – radius

Returns circular mask (use e.g. like `img[mask]` which will return a 1D vector containing all pixel values of `img` that fall into the mask)

Return type array

set_stack_data (*stack*, *start_acq=None*, *texps=None*)
Set the current data based on input.

Parameters

- **stack** (*array*) – 3D numpy array containing the image stack data
- **start_acq** (`array`, optional) – array containing acquisition time stamps
- **texps** (obj:*array*, optional) – array containing exposure times

get_data ()
Get stack data (containing of stack, acq. and exp. times).

Returns

3-element tuple containing

- `array`: stack data
- `array`: acq. time stamps
- `array`: exposure times

Return type tuple

apply_mask (*mask*)
Convolves the stack data with a input mask along time axis.

mask [array] 2D bool mask for image pixel access

Returns

3-element tuple containing

- `array`: 3D numpy array containing convolved stack data
- `array`: acq. time stamps
- `array`: exposure times

Return type tuple

get_time_series(*pos_x=None*, *pos_y=None*, *radius=1*, *mask=None*)
Get time series in a ROI.

Retrieve time series at a given pixel position *in stack coordinates* in a circular pixel neighbourhood.

Parameters

- **pos_x** (*int*) – x position of center pixel on detector
- **pos_y** (*int*) – y position of center pixel on detector
- **radius** (*float*) – radius of pixel disk on detector (centered around pos_x, pos_y, default: 1)
- **mask** (*array*) – mask for image pixel access, default is None, if the mask is specified and valid (i.e. same shape than images in stack) then the other three input parameter are ignored

Returns

2-element tuple containing

- `Series`: time series data
- `array`: pixel access mask used to convolve stack images

Return type tuple

merge_with_time_series(*time_series*, *method='average'*, ***kwargs*)
High level wrapper for data merging.

Choose from either of three methods to perform an index merging based on time stamps of stack and of other time series data (provided on input).

Parameters

- **time_series** (*Series*) – time series data supposed to be merged with stack data
- **method** (*str*) – merge method, currently available methods are:
  - average: determine new stack containing images averaged based on start / stop time stamps of each datapoint in input `time_series` (requires corresponding data to be available in input, i.e. `time_series` must be of type `DoasResults` of `pydoas` library).
  - nearest: perform merging based on nearest datapoint per image
  - interpolation: perform cross interpolation onto unified time index array from stack and time series data
- ****kwargs** – additional keyword args specifying additional merge settings (e.g. `itp_type=quadratic` in case `method=interpolation` is used)

Returns

2-element tuple containing

- *ImgStack*: new stack containing merged data
- `Series`: merged time series data

Return type tuple

crop_other_tseries(*time_series*)
Crops other time series object based on start / stop time stamps.

total_time_period_in_seconds()
Return start time stamp of first image.

get_nearest_indices(*tstamps_other*)
Find indices of time stamps nearest to img acq. time stamps.

Parameters **tstamps_other** – datetime, or datetime array of other time series for which closest index / indices are searched

get_nearest_img(*time_stamp*)
Return stack image which is nearest to input timestamp.

Searches the nearest image(s) with respect to input datetime(s)

Parameters **ndarray) time_stamps** (*(datetime,*) – the actual time stamp(s) (for instance from another time series object)

has_data()
Return bool.

sum(**args*, ***kwargs*)
Sum over all pixels of stack.

Parameters

- ***args** – non-keyword arguments passed to *sum()* of numpy array
- ****kwargs** – keyword arguments passed to *sum()* of numpy array

Returns result of summation operation

Return type float

mean(**args*, ***kwargs*)
Apply numpy.mean function to stack data.

Parameters

- ***args** – non keyword arguments passed to `numpy.mean()` applied to stack data
- ****kwargs** – keyword arguments passed to `numpy.mean()` applied to stack data

std(**args*, ***kwargs*)
Apply numpy.std function to stack data.

Parameters

- ***args** – non keyword arguments passed to `numpy.std()` applied to stack data
- ****kwargs** – keyword arguments passed to `numpy.std()` applied to stack data

shape
Return stack shape.

ndim
Return stack dimension.

show_img(*index=0*)
Show image at input index.

Parameters **index** (*int*) – index of image in stack

pyr_down(*steps=0*)
Reduce the stack image size using gaussian pyramid.

Parameters **steps** (*int*) – steps down in the pyramide

Returns new, downscaled image stack object

Return type *ImgStack*

pyr_up (*steps*)
Increasing the image size using gaussian pyramide.

Parameters **steps** (*int*) – steps down in the pyramide

Algorithm used: `cv2.pyrUp()`

to_pyrlevel (*final_state=0*)
Down / upscale image to a given pyramide level.

duplicate ()
Return deepcopy of this object.

load_stack_fits (*file_path*)
Load stack object (fits).

Note: FITS stores in Big-endian and needs to be converted into little-endian (see this issue). We follow the suggested fix and use:

```
byteswap().newbyteorder()
```

on any loaded data array.

Parameters **file_path** (*str*) – file path of stack

save_as_fits (*save_dir=None*, *save_name=None*, *overwrite_existing=True*)
Save stack as FITS file.

pyplis.processing.**find_registration_shift_optflow** (*on_img*, *off_img*, *roi_abs=[0, 0, 9999, 9999]*, ***flow_settings*)
Search average shift between two images using optical flow.

Computes optical flow between two input images and determines the registration shift based on peaks in two histograms of the orientation angle distribution and vector magnitued distribution of the retrieved flow field. The histogram analysis may be reduced to a certain ROI in the images.

The default settings used here correspond to the settings suggested by Peters et al., Use of motion estimation algorithms for improved flux measurements using SO2 cameras, JVGR, 2015.

Parameters

- **on_img** (*Img*) – onband image containing (preferably fixed) objects in the scene that can be tracked
- **off_img** (*Img*) – corresponding offband image (ideally recorded at the same time)
- **roi_abs** (*list*) – if specified, the optical flow histogram parameters are retrieved from the flow field within this ROI (else, the whole image is used)
- ****flow_settings** – additional keyword args specifying the optical flow computation and post analysis settings (see *pyplis.plumespeed.FarnebackSettings* for details)

Returns

2-element tuple containing

- float: shift in x-direction

- float: shift in y-direction

Return type tuple

class pyplis.processing.**PixelMeanTimeSeries**(*data*, *start_acq*, *std=None*, *texps=None*, *roi_abs=None*, *img_prep=None*, ***kwargs*)

A time series of mean pixel values.

This class implements a `pandas.Series` object with extended functionality representing time series data of pixel mean values in a certain image region.

Note: This object is only used to store results of a mean series analysis in a certain ROI, it does not include any algorithms for actually calculating the series

poly_model = None

__init__(*data*, *start_acq*, *std=None*, *texps=None*, *roi_abs=None*, *img_prep=None*, ***kwargs*)

Initialize pixel mean time series.

Parameters

- **data** (*ndarray*) – data array (is passed into pandas Series init -> `self.values`)
- **start_acq** (*ndarray*) – array containing acquisition time stamps (is passed into pandas Series init -> `self.index`)
- **std** (*ndarray*) – array containing standard deviations
- **texps** (*ndarray*) – array containing exposure times
- **roi_abs** (*list*) – image area from which data was extracted, list of shape: `[x0, y0, x1, y1]`
- **img_prep** (*dict*) – dictionary containing information about image preparation settings (e.g. blurring, etc..) or other important information which may need to be stored
- ****kwargs** – additional keyword parameters which are passed to the initiation of the `pandas.Series` object

texps = None

std = None

img_prep = {}

roi_abs = None

start

stop

get_data_normalised(*texp=None*)

Normalise the mean value to a given exposure time.

Parameters **texp (None)** (*float*) – the exposure time to which all deviating times will be normalised. If None, the values will be normalised to the largest available exposure time

Returns A new :class:‘PixelMeanTimeSeries‘instance with normalised data

fit_polynomial(*order=2*)

Fit polynomial to data series.

Parameters **order** (*int*) – order of polynomial

Returns

- poly1d, the fitted polynomial

includes_timestamp (*time_stamp*, *ext_border_secs=0.0*)
Check if input time stamp is included in this dataset.

Parameters

- **time_stamp** (`datetime`) – the time stamp to be checked
- **ext_border_secs** (`float`) – extend start / stop range (default 0 s)

Returns

- bool, True / False (timestamp is within interval)

get_poly_vals (*time_stamps*, *ext_border_secs=0.0*)
Get value of polynomial at input time stamp.

Parameters **time_stamp** (`datetime`) – poly input value

estimate_noise_amplitude (*sigma_gauss=1*, *median_size=3*, *plot=0*)
Estimate the amplitude of the noise in the data.

Steps:

1. **Determines high frequency variations by applying binomial** filter (sigma = 3) to data and subtract this from data, resulting in a residual
2. **Median filtering of residual signal to get rid of narrow peaks** (i.e. where the original data shows abrupt changes)
3. subtract both signals and determine std

..note:

```
Beta version: no guarantee it works for all cases
```

plot (*include_tit=True*, *date_fmt=None*, ***kwargs*)
Plot time series.

Parameters

- **include_tit** (`bool`) – Include a title
- **date_fmt** (`str`) – Date / time formatting string for x labels, passed to `DateFormatter` instance (optional)
- ****kwargs** – Additional keyword arguments passed to pandas Series plot method

Returns matplotlib axes instance

Return type axes

5.15 Fitting / Optimisation algorithms

Module containing optimisation routines.

pyplis.optimisation.**dilution_corr_fit** (*rads*, *dists*, *rad_ambient*, *i0_guess=None*, *i0_min=0.0*, *i0_max=None*, *ext_guess=0.0001*, *ext_min=0.0*, *ext_max=0.001*)
Perform least square fit on data.

Parameters

- **rads** (*ndarray*) – vector containing measured radiances
- **dists** (*ndarray*) – vector containing corresponding dictances
- **rad_ambient** (*float*) – ambient intensity
- **i0_guess** – guess value for initial intensity of topographic features, i.e. the reflected radiation before entering scattering medium (if None, then it is set 5% of the ambient intensity `rad_ambient`)
- **i0_min** (*float*) – minimum initial intensity of topographic features
- **i0_max** (*float*) – maximum initial intensity of topographic features
- **ext_guess** (*float*) – guess value for atm. extinction coefficient
- **ext_min** (*float*) – minimum value for atm. extinction coefficient
- **ext_max** (*float*) – maximum value for atm. extinction coefficient

pyplis.optimisation.**gauss_fit_2d**(*img_arr*, *cx*, *cy*, *g2d_asym=True*, *g2d_super_gauss=True*, *g2d_crop=True*, *g2d_tilt=False*, ***kwargs*)

Apply 2D gauss fit to input image at its maximum pixel coordinate.

Parameters

- **corr_img** (*array*) – correlation image
- **cx** (*float*) – x-position of peak in image (used for initial guess)
- **cy** (*float*) – y-position of peak in image (used for initial guess)
- **g2d_asym** (*bool*) – allow for assymetric shape (sigmax != sigmay), True
- **g2d_super_gauss** (*bool*) – allow for supergauss fit, True
- **g2d_crop** (*bool*) – if True, set outside (1/e amplitude) datapoints = 0, True
- **g2d_tilt** (*bool*) – allow gauss to be tilted with respect to x/y axis

Returns

3-element tuple containing

- array (popt): optimised multi-gauss parameters
- 2d array (pcov): estimated covariance of popt
- 2d array: correlation image

Return type tuple

pyplis.optimisation.**gauss_fit**(*data*, *idx=None*, *has_offset=False*, *plot=False*)

Fit Gaussian function to data.

Parameters

- **data** (*array*) – data array
- **idx** (*array*, optional) – indices of data (e.g. angles)
- **has_offset** (*bool*) – if True, the fitted Gauss is allowed to have a constant offset

Returns optimised parameters of gauss

Return type array

pyplis.optimisation.**get_histo_data**(*data*, ***kwargs*)

Determine histogram of data and set bin array to center of bins.

class pyplis.optimisation.**MultiGaussFit**(*data=None*, *index=None*, *noise_amp=None*, *noise_amp_thresh_fac=2.0*, *sigma_smooth=3*, *sigma_tol_overlaps=3*, *max_num_gaussians=20*, *max_iter=None*, *auto_bounds=True*, *do_fit=True*)

Environment to fit arbitrary amounts of Gaussians to noisy 1D (x,y) data.

It was initally desinged and developed for histogram data and aims to find a solution based on a minimum of required superimposed Gaussians to describe the distribution. Therefore, the fit is performed in a controlled way (i.e. allowed Gaussians are required to be within certain parameter bounds, details below) starting with a noise analysis (if noise level is not provided on class initialisation). Based on the noise level, and the x-range of the data, the boundaries for accepted gauss parameters are set. These are:

```
self.gauss_bounds["amp"][0] = 2*self.noise_amp
self.gauss_bounds["amp"][1] = (self.y_range - self.offset) * 1.5
self.gauss_bounds["mu"][0] = self.index[0]
self.gauss_bounds["mu"][1] = self.index[-1]
self.gauss_bounds["sigma"][0] = self.x_resolution/2.
self.gauss_bounds["sigma"][1] = self.x_range/2.
```

i.e. the amplitude of each of the superimposed Gaussians must be positive and larger then 2 times the noise amplitude. The max allowed amplitude is set 1.5 times the min / max difference of the data. The mean of each Gaussian must be within the index range of the data and the standard deviation must at least be half the x resolution (the smallest allowed peak must be at least have a of FWHM = 1 index) and the max FHWM must not exceed the covered x-range. The fit boundaries can also be set manually using *set_gauss_bounds()* but this might have a strong impact on the quality of the result.

Parameters

- **data** (*array*) – data array
- **index** (array, otional) – x-coordinates
- **noise_amp** (float, optional,) – amplitude of noise in the signal. Defines the minimum required amplitude for fitted Gaussians (you don't want to fit all the noise peaks). If None, it will be estimated automatically on data import using *estimate_noise_amp()*
- **noise_amp_thresh_fac** (*float*) – factor multiplied with noise_amp in order to determine the minimum amplitude threshold required for detecting additional peaks in residual (see *find_additional_peaks()*)
- **sigma_smooth** (*int*) – width of Gaussian kernel to determine smoothed analysis signal (is used to determine data baseline offset)
- **sigma_tol_overlaps** (*int*) – sigma range considered to find overlapping Gauss functions (after fit was applied). This is, for instance used in *analyse_fit_result()* in order to find the main peak parameters
- **max_num_gaussians** (*int*) – max number of superimposed, defaults to 20 Gaussians for data
- **max_iter** (*int*) – max number of iterations for optimisation, if None (default), use max_num_gaussians + 1
- **auto_bounds** (*bool*) – if True, bounds will be set automatically from data ranges whenever data is updated, defaults to True
- **do_fit** (*bool*) – if True and input data available & ok, then *run_optimisation()* will be called on initialisation, defaults to True

__init__(*data=None, index=None, noise_amp=None, noise_amp_thresh_fac=2.0, sigma_smooth=3, sigma_tol_overlaps=3, max_num_gaussians=20, max_iter=None, auto_bounds=True, do_fit=True*)
x.__init__(...) initializes x; see help(type(x)) for signature

residual
Get and return residual.

data_grad
Gradient of analysis signal.

data_grad_smooth
Smoothed gradient of analysis signal.

data_smooth
Smooth data using Gaussian kernel of width `self.sigma_smooth`.

min_amp
Minimum required amplitude to idenitify significant peaks.

init_results()
Initialize all result parameters.

set_data(*data, index=None*)
Set x and y data.

Parameters

- **data** (*array*) – data array which is fitted
- **index** (*:array*) – optional, x index array of data, if None, the array index of data is used

init_data()
Initialize the input data and set constraints for Gaussians.

set_gauss_bounds(*amp_range=None, mu_range=None, sigma_range=None*)
Manually set boundaries for gauss parameters.

Parameters

- **amp_range** (*array*) – accepted amplitude range, defaults to `[0, inf]`
- **mu_range** (*array*) – accepted mu range, defaults to `[-inf, inf]`
- **sigma_range** (*array*) – accepted range of standard deveiations, defaults to `[-inf, inf]`

init_gauss_bounds_auto()
Set parameter bounds for individual Gaussians.

estimate_main_peak_params()
Get rough estimate and position of main peak.

find_additional_peaks()
Search for significant peaks in the current residual.

Returns list containing additional peak parameters (for optimisation), or empty list if no additional peaks can be found

Return type list

estimate_peak_width(*dat, idx*)
Estimate width of a peak at given index.

The peak width is estimated by finding the closest datapoint smaller than 0.5 the amplitude of data at index position

Parameters

- **dat** (*array*) – data (with baseline zero)
- **idx** (*int*) – index position of peak

Returns Estimated peak width in index units

Return type int

prepare_fit_boundaries(*guess*)

Prepare the boundaries tuple.

For details see used least squares solver)

Prepare fit boundary tuple for a multi-gauss parameter array supposed to be optimised (e.g. for two Gaussians this could look like `params=[300, 10, 2, 150, 15, 1]` using the boundaries specified in `self.gauss_bounds`.

Note: If any of the parameters in `params` is out of the acceptance borders specified in `self.gauss_bounds`, the corresponding parameters will be updated to the corresponding boundary value.

Parameters **params** (*list*) – list of gauss parameters (e.g. `self.params`)

Returns

2-element tuple, containing

- `list`: new parameter list (only those matching boundaries)
- `tuple`: corresponding lower and upper boundaries

Return type tuple

do_fit(*x, y, guess*)

Perform a least squares minimisation.

Perform least squares optimisiation for initial set of parameters and input data (includes determination of fit boundary array for all initial peak guesses of input array).

Parameters

- **x** (*array*) – x-argument for model function (index of data)
- **y** (*array*) – y-argument for input function (data)
- **guess** (*list*) – initial guess of fit parameters

Returns True, if optimisation was successful, False if not

Return type bool

opt_iter(*add_params=None*)

Search additional peaks in residual and apply fit.

Extends current optimisation parameters by additional peaks (either provided on input or automatically searched in residual) and performs multi gauss fit.

add_params [list] list containing additional gauss parameters which are supposed to be added to `self.params` before the fit is applied

Returns

- False: Optimisation failed
- True: Optimisation was successful

Return type bool

run_optimisation()
Run optimisation.

result_ok()
Compare current peak to peak residual (ppr) with noise amplitude.

Returns bool 1 if `2*self.noise_amp > ppr`, else 0

find_overlaps(*sigma_tol=None*)
Find overlapping Gaussians for current optimisation params.

Loops over all current Gaussians (`self.gaussians`) and for each of them, finds all which fall into 3 $sigma$ range.

Parameters **sigma_tol** (`float`, optional) – sigma tolerance level for finding overlapping Gaussians, if None, use `sigma_tol_overlaps`.

Returns

2-element tuple containing

- **list, contains all Gaussians overlapping with Gaussian (within** sigma tolerance range defined by `sigma_tol`) at index *k* in list returned by `gaussians()`.
- list, integral values of each of the overlapping sub regions

Return type tuple

analyse_fit_result(*sigma_tol_overlaps=None*)
Analyse result of optimisation.

Find main peak (can be overlap of single Gaussians) and potential other peaks.

Parameters **sigma_tol** (`float`, optional) – sigma tolerance level for finding overlapping Gaussians, if None, use `sigma_tol_overlaps`.

Returns

4-element tuple containing

- `float`: center position (
  mu) of predominant peak
- `float`: corresponding standard deviation
- `float`: integral value of predominant peak
- **`list`: additional Gaussians (from fit result) that are** not lying within specified tolerance interval of main peak

Return type tuple

analyse_fit_result_old(*sigma_tol=None*)
Analyse result of optimisation.

Find main peak (can be overlap of single Gaussians) and potential other peaks.

Parameters **sigma_tol** (`float`, optional) – sigma tolerance level for finding overlapping Gaussians, if None, use `sigma_tol_overlaps`.

Returns

4-element tuple containing

- `float`: center position (mu) of predominant peak
- `float`: corresponding standard deviation
- `float`: integral value of predominant peak
- **`list`: additional Gaussians (from fit result) that are** not lying within specified tolerance interval of main peak

Return type tuple

`normalise_params`(*params=None*)
Get normalised distribution of Gaussians.

`gaussians`()
Get list containing fitted parameters for each Gaussian.

`integrate_gauss`(*amp*, *mu*, *sigma*, *start=-inf*, *stop=inf*)
Return integral value of one Gaussian.

Parameters

- **amp** (*float*) – amplitude of Gaussian
- **mu** (*float*) – mean of Gaussian
- **sigma** (*float*) – standard deviation
- **start** – left integral border, defaults to $-\infty$
- **stop** – right integral border, defaults to ∞

`integrate_all_gaussians`(*params=None*)
Determine the integral values of all Gaussians in `self.gaussians`.

Returns list integral values for each Gaussian

`det_moment`(*index*, *data*, *center*, *n*)
Determine n-th moment of distribution.

`create_test_data_singlegauss`(*add_noise=True*, *noise_frac=0.05*)
Make a test data set containing a single Gaussian (without offset).

The parameters of the Gaussian are `[300, 150, 20]`

Parameters

- **add_noise** (*bool*) – add noise to test data
- **noise_frac** (*float*) – determines noise amplitude (fraction relative to max amplitude of Gaussian)

`create_test_data_multigauss`(*add_noise=True*, *noise_frac=0.03*)
Create test data set containing 5 overlapping Gaussians.

Parameters

- **add_noise** (*bool*) – add noise to test data
- **noise_frac** (*float*) – determines noise amplitude (fraction relative to max amplitude of Gaussian)

create_test_data_multigauss2(*add_noise=True*, *noise_frac=0.03*)
Create test data set containing 5 overlapping Gaussians.

Parameters

- **add_noise** (*bool*) – add noise to test data
- **noise_frac** (*float*) – determines noise amplitude (fraction relative to max amplitude of Gaussian)

create_noise_dataset()
Make pure noise and set as current data.

apply_binomial_filter(*data=None*, *sigma=1*)
Return filtered data using 1D gauss filter.

Parameters

- **data** (array, optional) – data to be smoothed, if None, use `self.data`
- **sigma** (*int*) – width of smoothing kernel, defaults to 1

Returns smoothed data array

Return type array

first_derivative(*data=None*)
Determine and return first derivatieve of data.

The derivative is determined using the numpy method `gradient()`

Parameters **data** (array, optional) – data to be smoothed, if None, use `self.data`

Returns array containing gradients

Return type array

set_noise_amp(*ampl*)
Set the current fit amplitude threshold.

Parameters **ampl** (*float*) – amplitude of noise level

estimate_noise_amp(*sigma_gauss=3*, *sigma_tol=3*, *cut_out_width=None*)
Estimate the noise amplitude of the current data.

Parameters

- **sigma_gauss** (*int*) – width of smoothing kernel applied to data in order to determine analysis signal
- **sigma_tol** (*float*) – factor by which noise signal standard deviation is multiplied in order to estimate noise amplitude
- **cut_out_width** – specifyies the width of index neighbourhood around narrow peaks which is to be disregarded for statistics of noise amplitude. Such narrow peaks can remain in the analysis signal. If None, it is set 3 times the width of the smoothing kernel used to determine the analysis signal.

Returns

3-element tuple containing

- `float`: the analysis signal
- `array`: mask specifying indices used to determine the ampl.
- `array`: inital index array

Return type tuple

max()
Return max value and x position of current parameters (not of data).

num_of_gaussians
Get the current number of Gaussians, which is the length.

Returns

- float, `len(self.params) // 3`

max_amp
Get the max amplitude of the current fit results.

y_range
Range of y values.

x_range
Range of x values.

x_resolution
Return resolution of x data array.

get_sub_intervals_bool_array(*bool_arr*)
Get all subintervals of the input bool array.

Note: Currently not in use, but might be helpful at any later stage

---

get_residual(*params=None*, *mask=None*)
Get the current residual.

Parameters

- **params** (*list*) – Multi gauss parameters, if None, use `self.params`
- **mask** (*logical*) – use only certain indices

get_peak_to_peak_residual(*params=None*)
Return peak to peak difference of fit residual.

Parameters **params** (*list*) – mutligauss optimisation parameters, if default (None), use `self.params`

cut_sigma_range(*x*, *y*, *params*, *n_sigma=4*)
Cut out a N x sigma environment around Gaussian from data.

Parameters

- **x** (*array*) – x-data array
- **y** (*array*) – y-data array
- **params** (*list*) – Gaussian fit parameters [ampl, mu, sigma]
- **n_sigma** (*int*) – N (e.g. 3 => 3* sigma environment will be cutted)

check_peak_bounds(*params*)
Check if gauss params fulfill current boundary conditions.

Parameters **params** – parameters of a single gauss `[amp, mu, sigma]`

get_all_gaussians_within_sigma_range(*mu*, *sigma*, *sigma_tol=None*)
Find all current Gaussians within sigma range of a Gaussian.

Parameters

- **mu** (*float*) – mean (x pos) of considered Gaussian
- **sigma** (*float*) – corresponding standard deviation
- **sigma_tol** (float, optional) – sigma tolerance factor for finding overlaps, if None, use `sigma_tol_overlaps`

Returns list containing parameter lists `[amp, mu, sigma]` for all Gaussians of the current fit result having their mu values within the specified sigma interval of the input Gaussian

Return type list

get_all_gaussians_out_of_sigma_range (*mu*, *sigma*, *sigma_tol=None*)

Find all current Gaussians out of sigma range of a Gaussian.

Parameters

- **mu** (*float*) – mean (x pos) of considered Gaussian
- **sigma** (*float*) – corresponding standard deviation
- **sigma_tol** (float, optional) – sigma tolerance factor for finding overlaps, if None, use `sigma_tol_overlaps`

Returns list containing parameter lists `[amp, mu, sigma]` for all Gaussians of the current fit result having their mu values within the specified sigma interval of the input Gaussian

Return type list

plot_signal_details ()

Plot signal and derivatives both in original and smoothed version.

Returns axes of two subplots

Return type array

plot_data (*ax=None*, *sub_min=False*)

Plot the input data.

Parameters

- **ax** – matplotlib axes object (default = None)
- **sub_min** (*bool*) – if true, `self.offset` will be subtracted from data, (default = False)

plot_multi_gaussian (*x=None*, *params=None*, *ax=None*, *color='r'*, *lw=2*, ***kwargs*)

Plot multi gauss.

Parameters

- **x** (*array*) – x data array, if None, use `self.index` (default = None)
- **params** (*list*) – multi gauss parameter list if None, use `self.params` (default = None)
- **ax** (*axes*) – matplotlib axes object (default = None)
- ****kwargs** – additional keyword args passed to matplotlib plot method

plot_gaussian (*x*, *params*, *ax=None*, ***kwargs*)

Plot gaussian.

Parameters

- **x** (*array*) – x data array

- **params** (*list*) – single gauss parameter list
- **ax** (*axes*) – matplotlib axes object (default = None)
- ****kwargs** – additional keyword args passed to matplotlib plot method

plot_result (*add_single_gaussians=False*, *figsize=(16, 10)*)
Plot the current fit result.

Parameters **add_single_gaussians** (*bool*) – if True, all individual Gaussians are plotted

print_gauss (*ind*)
Print gauss string.

Parameters **ind** (*int*) – index of gauss in `self.params`

gauss_str (*g*)
Return string representation of a Gaussian.

Parameters **g** (*list*) – gauss parameter list `[amp, mu, sigma]`

info ()
Print string representation.

has_data
Return True, if data available, else False.

has_results ()
Check if fit results are available.

class pyplis.optimisation.**PolySurfaceFit** (*data_arr*, *mask=None*, *polyorder=3*, *pyrlevel=4*, *do_fit=1*)
Fit a 2D polynomial to data (e.g. a blue sky background image).

This class can be used to fit 2D polynomials to image data. It includes specifying pixels supposed to be used for the fit which have to be provided using a mask. The fit can be performed at arbitrary Gauss pyramid levels which can dramatically increase the performance.

Note: The fit result image can be accessed via the attribute `model`

Parameters

- **data_arr** (*array*) – image data to be fitted (NxM matrix)
- **mask** (*array*) – mask specifying pixels considered for the fit (if None, then all pixels of the image data are considered
- **polyorder** (*int*) – order of polynomial for fit (default=3)
- **pyrlevel** (*int*) – level of Gauss pyramid at which the fit is performed (relative to Gauss pyramid level of input data)
- **do_fit** (*bool*) – if True, and if input data is valid, then the fit is performed on intialisation of the class

__init__ (*data_arr*, *mask=None*, *polyorder=3*, *pyrlevel=4*, *do_fit=1*)
x.__init__(. . .) initializes x; see help(type(x)) for signature

set_data (*data_arr*, *mask=None*)
Set the data array (must be dimension 2).

Create `self.mask` for array shape which can be used to exclude picxel areas from the image

Parameters

- **data_arr** (*array*) – image data (can also be `Img`)
- **mask** (*array*) – boolean mask specifying pixels considered for fit, if None, all pixels are considered

Returns True if data is valid, False if not

Return type bool

activate_auto_update(*val=1*)
Activate or deactivate auto update mode.

If active, the fit is reapplied each time some input parameter is changed

Parameters **val** (*bool*) – new value for `auto_update`

change_pyrlevel(*newlevel*)
Change the level in the Gaussian pyramide where the fit is applied.

change_polyorder(*new_order*)
Change the order of the polynomial which is fitted.

Sets new poly order and re-applies fit in case `auto_update == True`

Parameters **new_order** (*int*) – new order of poly fit

exclude_pix_range_rect(*x0, x1, y0, y1*)
Add a rectangular pixel area which will be excluded from the fit.

Parameters

- **x0** (*int*) – start x coordinate (original image resolution)
- **x1** (*int*) – stop x coordinate (original image resolution)
- **y0** (*int*) – start y coordinate (original image resolution)
- **y1** (*int*) – stop y coordinate (original image resolution)

exclude_pix_range_circ(*x0, y0, r*)
Add a circular pixel area which will be excluded from the fit.

Parameters

- **x0** (*int*) – centre x coordinate (original image resolution)
- **y0** (*int*) – centre y coordinate (original image resolution)
- **r** (*int*) – radius in pixel coordinates (original image resolution)

pyr_down(*arr, steps*)
Reduce the image size using Gaussian pyramide.

Parameters **steps** (*int*) – steps down in the pyramide

Algorithm used: `cv2.pyrDown()`

pyr_up(*arr, steps*)
Increase the image size using Gaussian pyramide.

Parameters **steps** (*int*) – steps down in the pyramide

Algorithm used: `cv2.pyrUp()`

make_weights_mask()
Make a mask for pixel fit weights based on values in self.mask.

do_fit()
Apply the fit to the data and write results.

get_residual()
Get the current residual image array.

plot_result()
Plot the fit result.

Plots the results, the original data and the residual in two versions (2 different intensity ranges)

5.16 Mathematical model functions

Pyplis module containing mathematical model functions.

pyplis.model_functions.**cfun_kern2015**(*x*, *a0*, *a1*)

pyplis.model_functions.**cfun_kern2015_offs**(*x*, *a0*, *a1*, *a2*)

class pyplis.model_functions.**CalibFuns**
Class containing functions for fit of calibration curve.

__init__()
x.__init__(. . .) initializes x; see help(type(x)) for signature

available_poly_orders(*through_origin=False*)
Return the available polynomial orders.

through_origin [bool] polys without offset

Returns list containing available polyorders

Return type list

print_poly_info()
Print information about available polynomials.

print_custom_funs_info()
Print information about available curtom calib functions.

get_custom_fun(*key='kern2015'*)
Return an available custom calibration function.

Parameters key (*str*) – access key of custom function (call print_custom_funs_info() for info about available functions)

Returns

Return type the function object

get_poly(*order=1*, *through_origin=False*)
Get a polynomial of certain order.

Parameters

- **order** (*int*) – order of polynomial (choose from 1-3)
- **through_origin** (*bool*) – if True, the polynomial will have no offset term

Returns the polynomial function object (callable)

Return type function

`pyplis.model_functions.`**`dilutioncorr_model`**(*dist*, *rad_ambient*, *i0*, *ext*)

Model function for light dilution correction.

This model is based on the findings of Campion et al., 2015.

Parameters

- **dist** (*float*) – distance of dark (black) object in m
- **rad_ambient** (*float*) – intensity of ambient atmosphere at position of dark object
- **i0** (*float*) – initial intensity of dark object before it enters the scattering medium. It is determined from the illumination intensity and the albedo of the dark object.
- **atm_ext** (*float*) – atmospheric scattering extincion coefficient ϵ (in Campion et al., 2015 denoted with σ).

`pyplis.model_functions.`**`gaussian_no_offset`**(*x*, *ampl*, *mu*, *sigma*)

1D gauss with baseline zero.

Parameters

- **x** (*float*) – x position of evaluation
- **ampl** (*float*) – Amplitude of gaussian
- **mu** (*float*) – center poistion
- **sigma** (*float*) – standard deviation

Returns float value at position x

`pyplis.model_functions.`**`gaussian`**(*x*, *ampl*, *mu*, *sigma*, *offset*)

1D gauss with arbitrary baseline.

Parameters

- **x** (*float*) – x position of evaluation
- **ampl** (*float*) – Amplitude of gaussian
- **mu** (*float*) – center poistion
- **sigma** (*float*) – standard deviation
- **offset** (*float*) – baseline of gaussian

Returns float value at position x

`pyplis.model_functions.`**`multi_gaussian_no_offset`**(*x*, **params*)

Superimposed 1D gauss functions with baseline zero.

Parameters

- **x** (*array*) – x array used for evaluation
- ***params** (*list*) – List of length L = 3xN were N corresponds to the number of gaussians e.g.:

```
[100,10,3,50,15,6]
```

would correspond to 2 gaussians with the following characteristics:

1. Peak amplitude: 100, Mu: 10, sigma: 3
2. Peak amplitude: 50, Mu: 15, sigma: 6

pyplis.model_functions.**multi_gaussian_same_offset**(*x*, *offset*, **params*)
Superimposed 1D gauss functions with baseline (offset).

See *multi_gaussian_no_offset()* for instructions

pyplis.model_functions.**supergauss_2d**(*position*, *amplitude*, *xm*, *ym*, *sigma*, *asym*, *shape*, *offset*)
2D super gaussian without tilt.

Parameters

- **position** (*tuple*) – position (x, y) of Gauss
- **amplitude** (*float*) – amplitude of peak
- **xm** (*float*) – x position of maximum
- **ym** (*float*) – y position of maximum
- **asym** (*float*) – assymetry in y direction (1 is circle, smaller means dillated in y direction)
- **shape** (*float*) – super gaussian shape parameter (1 is gaussian)
- **offset** (*float*) – base level of gaussian

pyplis.model_functions.**supergauss_2d_tilt**(*position*, *amplitude*, *xm*, *ym*, *sigma*, *asym*, *shape*, *offset*, *theta*)
2D super gaussian without tilt.

Parameters

- **position** (*tuple*) – position (x, y) of Gauss
- **amplitude** (*float*) – amplitude of peak
- **xm** (*float*) – x position of maximum
- **ym** (*float*) – y position of maximum
- **asym** (*float*) – assymetry in y direction (1 is circle, smaller means dillated in y direction)
- **shape** (*float*) – super gaussian shape parameter (2 is gaussian)
- **offset** (*float*) – base level of gaussian
- **theta** (*float*) – tilt angle (rad) of super gaussian

5.17 I/O routines

Module containing all sorts of I/O-routines (e.g. test data access).

pyplis.inout.**data_search_dirs**()
Get basic search directories for package data files.

Data files are searched for in *~/my_pyplis*, *./data* and, if set, in the *PYPLIS_DATADIR* environment variable.

pyplis.inout.**zip_example_scripts**(*repo_base*)

pyplis.inout.**get_all_files_in_dir**(*directory*, *file_type=None*, *include_sub_dirs=False*)
Find all files in a certain directory.

Parameters

- **directory** (*str*) – path to directory
- **file_type** (str, optional) – specify file type (e.g. "png", "fts"). If unspecified, then all files are considered

- **include_sub_dirs** (*bool*) – if True, also all files from all sub-directories are extracted

Returns sorted list containing paths of all files detected

Return type list

`pyplis.inout.`**`create_temporary_copy`**(*path*)

`pyplis.inout.`**`download_test_data`**(*save_path=None*)
Download pyplis test data.

Parameters **save_path** – location where path is supposed to be stored

Code for progress bar was "stolen" here (last access date: 11/01/2017) -progress-bar-in-python

`pyplis.inout.`**`find_test_data`**()
Search location of test data folder.

`pyplis.inout.`**`all_test_data_paths`**()
Return list of all search paths for test data.

`pyplis.inout.`**`set_test_data_path`**(*save_path*)
Set local path where test data is stored.

`pyplis.inout.`**`get_camera_info`**(*cam_id*)
Try access camera information from file "cam_info.txt" (package data).

Parameters **cam_id** (*str*) – string ID of camera (e.g. "ecII")

`pyplis.inout.`**`save_new_default_camera`**(*info_dict*)
Save new default camera to data file *cam_info.txt*.

Parameters **info_dict** (*dict*) – dictionary containing camera default information

Only valid keys will be added to the

`pyplis.inout.`**`save_default_source`**(*info_dict*)
Add a new default source to file source_info.txt.

`pyplis.inout.`**`get_all_valid_cam_ids`**()
Load all valid camera string ids.

Reads info from file cam_info.txt which is part of package data

`pyplis.inout.`**`get_cam_ids`**()
Load all default camera string ids.

Reads info from file cam_info.txt which is part of package data

`pyplis.inout.`**`get_source_ids`**()
Get all existing source IDs.

Reads info from file my_sources.txt which is part of package data

`pyplis.inout.`**`get_source_info`**(*source_id*, *try_online=True*)
Try access source information from file "my_sources.txt".

File is part of package data

Parameters

- **source_id** (*str*) – string ID of source (e.g. Etna)
- **try_online** (*bool*) – if True and local access fails, try to find source ID in online database

pyplis.inout.**get_source_info_online**(*source_id*)
Try to load source info from online database (@ www.ngdc.noaa.gov).

Parameters **source_id** (*str*) – ID of source

pyplis.inout.**get_icon**(*name*, *color=None*)
Try to find icon in lib icon folder.

Parameters

- **name** (*str*) – name of icon (i.e. filename is <name>.png)
- **(None)** (*color*) – color of the icon ("r", "k", "g")

Returns icon image filepath if valid

5.18 Custom image import methods

Custom image load methods for different camera standards.

Note: This file can be used to include cusotmised image import method. Please re-install pyplis after defining your customised import method here. The method requires the following input / output:

1. Input: `str`, file_path -> full file path of the image
2. **Optional input: dict, dictionary specifying image meta information** (e.g. extracted from file name before image load)
3. Two return parameters
   1. `ndarray`, the image data (2D numpy array)
   2. `dict`, additional meta information (is required as return value

      , if no meta data is imported from your custom method, then simply return an empty dictionary. Please also make sure to use valid pyplis image meta data keys (listed below)

Valid keys for import of image meta information:

'start_acq', 'stop_acq', 'texp', 'focal_length', 'pix_width', 'pix_height', 'bit_depth', 'f_num', 'read_gain', 'filter', 'path', 'file_name', 'file_type', 'device_id', 'ser_no', 'wvlngth', 'fits_idx', 'temperature', 'user_param1', 'user_param2', 'user_param3'

pyplis.custom_image_import.**load_ecII_fits**(*file_path*, *meta=None*, ***kwargs*)
Load NILU ECII camera FITS file and import meta information.

pyplis.custom_image_import.**load_hd_custom**(*file_path*, *meta=None*, ***kwargs*)
Load image from HD custom camera.

The camera specs can be found in Kern et al. 2015, Intercomparison of SO2 camera systems for imaging volcanic gas plumes

Images recorded with this camera type are stored as .tiff files and are

Parameters

- **file_path** – image file path
- **meta** (*dict*) – optional, meta info dictionary to which additional meta information is suppose to be appended

Returns

- ndarray, image data
- dict, dictionary containing meta information

`pyplis.custom_image_import.`**`load_hd_new`**(*file_path*, *meta=None*, ***kwargs*)

Load new format from Heidelberg group.

This format contains IPTC information

Parameters

- **`file_path`** – image file path
- **`meta`** (*`dict`*) – optional, meta info dictionary to which additional meta information is supposed to be appended

Returns

- ndarray, image data
- dict, dictionary containing meta information

`pyplis.custom_image_import.`**`load_qsi_lmv`**(*file_path*, *meta=None*, ***kwargs*)

Load images for QSI cam from LMV.

Laboratoire Magmas et Volcans, Université Clermont Auvergne - CNRS - IRD, OPGC

This format contains IPTC information

Parameters

- **`file_path`** (*`str`*) – image file path
- **`meta`** (*`dict`*) – optional, meta info dictionary to which additional meta information is supposed to be appended

Returns

2-element tuple, containing:

- ndarray, image data
- dict, dictionary containing meta information

Return type tuple

`pyplis.custom_image_import.`**`load_usgs_multifits`**(*file_path*, *meta=None*)

`pyplis.custom_image_import.`**`load_usgs_multifits_uncompr`**(*file_path*, *meta=None*)

`pyplis.custom_image_import.`**`load_comtessa`**(*file_path*, *meta=None*)

Load image from a multi-layered fits file (several images in one file).

Meta data is available only inside the header.

This corresponds to image data from the COMTESSA project at Norwegian Institute for Air Research.

Note: The comtessa *.fits files have several timestamps: 1) Filename –> minute in which the image was saved. 2) Meta information in the image header –> computer time when the image was saved. 3) First 14 image pixels contain a binary timestamp –> time when exposure was finished. Here nr 3) is saved as meta['stop_acq']. meta['start_acq'] is calculated from meta['stop_acq'] and meta['texp']. meta['user_param1'] is the gain (float type).

Parameters

- **file_path** (*string*) – image file path
- **meta** (*dictionary*) – optional, meta info dictionary to which additional meta information is appended. The image index should be provided with key "fits_idx".

Returns

- *ndarray* – image data
- *dict* – dictionary containing meta information

5.19 Helper functions

Pyplis module containing all sorts of helper methods.

pyplis.helpers.**exponent**(*num*)

pyplis.helpers.**matlab_datenum_to_datetime**(*num*)

pyplis.helpers.**get_pyr_factor_rel**(*img1*, *img2*)

Get difference in pyramid level between two input images.

Parameters

- **img1** (Img or ndarray) – First image
- **img2** (Img or ndarray) – Second image

Raises **ValueError** – if image shapes can not be matched by changinf the pyramid level of either of the 2 images

Returns Difference in Gauss pyramid level of img2 relative to img1, i.e. a negative number means, that **:param:'img2'** is larger than **:param:'img1'**

Return type int

pyplis.helpers.**nth_moment**(*index*, *data*, *center=0.0*, *n=0*)

Determine n-th moment of distribution.

Parameters

- **index** (*array*) – data x axis index array
- **data** (*array*) – the data distribution
- **center** (*float*) – coordinate around which the moment is determined
- **n** (*int*) – number of moment

pyplis.helpers.**set_ax_lim_roi**(*roi*, *ax*, *xy_aspect=None*)

Update axes limits to ROI coords (for image disp).

Note: Hard coded in a rush, probably easier solution to it ;)

Parameters

- **roi** (*list*) – `[x0, y0, x1, y1]`
- **ax** (*Axes*) – the Axes showing the image

Returns trivial

Return type Axes

`pyplis.helpers.closest_index(time_stamp, time_stamps)`
Find index of time stamp in array to other time stamp.

Parameters

- **time_stamp** (*datetime*) – time stamp for which closest match is searched
- **time_stamps** (*iterable*) – ordered list of time stamps to be searched (i.e. first index is earliest, last is latest)

Returns index of best match

Return type int

`pyplis.helpers.to_datetime(value)`
Evaluate time and / or date input and convert to datetime.

`pyplis.helpers.isnum(val)`
Check if input is number (int or float) and not nan.

Returns bool, True or False

`pyplis.helpers.mesh_from_img(img_arr)`
Create a mesh from an 2D numpy array (e.g. image).

Parameters **img_arr** (*ndarray*) – 2D numpy array

Returns mesh

`pyplis.helpers.make_circular_mask(h, w, cx, cy, radius, inner=True)`
Create a circular access mask for accessing certain pixels in an image.

Parameters

- **h** (*int*) – height of mask
- **w** (*int*) – width of mask
- **cx** (*int*) – x-coordinate of center pixel of disk
- **cy** (*int*) – y-coordinate of center pixel of disk
- **radius** (*int*) – radius of disk
- **inner** (*bool*) – if True, all pixels within the disk are True, all outside are False, vice versa if False

Returns the pixel access mask

Return type ndarray

`pyplis.helpers.get_img_maximum(img_arr, add_blur=0)`
Get coordinates of maximum in image.

Parameters

- **img_arr** (*array*) – numpy array with image data data
- **gaussian_blur** (*int*) – apply gaussian filter before max search

`pyplis.helpers.sub_img_to_detector_coords(img_arr, shape_orig, pyrlevel, roi_abs=None)`
Convert a shape manipulated image to original detecor coords.

Parameters

- **img_arr** (*ndarray*) – the sub image array (e.g. corresponding to a certain ROI and / or pyrlevel)
- **shape_orig** (*tuple*) – original image shape (detector dimension)
- **pyrlevel** (*int*) – the pyramid level of the sub image
- **roi_abs** (*list*) – region of interest (in absolute image coords) of the sub image

Note: Regions outside the ROI are set to 0

pyplis.helpers.**check_roi**(*roi*, *shape=None*)
Check if input fulfills all criteria for a valid ROI.

Parameters

- **roi** – the ROI candidate to be checked
- **shape** (*tuple*) – dimension of image for which the ROI is supposed to be checked (optional)

pyplis.helpers.**subimg_shape**(*img_shape=None*, *roi=None*, *pyrlevel=0*)
Get shape of subimg after cropping and size reduction.

Parameters

- **img_shape** (*tuple*) – original image shape
- **roi** (*list*) – region of interest in original image, if this is provided img_shape param will be ignored and the final image size is determined based on a cropped image within the roi
- **pyrlevel** (*int*) – scale space parameter (Gauss pyramide) for size reduction

Returns

- tuple, (height, width) of (cropped and) size reduced image

pyplis.helpers.**same_roi**(*roi1*, *roi2*)
Check if two ROIs are the same.

Parameters

- **roi1** (*list*) – list with ROI coords `[x0, y0, x1, y1]`
- **roi2** (*list*) – list with ROI coords `[x0, y0, x1, y1]`

pyplis.helpers.**roi2rect**(*roi*, *inverse=False*)
Convert ROI to rectangle coordinates or vice versa.

Parameters

- **roi** (*list*) – list containing ROI corner coords `[x0 , y0, x1, y1]` (input can also be tuple)
- **inverse** (*bool*) – if True, input param `roi` is assumed to be of format `[x0, y0, w, h]` and will be converted into ROI

Returns

- tuple, (x0, y0, w, h) if param `inverse == False`
- tuple, (x0, y0, x1, y1) if param `inverse == True`

pyplis.helpers.**map_coordinates_sub_img**(*pos_x_abs*, *pos_y_abs*, *roi_abs=None*, *pyrlevel=0*, *inverse=False*)
Map absolute pixel coordinate to cropped and / or downscaled image.

Parameters

- **pos_x_abs** (*int*) – x coordinate in absolute image coords (can also be an array of coordinates)
- **pos_y_abs** (*int*) – y coordinate in absolute image coords (can also be an array of coordinates)
- **roi_abs** (*list*) – list specifying rectangular ROI in absolute image coordinates (i.e. `[x0, y0, x1, y1]`)
- **pyrlevel** (*list*) – level of gauss pyramid
- **inverse** (*bool*) – if True, do inverse transformation (False)

`pyplis.helpers.`**`map_roi`**(*roi_abs*, *pyrlevel_rel=0*, *inverse=False*)

Map a list containing start / stop coords onto size reduced image.

Parameters

- **roi_abs** (*list*) – list specifying rectangular ROI in absolute image coordinates (i.e. `[x0, y0, x1, y1]`)
- **pyrlevel_rel** (*int*) – gauss pyramid level (relative, use negative numbers to go up)
- **inverse** (*bool*) – inverse mapping

Returns

- roi coordinates for size reduced image

`pyplis.helpers.`**`shifted_color_map`**(*vmin*, *vmax*, *cmap=None*)

Shift center of a diverging colormap to value 0.

Note: This method was found here (last access: 17/01/2017). Thanks to Paul H who provided it.

Function to offset the "center" of a colormap. Useful for data with a negative min and positive max and if you want the middle of the colormap's dynamic range to be at zero level

Parameters

- **vmin** – lower end of data value range
- **vmax** – upper end of data value range
- **cmap** – colormap (if None, use default cmap: seismic)

Returns

- shifted colormap

`pyplis.helpers.`**`rotate_xtick_labels`**(*ax*, *deg=30*, *ha='right'*)

Rotate xtick labels in matplotlib axes object.

`pyplis.helpers.`**`rotate_ytick_labels`**(*ax*, *deg=30*, *va='bottom'*)

Rotate xtick labels in matplotlib axes object.

`pyplis.helpers.`**`bytescale`**(*data*, *cmin=None*, *cmax=None*, *high=255*, *low=0*)

Bytescale an image array.

Byte scales an array (image).

Note: This function was copied from the Python Imaging Library module pilutil in order to ensure stability due to re-occuring problems with the PIL installation / import.

Byte scaling means converting the input image to uint8 dtype and scaling the range to `(low, high)` (default 0-255). If the input image already has dtype uint8, no scaling is done.

Parameters

- **data** (*ndarray*) – image data array
- **cmin** – optional, bias scaling of small values. Default is `data.min()`
- **cmin** – optional, bias scaling of large values. Default is `data.max()`
- **high** – optional, scale max value to *high*. Default is 255
- **low** – optional, scale min value to *low*. Default is 0

Returns

- uint8, byte-scaled 2D numpy array

Examples

```
>>> from pyplis.helpers import bytescale
>>> img = np.array([[ 91.06794177,   3.39058326,  84.4221549 ],
...                 [ 73.88003259,  80.91433048,   4.88878881],
...                 [ 51.53875334,  34.45808177,  27.5873488 ]])
>>> bytescale(img)
array([[255,   0, 236],
       [205, 225,   4],
       [140,  90,  70]], dtype=uint8)
>>> bytescale(img, high=200, low=100)
array([[200, 100, 192],
       [180, 188, 102],
       [155, 135, 128]], dtype=uint8)
>>> bytescale(img, cmin=0, cmax=255)
array([[91,  3, 84],
       [74, 81,  5],
       [52, 34, 28]], dtype=uint8)
```

5.20 Forms and geometrical objects

class pyplis.forms.**FormCollectionBase**(*forms_dict=None*)

Abtract base class representing collection of geometrical forms.

Abstract class providing basic functionality for object collections. Note that the basic management functions for adding / deleting forms *add()* , *remove()* create the objects based on start (x,y) and stop (x,y) points, i.e. [x0, y0, x1, y1]

This class and classes inheriting from it show large similarities to dictionaries

__init__(*forms_dict=None*)

Class initialisation.

add(*x0*, *y0*, *x1*, *y1*, *id=None*)

Create a new form from input coordinates.

Parameters

- **x0** (*int*) – x coordinate of start point
- **y0** (*int*) – y coordinate of start point
- **x1** (*int*) – x coordinate of stop point
- **y1** (*int*) – y coordinate of stop point
- **id** (*str*) – name of form (if None, it will be set automatically based on current object counter value)

tot_num
Return current number of forms in collection.

update (*x0*, *y0*, *x1*, *y1*, *id*)
Update an existing form (or create new if it does not exist).

check_input (*x0*, *y0*, *x1*, *y1*)
Check if input for adding a form has the right format.

remove (*id*)
Remove one form from collection.

Parameters **id** (*str*) – string id of the form to be deleted

rename (*current_id*, *new_id*)
Rename one form.

has_form (*form_id*)
Check if collection has a form with input ID, returns bool.

form_info (*form_id*)
Print information about one of the forms in this collection.

Parameters **form_id** (*str*) – string ID of form

keys ()
Return names of all current forms.

values ()
Return all current forms.

get (*form_id*)
Get one form.

Parameters **form_id** (*str*) – name of the form within collection

class pyplis.forms.**LineCollection** (*forms_dict=None*)
Class specifying line objects on images.

__init__ (*forms_dict=None*)
Class initialisation.

class pyplis.forms.**RectCollection** (*forms_dict=None*)
Class specifying rectangle objects on images.

__init__ (*forms_dict=None*)
Class initialisation.

add (*x0*, *y0*, *x1*, *y1*, *id=None*)
Create a new form from input coordinates.

Parameters

- **x0** (*int*) – x coordinate of start point
- **y0** (*int*) – y coordinate of start point
- **x1** (*int*) – x coordinate of stop point
- **y1** (*int*) – y coordinate of stop point
- **id** (*str*) – identification string of object (if None, the id will be set automatically based on current object counter value)

CHAPTER 6

Supplementary material

Pyplis download area

6.1 Poster EGU General Assembly, Vienna 2017

Download the poster `PDF here`

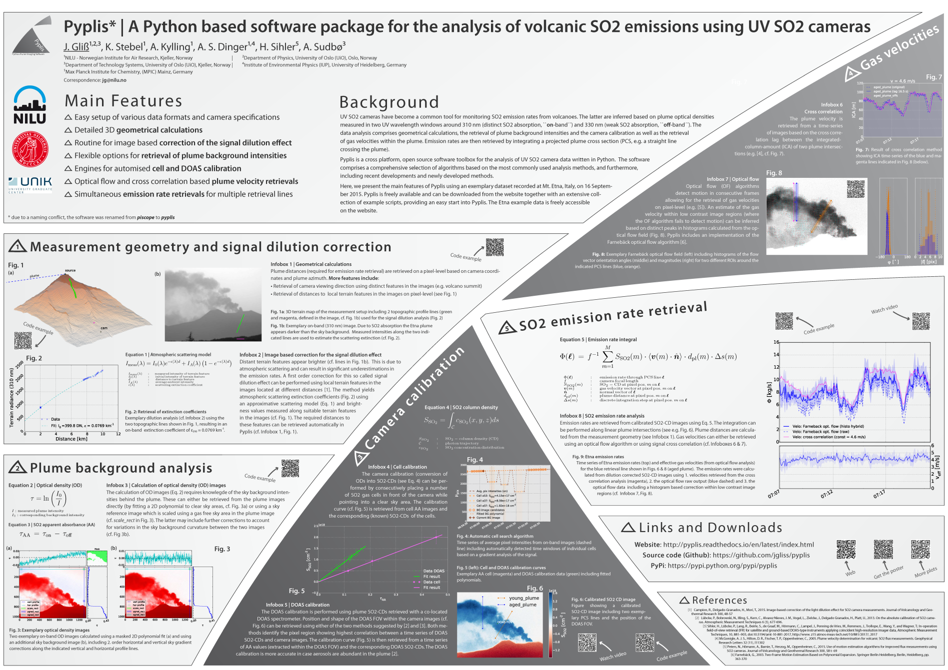


Fig. 1: Pyplis poster for the EGU General Assembly meeting, Vienna, April 2017

CHAPTER 7

Changelog

Automatically generated (and slightly modified) using:

```
git log --pretty=format:"- %ad, %aN%x09%s" --date=short v1.3.0..HEAD > changelog/
↪CHANGELOG_v130_v143.rst
```

7.1 Release 1.3.0 -> 1.4.3

- 2019-05-22, jgliss Updated README
- 2019-05-22, jgliss Updated README
- 2019-05-22, jgliss Updated README
- 2019-05-22, jgliss Updated README
- 2019-05-22, jgliss Updated README
- 2019-05-22, jgliss Removed release note for version 1.3.0
- 2019-05-21, jgliss Fixed some formatting errors in README
- 2019-05-21, jgliss Updated version to 1.4.3, cleaned up and reformatted README so that it passes twine check and included Python 3 support into classifiers in setup.py
- 2019-05-21, jgliss Updated version to 1.4.2
- 2019-05-21, jgliss Removed / reformatted problematic parts from README.rst, so that it will be (hopefully) rendered properly in PyPi
- 2019-05-21, jgliss Removed pyplis logo from README (since it could not be rendered on PyPi) and updated version to 1.4.1
- 2019-05-21, jgliss Changed version to 1.4.0 (ready for release)
- 2019-05-21, jgliss Added LICENSE file to MANIFEST.in

- 2019-05-21, jgliss Refined tests performed in script ex06_doas_calib.py and accounted for minor differences in results between python 2.7 and 3.7
- 2019-05-21, jgliss Minor changes in output of radius search in doascalib.py
- 2019-05-21, jgliss Updated requirements file and added explicit conda environment file for python 2.7
- 2019-05-20, jgliss Updated README
- 2019-05-20, jgliss Updated README
- 2019-05-20, jgliss Added numpy and six to environment file
- 2019-05-20, jgliss Updated version
- 2019-05-20, jgliss Added pydoas to conda-forge dependency in pyplis_env.yml (no pip dependencies anymore)
- 2019-05-20, jgliss Added import of mpl_toolkits.mplot3d in 3d plotting routine of MeasGeometry
- 2019-05-20, jgliss Updated, simplified and merged two conda environment files for python 2 and 3 into new one that works for both: pyplis_env.yml
- 2019-05-20, jgliss Updated version and fixed one test in ex02_meas_geometry.py
- 2019-05-20, jgliss Merge branch 'py3'
- 2019-05-12, jgliss Fixed some minor Deprecationwarnings
- 2019-05-12, jgliss Fixed failing test related to viewing direction correction of camera
- 2019-05-12, jgliss Fixed import error in geometry.py of geonum class TopoAccessError, due to recent API changes in geonum
- 2019-05-12, Jonas Merge pull request #23 from johannjacobsohn/fix-typos
- 2019-05-12, jgliss Fixed relative imports of geonum in example scripts; fixed failing test in ex06_doas_calib.py
- 2019-05-12, jgliss Added conda environment file for python 3 installation (not yet tested)
- 2019-04-23, Jonas Gliss Updated pyplis_env_py27.yml
- 2019-04-23, Jonas Gliss Updated README
- 2019-04-23, Jonas Gliss Updated README
- 2019-04-23, Jonas Gliss Updated README
- 2019-04-23, Jonas Gliss Updated VERSION to 1.4.0.dev1
- 2019-04-23, Jonas Gliss Updated installation instructions in README
- 2019-04-23, Jonas Gliss Updated script ex02_meas_geometry.py due to failing tests
- 2019-04-23, Jonas Gliss Added python 2.7 conda environment file pyplis_env_py27.yml
- 2019-04-22, jgliss Merged and updated README from py3 branch
- 2019-04-22, jgliss Updated geometry.py due to recent API updates in new geonum release v1.4.0
- 2019-04-22, jgliss Updated example scripts due to recent API changes in latest version 1.4.0 of geonum
- 2019-03-01, Jonas Update README.rst
- 2019-02-25, Johann Jacobsohn cleanup
- 2019-02-22, Johann Jacobsohn fixup use of depreciated methods and properties
- 2019-02-19, Johann Jacobsohn fixup typos
- 2019-02-19, Solvejg Dinger Fix broken link in Readme

- 2019-02-19, Solvejg Dinger Merge pull request #10 from heliotropium72/patch-1
- 2019-02-19, Solvejg Dinger Adapt suggestion for one-liner
- 2019-02-11, Jonas Gliss Refers to #13: Added new default camera usgs_9mm; the 12mm focal length version can now be accessed via the former ID usgs but also via usgs_12mm
- 2019-01-25, Solvejg Dinger Fixed rendering problem of README
- 2019-01-25, Solvejg Dinger Update README.rst
- 2019-01-22, Solvejg Dinger Merge pull request #16 from johannjacobsohn/py3
- 2019-01-17, Johann Jacobsohn WIP: fixup to make tests pass, needs to be investigated
- 2019-01-17, Johann Jacobsohn py3.7 doesn't like __dir__ to be overwritten…
- 2019-01-17, Johann Jacobsohn add python 3.5 and 3.7 to tox, update pytest dependency and remove sitepackages from tox to make tests more consistent
- 2019-01-15, heliotropium72 Fix pytest fixtures to correct usage
- 2019-01-14, heliotropium72 Revert previous WIP
- 2019-01-02, jgliss Partly fixed failing tests
- 2018-12-27, jgliss Fixed bug arising from division returning float rather than int, due to recent Python 3 port
- 2018-11-29, Johann Jacobsohn Revert "skip failing test until fixed"
- 2018-11-29, Johann Jacobsohn WIP to pass tox scripts - needs to be reverted and fixed!
- 2018-11-29, Johann Jacobsohn add RUN_EXAMPLE_SCRIPTS.py to tox, use sitepackages for Basemap
- 2018-11-29, Johann Jacobsohn fix floordiv issue from python3 work, and cleanup
- 2018-11-28, Johann Jacobsohn protect possible None value against < operator, which fails in python3. Remove try..except block that only obscures exceptions
- 2018-11-28, Johann Jacobsohn remove __deepcopy__, underlying issue needs to be fixed in geonum
- 2018-11-28, Johann Jacobsohn make sure to use int for array indices
- 2018-11-28, Johann Jacobsohn numpy.sum does not handle .values() as you would expect in python3…
- 2018-11-28, Johann Jacobsohn truediv -> __truediv__, fix typo
- 2018-11-27, Johann Jacobsohn fix reading of cam_info.txt in python3
- 2018-11-27, Johann Jacobsohn try to load test data from envvar PYPLIS_DATADIR if not found in default location (helpful for testing)
- 2018-11-27, Johann Jacobsohn fixup: CalibData.senscorr_mask does actually not accecpt ndarray
- 2018-11-23, Johann Jacobsohn add RUN_INTRO_SCRIPTS.py to tox
- 2018-11-16, Johann Jacobsohn fix pytest to version 3
- 2018-11-16, Johann Jacobsohn fix difference in exception handling between python 2 and 3, restrict exceptions to KeyError
- 2018-11-14, Johann Jacobsohn skip failing test until fixed
- 2018-11-14, Johann Jacobsohn fixup requirements and tox environment
- 2018-11-14, Johann Jacobsohn fixup import syntax for py3
- 2018-11-14, Johann Jacobsohn fixup deepcopy of MeasGeometry in py3
- 2018-11-14, Johann Jacobsohn fix test_io in py3

- 2018-11-14, Johann Jacobsohn fix urlib import for python3
- 2018-11-14, Johann Jacobsohn small pydocstyle fixes
- 2018-11-14, Johann Jacobsohn __div__ -> truediv
- 2018-11-14, Johann Jacobsohn return value of .keys() and .values() have changed between python 2 and 3, this should make it consistent between the two
- 2018-11-14, Johann Jacobsohn add __future__ to help make python 2 and 3 behave consistent
- 2018-11-09, Johann Jacobsohn add pylint and disable every offending check
- 2018-11-05, Johann Jacobsohn scripts/ex07_doas_cell_calib.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/utils.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/setupclasses.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/processing.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/plumespeed.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/plumebackground.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/model_functions.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/inout.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/image.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/imagelists.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/geometry.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/forms.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/fluxcalc.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/doascalib.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/dilutioncorr.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/dataset.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/cellcalib.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn pyplis/calib_base.py: improve python3 compability, fix bugbear lint
- 2018-11-05, Johann Jacobsohn add bugbear lint to improve python 3 compability
- 2018-11-05, Johann Jacobsohn Light phrasing changes to conform to PEP 257 and pass pydocstyle validation
- 2018-06-01, Johann Jacobsohn lint pyplis/test/test_dataset_module.py
- 2018-05-25, Johann Jacobsohn lint scripts/more_scripts/ex002_plume_background_combi.py
- 2018-05-25, Johann Jacobsohn lint scripts/more_scripts/ex001_save_all_calib_imgs.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex12_emission_rate.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex11_signal_dilution.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex10_bg_imglists.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex0_7_cellcalib_manual.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex0_6_pcs_lines.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex0_5_optflow_livecam.py

- 2018-05-25, Johann Jacobsohn lint scripts/ex0_4_imglists_auto.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex0_3_imglists_manually.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex0_2_camera_setup.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex0_1_img_handling.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex09_velo_optflow.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex08_velo_crosscorr.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex07_doas_cell_calib.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex06_doas_calib.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex05_cell_calib_auto.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex04_prep_aa_imglist.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex03_plume_background.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex02_meas_geometry.py
- 2018-05-25, Johann Jacobsohn lint scripts/ex01_analysis_setup.py
- 2018-05-25, Johann Jacobsohn lint scripts/SETTINGS.py
- 2018-05-25, Johann Jacobsohn lint scripts/RUN_INTRO_SCRIPTS.py
- 2018-05-25, Johann Jacobsohn lint scripts/RUN_EXAMPLE_SCRIPTS.py
- 2018-05-25, Johann Jacobsohn lint pyplis/utils.py
- 2018-05-25, Johann Jacobsohn lint pyplis/test/test_io.py
- 2018-05-25, Johann Jacobsohn lint pyplis/test/test_image_module.py
- 2018-05-25, Johann Jacobsohn lint pyplis/test/test_highlevel_examples.py
- 2018-05-25, Johann Jacobsohn lint pyplis/test/test_dataset_module.py
- 2018-05-25, Johann Jacobsohn lint pyplis/test/__init__.py
- 2018-05-25, Johann Jacobsohn lint pyplis/setupclasses.py
- 2018-05-25, Johann Jacobsohn lint pyplis/processing.py
- 2018-05-25, Johann Jacobsohn lint pyplis/plumespeed.py
- 2018-05-25, Johann Jacobsohn lint pyplis/plumebackground.py
- 2018-05-25, Johann Jacobsohn lint pyplis/optimisation.py
- 2018-05-25, Johann Jacobsohn lint pyplis/model_functions.py
- 2018-05-25, Johann Jacobsohn lint pyplis/inout.py
- 2018-05-25, Johann Jacobsohn lint pyplis/imagelists.py
- 2018-05-25, Johann Jacobsohn lint pyplis/image.py
- 2018-05-25, Johann Jacobsohn lint pyplis/helpers.py
- 2018-05-25, Johann Jacobsohn lint pyplis/glob.py
- 2018-05-25, Johann Jacobsohn lint pyplis/geometry.py
- 2018-05-25, Johann Jacobsohn lint pyplis/forms.py
- 2018-05-25, Johann Jacobsohn lint pyplis/fluxcalc.py

- 2018-05-25, Johann Jacobsohn lint pyplis/exceptions.py
- 2018-05-25, Johann Jacobsohn lint pyplis/doascalib.py
- 2018-05-25, Johann Jacobsohn lint pyplis/dilutioncorr.py
- 2018-05-25, Johann Jacobsohn lint pyplis/dataset.py
- 2018-05-25, Johann Jacobsohn lint pyplis/custom_image_import.py
- 2018-05-25, Johann Jacobsohn lint pyplis/cellcalib.py
- 2018-05-25, Johann Jacobsohn lint pyplis/calib_base.py
- 2018-11-01, Solvejg Dinger Bug Fix: Setting of calibration coefficients
- 2018-05-25, Johann Jacobsohn lint pyplis/__init__.py
- 2018-05-25, Johann Jacobsohn lint scripts/priv_01_plot_raw_vs_calib.py
- 2018-05-24, Johann Jacobsohn add python 3 to tox
- 2018-05-17, Johann Jacobsohn add tox to run tests
- 2018-09-09, jgliss Merge branch 'heliotropium72-multi_layer_fits'
- 2018-09-09, jgliss fixed minor bug (download location of test data)#
- 2018-09-09, jgliss Fixed minor bug
- 2018-09-09, jgliss Merge branch 'multi_layer_fits' of https://github.com/heliotropium72/pyplis into heliotropium72-multi_layer_fits
- 2018-06-22, heliotropium72 Fix binary_mask fixture
- 2018-06-21, heliotropium72 Add sky_mask to image lists
- 2018-06-21, heliotropium72 Add class ImgListLayered
- 2018-06-21, heliotropium72 Make *OptflowFarneback.calc_flow* compatible with OpenCV 2.x and 3.x
- 2018-06-20, Jonas Merge pull request #8 from heliotropium72/patch-3
- 2018-06-20, Jonas Merge pull request #7 from heliotropium72/patch-2
- 2018-06-19, heliotropium72 Add get_masked_img method and corresponding tests
- 2018-06-18, Solvejg Dinger Merge pull request #11 from heliotropium72/patch-3
- 2018-06-18, Solvejg Dinger Fix indentation level
- 2018-06-18, Solvejg Dinger Merge pull request #10 from heliotropium72/patch-3
- 2018-06-18, Solvejg Dinger Merge pull request #9 from heliotropium72/patch-2
- 2018-06-07, Solvejg Dinger Add custom import method for Comtessa image data
- 2018-06-07, Solvejg Dinger Extended image.meta dictionary
- 2018-05-27, jgliss Fixed minor bug; added custom image read method template for QSI camera LMV group, France
- 2018-05-26, jgliss Updated formatting
- 2018-05-26, jgliss Updated formatting
- 2018-05-26, jgliss Deleted history of old binary files (repo size down to 30MB from ~500MB); Added content of scripts_out folder to .gitignore; updated README
- 2018-05-23, jgliss Removed strict PIL dependency for image read (now using opencv method imread)

7.2 Release 1.0.0 -> 1.0.1

This release includes only minor changes compared to the last one. These are mainly related to the access and handling / modelling of the plume background intensities in `ImgList` objects.

7.2.1 25/11/2017 - 12/01/2018 (v1.0.0 -> v1.0.1)

1. Fixed some bugs
2. Added more tests.
3. Improved access of plume background images in `ImgList` objects

7.3 Release 1.0.1 -> 1.3.0

Note: This release includes major API changes, performance improvements and bug fixes compared to version 1.0.1. Please update your installation as soon as possible.

7.3.1 Summary

Measurement geometry (`MeasGeometry`):

- more accurate plume distance retrievals (i.e. now also in dependency of vertical distance).
- redesigned API -> improved user-friendliness.

Image analysis: Image registration shift can now be applied to images.

- `shift()` in class Img.
- Comes with new *mode* (`shift_mode`) in `ImgList` objects.
- Default on / off shift for camera can be set in `Camera` using attribute `reg_shift_off` (and correspondingly, in file *cam_info.txt*).

Camera calibration. Major improvements and API changes:

- new abstraction layer (`calib_base`) including new calibration base class. `CalibData`: both `DoasCalibData` and `CellCalibData` are now inherited from new base class `CalibData`. Advantages and new features:
  - arbitrary definition of calibration fit function.
  - fitting of calibration curve, I/O (read / write FITS) and visualisation of DOAS and cell calibration data are now unified in `CalibData`.

Further changes

- `ImgStack` more intuitive and flexible (e.g. dynamically expandable).
- Improved index handling and performance of image list objects (`imagelists`).
- `PlumeBackgroundModel`: revision, clean up and performance improvements.
- Improved user-friendliness and performance of plume background retrieval in `ImgList` objects.
- Correction for signal dilution (`DilutionCorr`): increased flexibility and user-friendliness.

- Improved flexibility for image import using `Dataset` class (e.g. on / off images can be stored in the same file).
- Reviewed and largely improved performance of general workflow (i.e. iteration over instances of `ImgList` in `calib_mode`, `dilcorr_mode` and `optflow_mode`).

Major bug fixes

- Fixed conceptual error in cross-correlation algorithm for velocity retrieval (`find_signal_correlation()` in module `plumespeed`).
- Fixed: `ImgList` in AA mode used current off-band image (at index `idx_off`) both for the current and next on-band image (and not `idx_off+1`).

7.3.2 1.0.1 -> 1.1.0

1. `Img` object

- Included read / write of meta info dictionary for FITS load / save
- New method `is_darkcorr()`

2. DOAS calibration (`doascalib`)

- More flexible retrieval of DOAS calibration curves

3. `ImgStack`

- new method `ImgStack.sum()`
- Can now be dynamically extended (i.e. dynamic update of 3D array size). Corresponding API changes:
  - REMOVED: method `append_img()`
  - NEW methods: `insert_img()`, `add_img()`, `init_stack_array()`

4. Measurement geometry (`MeasGeometry`)

- More accurate plume distance retrieval (now also in vertical direction, cf. Fig. 2 from example script 2)

5. Other changes

- Moved Etna test data to new URL
- Fixed bugs

7.3.3 1.1.0 -> 1.2.1

1. `Img` object

- new method `Img.sum()`

2. Image list classes (`imagelists`, MAJOR API CHANGES)

- Improved flexibility and clarity in index management
- New attribute `skip_files` (i.e. load only every nth image from the filelist)
- New method `iter_indices()`
- Renamed method `update_index_linked_lists()` to `change_index_linked_lists()`
- Removed method `change_index()`

3. Plume background retrieval (`plumebackground`)

- Getter / setter for attr. `surface_fit_mask` (ensure it is type `Img`)

4. Changes related to I/O

- Moved option `` `LINK_OFF_TO_ON `` from `Dataset` to `BaseSetup` (no API changes in `Dataset`)
- New I/O option `ON_OFF_SAME_FILE` in `BaseSetup` that can be set if on and off images are stored in one (e.g. FITS) file (like for the new USGS CVO camera type)
- I/O options for data import can now be specified in file *cam_info.txt* for each camera individually using keyword `io_opts` and is stored as dict in `CameraBaseInfo` (base class of `Camera`)
- Included I/O info for camera of USGS CVO (uses previous point)
- Source info can now be saved automatically to file *my_sources.txt*

5. Other changes

- New method `matlab_datenum_to_datetime()` in `helpers`
- Fixed bugs

7.3.4 1.2.1 -> 1.3.0

Note: This version includes major refactoring and changes in API, aiming for more transparency and intuitive design. For instance, both the `DoasCalibData` and `CellCalibData` now inherit from a new base class `CalibData` (in new module `calib_base`).

Note: Changes related to camera calibration API (e.g. renaming, refactoring or removing of methods) are not resolved in full detail below (following point 1.).

1. Camera calibration

- NEW MODULE `calib_base` containing new calibration base class `CalibData` (both `DoasCalibData` and `CellCalibData` inherit from this base object)
- MAIN CHANGES associated with with refactoring into general base class `CalibData`
  - NEW FEATURE: Fit function for calibration data (both cell and DOAS) can now be defined arbitrarily (before, only polynomials were possible). See also module `model_functions`, in particular new class `CalibFuns`
  - I/O (.e.g to / from FITS, or csv) are now unified for cell and DOAS calibration
  - Visualisation (e.g. plot of calibration curve and data) now unified for cell and DOAS calibration
  - New default fit function based on Kern et al. 2015
  - UNCERTAINTY treatment: Error in calibrated CDs is now computed based on the standard deviation of fit residual (if more than 10 datapoints are available for retrieval of calibration curve).

2. `Img` object

- Renamed attribute “alt_offset” -> “altitude_offset”
- Moved custom import for ECII camera into new custom method `load_ecII_fits()` in module `custom_image_import`

- New attributes:
  - is_cropped
  - is_resized
  - is_shifted
- New methods:
  - shift() (applies x/y pixel shift of image)
  - convolve_with_mask(), for instance, when applied to an AA image, the input mask may be, e.g. a parameterised DOAS FOV (e.g. fitted 2D super-Gauss). The function then returns the weighted average AA within the FOV.
  - get_thresh_mask()

3. Image list classes (imagelists)

- **New list mode** shift_mode (only for offband lists, i.e. lists with attribute type="off"): activate / deactivate shift (dx, dy) of images on image load (cf. other list modes, such as tau_mode, calib_mode, optflow_mode). If activated, the default shift reg_shift_off of the assigned Camera instance is used (is set (0, 0) if not explicitly defined (either in file *cam_info.txt* for a camera type (cf. cam "usgs" therein) or in instance of Camera directly).
- **Reviewed and optimised:**
  - correct_dilution() reviewed, largely rewritten and optimised
- **New attributes:**
  - update_cam_geodata (default is False). If True, the measurement geometry (i.e. plume distance) is automatically updated if image files contain camera geodata (e.g. lat, lon, viewing direction).
- **New methods:**
  - calc_plumepix_mask() (for dilution correction)
  - timestamp_to_index() (returns list index corresponding to a datetime object)
  - _iter_num() (number of iterations to loop through the whole list, resulting from the total number of files nof and skip_files)
- pop() now raises *NotImplementedError*
- Introduced @property methods (getter / setter) for the attributes skip_files (newly introduced in v1.2.1, see above) and edit_active to ensure index update and reload (on change)
- Further changes, deprecated, renamed
  - Introduced new input parameter *reload_here* in goto_img() (if True, load() is called even if the new index is the same as the current index, defaults to False)
  - **Deprecated**:
    - Removed attribute which_bg (now handled automatically by @property attribute bg_img)
  - **Renamed**
    - aa_corr_mask -> senscorr_mask
    - DARK_CORR_OPT -> darkcorr_opt
- Bug fixes:
  - Fixed: on-band list AA mode used current off-band image (at index idx_off) both for the current and next on-band image (and not idx_off+1).

4. Measurement geometry (`MeasGeometry`, MAJOR API CHANGES)

- Improved user-friendliness and performance: getter / setter methods for all attributes
  - Intended access / modification of attributes is via new getter / setter methods (e.g. `` `geom.cam["lon"] `` -> `geom.cam_lon`)
  - Comes with better handling of recomputation requirements of geometry in case individual parameters (e.g. camera viewing direction, position, wind direction) are updated (in this context, note new attribute `update_cam_geodata` in `ImgList` objects). Specifically:
  - Method `update_geosetup()` is called whenever a relevant attribute is updated via the corresponding setter method. This ensures, that derived values such as plume distance are always up-to-date with the current attributes.
  - Attribute dictionaries now private (e.g. `.cam` -> `._cam`, `.source` -> `._source`).
- New methods:
  - `get_topo_distance_pix()` (determines distance to local topography in viewing direction of individual image pixel)

5. `PlumeBackgroundModel` (Review and clean-up)

- New attribute `last_tau_image`
- New method `_init_bgsurf_mask()`: initiate mask for 2D background polynomial surface fit (only relevant for correction mode `mode=0`)
- **Removed**
  - dictionary `_current_imgs`: kept copies of input images (private dictionary)
  - Methods: `get_current()`, `pyrlevel()`, `current_plume_background()`, `subtract_tau_offset()`, `_prep_img_type()`, `set_current_images()`, `plot_tau_result_old()`

6. `DilutionCorr`

- Retrieval of extinction coefficients for dilution correction based on dark terrain features can now also be performed for individual pixel coordinates in the images, in addition to the distance retrieval based on lines in the images (see example script 11)
- New methods:
  - `add_retrieval_point()`
  - `add_retrieval_line()`

7. Module `model_functions`:

- New calibration fit function(s) based on Kern et al., 2015
- New class `CalibFuns` for access of calibration fit functions

8. Plume velocity retrievals (`plumespeed.py`)

- Cross correlation method (`find_signal_correlation()`)
  - Improved retrieval robustness: introduced percentage max shift that describes the maximum shift in percent of the second relative to the first time-series based on the total length of both series.
  - Fixed systematic retrieval error: Before, the second signal was rolled over the first, meaning, that the "end" of the 2. signal was attached to it's beginning and thus, correlated with the beginning of the first signal. That behaviour has been resolved.
- Optical flow (`OptflowFarneback` and `FarnebackSettings`)

  - i_min (lower end of contrast range for optical flow calculation) can now also be smaller than 0.

9. I/O and setup classes (modules inout and setupclasses)

- **my_pyplis** folder is now created on installation (in user home directory)
  - includes copies of *cam_info.txt* file and *my_sources.txt*
- New method save_default_source() in inout (is saved in file *my_sources.txt*)
- New method save_to_database() in Source (wrapper method for save_default_source())
- New I/O option REG_SHIFT_OFF in classes BaseSetup and MeasSetup: if True (and if image lists are created using Dataset and corresponding MeasSetup object), then, the off-band images (in off-band ImgList) are automatically shifted to on-band images (in on-band ImgList) using the registration shift that is specified in Camera.reg_shift_off (can be set in file *cam_info.txt*)

10. Other changes

- New method integrate_profile() in class LineOnImage
- New method make_circular_mask() in module helpers.py
- In fluxcalc (and all included classes): renamed attr cd_err_rel to cd_err (note changes in uncertainty treatment of calibration data!)
- EmissionRateSettings: new option / attribute min_cd_flow (in addition to already existing min_cd) that may be used to explicitly define the minimum column-density of an image pixel for it to be considered valid with respect to optical flow histogram analysis (before, the threshold min_cd was used). Is set equal min_cd if not explicitly specified
- Moved class LineOnImage into module utils
- Moved method model_dark_image() from processing to image as well as class ProfileTimeSeriesImg
- Changed input parameter of model_dark_image() in processing
- Changed default colormap for optical density (and calibrated) images from *bwr* to *viridis* (perceptually uniform)
- **Major performance improvements**: reviewed typical workflow chain and removed irrelevant duplications of image arrays in certain objects ()
- Fixed bugs
- Included new tests (test suite still very low coverage…)

7.4 Release 0.11.2 -> 1.0.0

7.4.1 30-31/03/2017

1. Renamed roi_rad in Farneback classes to roi_rad_abs (makes it clearer in which coordinates it is supposed to be defined). The old name also still works but a warning is given if used.
2. Renamed hist_dir_sigma in Farneback classes to hist_sigma_tol since it is applied both to the fit result of the main peak of the orientation histogram but also to determine the uncertainty in the length histogram from the moments analysis. The old name also still works but a warning is given if used.
3. More features in LineOnImage

- Global velocity estimates (and uncertainties) can now be assigned (e.g. for emission rate analysis)

- `LocalPlumeProperties` can now be assigned (e.g. for emission rate analysis and if a time series of local displacement vectors for velocity retrieval was calculated beforehand and the results are accessible in a text file)).

7.4.2 01-07/04/2017

1. Added features in `LocalPlumeProperties`

- Interpolation can now also be performed onto index array of other time series (e.g. image list time stamps)
- New method `apply_significance_thresh()` in `LocalPlumeProperties`: can be used to remove data points with significance lower than provided thresh (and combined e.g. with interpolation).
- New method `to_pyrlevel()`: converts displacement lengths (and errors) to a given Gauss pyramid level.

2. Expanded functionality in classes `EmissionRateAnalysis` and `EmissionRateSettings`:

- global velocities can now be assigned for each PCS line individually (need to be assigned in the `LineOnImage` objects directly).
- `LocalPlumeProperties` assigned to retrieval lines (`LineOnImage` objects) in `EmissionRateAnalysis` are now considered for analysis when using `velo_mode_farneback_histo` (e.g. they might be calculated and post processed beforehand).
- New velocity method `farneback_hybrid` for velocity retrievals: uses optical flow output along line modified such that vectors that are not in expectation range (retrieved from histo analysis) are replaced by the average flow vector from the histo analysis.

3. New attribute `residual` in `DoasCalibData`
4. Fixed some bugs related to scale space conversion in `ImgList` objects (e.g. related to activation of `tau_mode`, dilution correction)
5. Corrected bug related to SO2-uncertainty based on slope error of calibration curve from covariance matrix of poly fit. Previously: used value of slope error as measure of uncertainty (wrong), now: use relative error, e.g. calibration curve zero y-axis offset and with slope, slope err: `m=1e19, m_err=1e17` then the mapped SO2 error (for a given tau value `tau0`) is determined as :`so2 = tau0 * m` and `so2_err = so2 * m_err / m`
6. Added mathematical operators to `EmissionRateResults` class

- __add__: use "+" operator to add results (e.g. retrieved at two different lines from two crater emissions)
- __sub__: use "-" operator to subtract results (e.g. retrieved at two different positions downwind of the crater emissions)

7.4.3 10/04/2017

1. Added option in `make_stack()` in `ImgList` objects: the method includes now the option to specify a reference ROI in the image (e.g. sky reference area) and a corresponding min / max range for the expectation value in that range: if the input is specified, then only images are added to the stack that are within the specified range within the ROI.
2. New features in `EmissionRateAnalysis` and `EmissionRateSettings`

- Added same feature (as described in 1.) to emission rate retrieval classes, relevant attributes in `EmissionRateSettings` class are:
  - `ref_check_mode`: activate / deactivate the new mode
  - `bg_roi_abs` (ROI used for check)

- ref_check_lower_lim: lower intensity limit
- ref_check_upper_lim: upper intensity limit

- Moved attr. bg_roi from analysis class to settings class and renamed to bg_roi_abs.

7.4.4 11/04/2017

1. Added check of date information in get_img_meta_all_filenames() of ImgList which is, for instance, used for accessing datetime information of acq. times of all images in the list: a problem may occur if the file names only include information of acq. times of the images but not dates. Then, the retrieved timestamps (numpy array of datetime objects) will only include acq. times of all images and the default date: 1/1/1900. If this is the case, then the method replaces these default dates in the array using the date stored in the meta header of the currently loaded image in the the list. This is, for instance relevant for the HD default camera which includes date information in the tiff header (will be loaded and stored in meta header of Img class on load, but not in the file names).

7.4.5 12/04 - 04/05/2017

1. Minor changes in plot style for standard outputs
2. Worked on docs

7.4.6 04/05 - 21/05/2017 (v0.11.4 -> v0.12.0)

Note: Not downwards compatible change in fluxcalc.py: changed name of velocity retrieval modes and functions related to optical flow from e.g. farneback_hybrid to flow_hybrid.

1. Minor improvements in documentation of example scripts
2. Changes in docs
3. Minor changes in plot style for standard outputs
4. DOAS calibration polynomial is now fitted only using mantissa of the CDs (to avoid large number warning in polyfit)
5. Changes in optimisation strategy for optical flow histogram analysis and correction (modules: plumespeed.py, fluxcalc.py)

1. Minimum required length (per line and image is set at lower end of 1sigma of expectation interval of histo analysis
2. More sophisticated uncertainty analysis for effective velocities

1. Changed all names in fluxcalc.py related to optical flow based velocity retrievals which included farneback to flow (not downward compatible)
2. New class EmissionRateRatio in fluxcalc

7.4.7 22/05 - 29/08/2017 (v0.11.4 -> v0.12.0)

1. Minor bug fixes
2. Added functionality to Img objects

3. DOAS calibration data can now be fitted using weighted regression based on DOAS fit errors. Note, that new default is weighted fitting, if applicable (i.e. if uncertainties are available).
4. New class `VeloCrossCorrEngine` in `plumespeed.py` for high level computing of cross correlation based velocity retrievals. Note that this includes changes in example script 8, which now uses the new class. Thus, running the current version of example script 8 will not work with older versions of Pyplis.
5. Started with implementation of test suite using pytest

7.4.8 30/08 - 05/10/2017 (v0.12.0 -> v0.13.4)

1. Minor bug fixes
2. Improved convenience functionality of classes in `doascalib` by adding some @property decorators.
3. New high-level default method `run_fov_fine_search()` in `DoasFOVEngine`
4. Renamed key vor wind velocity (and error) in `MeasGeometry` from "vel" to "velo"
5. New method `find_movement()` in `plumespeed`. The method performs an iterative computation of the optical flow between two images under variation of the considered input brightness ranges.
6. Improved functionality for automated retrieval of sky-background pixels in an plume image (now uses new method `find_movement()` to identify and exclude pixels showing motion.

7.4.9 5/10/2017 - 25/11/2017 (v0.13.4 -> v1.0.0)

1. Fixed some bugs
2. Started with setting up a test-suite (available in the GitHub repo but not yet included in standard installation of the code)
3. Added test-dataset of size reduced images from the Etna testdat (mainly for tests. This dataset is not yet included in the standard installation

Automatic SRTM access can now be deactivated in `MeasGeometry` objects

1. Made MultiGaussFit optional for histogram post analysis of optical flow
2. Removed requirement for `progressbar`
3. Changed colour and plot styles in some of the standard plotting methods (e.g. cross-correlation velocity)
4. Improvements and new methods in `CellCalibData` objects (e.g. fitting of calibration curve, access to co-variance matrix, slope error, calculation of uncertainties).
5. Renamed some methods
6. Improvements in efficiency and new methods in `MeasGeometry` objects.

New methods in `helpers.py`

1. Minor changes to example scripts
2. Major changes to `ImgList` objects

1. New list mode: `dilution_corr`: images are loaded as dilution corrected images using the method from Campion et al., 2015. Can be activated and deactivated like all other modes (e.g. `tau_mode`).
2. Updated all list methods related to signal dilution correction.
3. @property decorators (and setters) for plume distance and integration step length, i.e. `plume_dists` and `integration_step_length`

4. Renamed `next_img()` and `prev_img()` to `goto_next()` and `goto_prev()` respectively (old names still work as well)

1. Changes to `DoasFOV`: `fov_mask` is now called `fov_mask_rel`. Renamed `transform_fov_mask_abs_coords()` to `fov_mask_abs()`.
2. `EmissionRateAnalysis` can now also be run with setting `dilcorr` using the new `dilcorr_mode` of `ImgList` objects (see above and example script 12).
3. Some new features in class `Img` (e.g. `avg_in_roi()`, or `erode()`).

7.5 Release 0.9.2 -> 0.11.2

7.5.1 28/02/2017 - 01/03/2017

1. Allowing for defining custom image import method (file custom_image_import.py)
2. Fixed bug regarding assignment of dark / offset lists for HD-Custom camera: if multiple type dark (and / or offset) image lists exist (based on camera file naming convention after file separation in Dataset), then the list with the shortest difference in the exposure time is set (using the first image of the respective image lists for intercomparison)
3. Expanded handling of start / stop time specifications in Setup classes and Dataset classes (for initialisation of working environment) -> now, the user can also provide time stamps (datetime.time objects) or dates (datetime.date objects) as input and they will converted automatically to datetime. Also, if the default date is set (i.e. 1/1/1900) in a Setup, it will be disregarded and only the time stamps HHMMSS are considered to identify image files belonging to specified start /stop interval.
4. Minor convenience changes in Dataeset enabling to set attributes and options of MeasSetup class directly from Dataset class using @property decorators (e.g. for start, stop, base_dir, etc.)
5. Updated specs for HD-Custom camera such that cell calibration data can be imported
6. Expanded functionality for dark and offset list assignments in Dataset and ImgList objects.

1. Master dark / offset images are now searched for each dark / offset image list individually
2. Customised assignment of dark / offset lists in image lists for cameras where meas type is specified in own filename substring (e.g. HD cam).

7.5.2 02/03/2017

1. Included new default camera type "hd_new" (2. camera from group in Heidelberg, Germany). Currently missing detector and optics specs
2. Expanded flexibility for meta information access via filename for acquisition time, meas_type and filter_id in Camera class: now, the conversion / identification strings can also include the actual delimiter (e.g. delim="_", time_info_pos - 0, time_info_str="%Y%m%d_%H%M%S_%f" or filter_id_pos=3 and filter.acronym="A_dark"). This is for instance required for file naming convention of new default SO2 camera type "hd_new".
3. Improved functionality for dark and offset image access in ImgList classes
4. Improved data import speed in Dataset -> search of master_dark image is only applied to lists that actually include image data

7.5.3 03/03/2017

1. Included image check for negative numbers or zeros after dark image correction and before tau / AA image calculation: correction is directly applied to images (no warning), i.e. pixels <- 0 are set to smallest positive float of system.
2. Removed bugs regarding image time stamps in MeasSetup and image match search in Dataset (when specifying start / stop time stamps are provided as time object and not as datetime object). These two bugs resulted from changes applied in 0.9.3.dev1 (1/3/2017) and are irrelevant for previous versions.

7.5.4 05/03/2017

Note: Detected and fixed bug related to signal cross correlation based plume velocity retrievals after pandas updgrade from 0.16.2 -> 0.19.2.

7.5.5 06/03/2017

1. Removed bug in `ImgStack` method `merge_with_time_series`: generalised catch of first exception (applies if `doas_series` is pandas `Series` object and not pydoas `DoasResults`).

7.5.6 07/03/2017

1. Improved performance in long "for" loops (e.g. `make_stack()`, `get_mean_value()` in `BaseImgList` or file searching methods in `Dataset` by removing `self.` operations within the loops)
2. `EmissionRateResults` can now be saved as text file and has some new methods, the most important ones are:

- `__str__()`: informative string representation
- `to_dict()`: converts results to dictionary
- `to_pandas_dataframe()`: converts object into pandas `DataFrame` class
- `from_pandas_dataframe()`: imports data from pandas `DataFrame` class
- `save_txt()`: save results as text file

1. Updated options for xlabel formatting when plotting time series
2. Improved optical flow histogram analysis

- Renamed settings param `sigma_tol_mean_dir` to `hist_dir_sigma`
- New: choose from two options for retrieval of average displacement length from length histogram (in `get_main_flow_field_params()` of `OpticalFlowFarneback`):
  - "argmax": uses bin with largest count as mean displacement estimate (new)
  - "multigauss": tries to perform `MultiGaussFit` to data and if this fails, uses method "argmax"
- new global settings parameters for maximum number of fitted gaussians in both orientation and length histogram, can now be set via `OpticalFlowFarnebackSettings`

7.5.7 08/03/2017

1. New functions in `Img` class:

- `to_binary` and corresponding entry `is_bin` in `edit_log`` dict.
- `dilate`: apply morphological transform *dilation* to image using method `cv2.dilate`
- `invert`: inverts an image object (added entry in edit_log)

1. New method `get_mean_img` in image list classes: determines average image based on start / stop index (or for all images in list)
2. Removed bug in `Img` method `to_pyrlevel` for going up in pyramid

7.5.8 09/03/2017

1. Class `Dataset` objects can now be initiated with diffent `ImgList` types (e.g. `CellCalibEngine` is now initiated with `CellImgList` objects)
2. implementation of method to apply dilution correction to an plume image `correct_img` moved as global method in `dilutioncorr.py` and is now a wrapper in `DilutionCorr` class.
3. New methods in `DilutionCorr` class:

- `get_ext_coeffs_imglist`: retrieve extinction coefficients for all images in an `ImgList` object.

7.5.9 13/03/2017

1. New functions in `ImgList`:
   - `get_thresh_mask()`: get mask based on intensity threshold (e.g. tau thresh)
   - `prepare_bg_fit_mask()`: (BETA) for background modelling mode 0 (PolySurfaceFit). Determines a mask specifying background pixels based on intensities in reference rectangles and optical flow analysis (slow).
   - `correct_dilution()`: correct current image for signal dilution
   - `set_bg_img_from_polyfit()`: determines an *initial* background image in list using `PolySurfaceFit`. The result is set as new ``bg_img` attribute, i.e. is used for backrgound modelling in modes 1-6. This can be done if no measured sky radiance image is available.
   - `correct_dilution()`: applies Dilution correction to current image if all requirements are fulfilled for that
   - start / stop indices can now be set in `make_stack()`
2. Removed automatic load of previous image in `ImgList` objects
3. Included AA image calculation for CORR_MODE – 0 in `PlumeBackgroundModel`.
4. Removed dark corr check between plume and BG image in `PlumeBackgroundModel` when modelling tau images.

7.5.10 14/03/2017

1. Image dilution correction method (`correct_img()`, `dilutioncorr`) can now also be called with a single floating point number specifying a constant plume distance (less accurate, but may be required for a fast check or if measurement geometry is not available)
2. New methods in `ImgList`:

- `prep_data_dilutioncorr()`: prepares relevant data for dilution correction (used e.g. in `correct_dilution()`)
- `correct_dilution_all()`: corrects and saves all images in list for signal dilution (optionally also attached off band list)

7.5.11 15/03/2017

1. NEW background correction mode (VALUE - 99) in `PlumeBackgroundModel`. In this mode, plume and background image are used as they are without any modifications. This mode may be used in case a plume background image
2. `PlumeBackgroundModel` raises `AttributeError` in case plume and background image have different vignetting correction states.
3. Changed scaling of plume background (mode=0) to plume image such that it is done for an initial tau image fulfilling tau=0 in `scale_rect` (little faster)
4. Additional features in `ImgList` objects:

   - Background modelling mode can now be set directly using `BG_MODEL_MODE` which takes care of changing the mode and directly reloads the images in the list.
   - New mode for background correction in lists: now, also a background list can be linked and assigned using `bg_list`. The background image access mode (from list vs. global BG image) can be set via `which_bg`

   1. Renamed method `apply_current_edit` to `_apply_edit` in `ImgList` classes

7.5.12 16/03/2017 - 20/03/2017

Note: major changes related to optical flow histogram analysis. Retrieval of main flow field parameters using MultiGaussFit does not work at the moment. Will be fixed tomorrow in

1. Improved performance and stability of optical flow histogram analysis by better controlling the number of bins and settings allowing for a threshold specifying the minimum number of significant vectors for retrieval of mean flow field parameters (see new parameter `min_count_frac` in optical flow settings class).
2. `LineOnImage` object can now also created rotated ROIs, i.e. rectangles aligned with the line orientation
3. Remove `prepare_intensity_condition_mask()` from `OpticalFlowFarneback` (it was causing more confustion than help).
4. Removed multigauss analysis of length histogram due to instability of the fit. The mean displacement length is now determined using mean and std of all pixels longer than `min_length` and which are pointing into the right direction. The latter is retrieved from multi Gauss fit applied to orientation histogram in specified ROI.

- removed `hist_len_how` in `OpticalFlowFarnebackSettings`
- removed `hist_len_gnum_max` in `OpticalFlowFarnebackSettings`

- removed `estimate_mean_len_argmax()` from `OpticalFlowFarneback`

5. New parameter `hist_dir_binres` in optical flow settings class: can be used to set the bin width in degrees for the fit of the flow orientation histogram (defaults to 10)
6. New parameter `roi_rad` in optical flow settings class: can be used to set the ROI used for setting min / max intensity range before calculating flow field (only relevant if `auto_update` is True)
7. Finalised and tested new retrieval of local optical flow parameters including rotated ROIs of multiple lines.
8. Applied appropriate changes to all relevant example scripts.
9. Improved plotting methods for optical flow histogram analysis, including new methods:

- `plot_orientation_histo()`
- `plot_length_histo()`

10. Added features to `LocalPlumeProperties`, most important ones:

- `displacement_vectors`: array containing all displacement vectors of the time series
- `significance`: array containing information about the fraction of significant optical flow vectors within ROI used to retrieve the mean flow field parameters (can be used, e.g. for quality check and is included in `plot()` function).
- `plot_magnitudes()`: plots time series (+/- error) of displacement magnitudes.
- `plot()` (Beta): plot 3 subplots showing detailed information of retrieval results
- `to_dict()`: dictionary representation of object
- `from_dict()`: import data from dictionary
- `to_pandas_dataframe()`: convert object into pandas DataFrame object
- `from_pandas_dataframe()`: self explanatory
- `save_txt()`: save data as text file
- `load_txt()`: load data from text file
- `interpolate()`: interpolate missing data points (uses pandas DataFrame interpolation method)
- `dropna()`: remove missing data points (uses pandas DataFrame dropna method)
- `apply_median_filter()`: applies median filter to data
- `apply_gaussian_filter()`: applies median filter to data

7.5.13 22/03/2017

1. Added older versions of example scripts (for older pyplis versions)
2. Renamed some output files of example scripts
3. Added version check in all example scripts (raises error if version conflict occurs)

7.5.14 23/03/2017

1. Added functionality to `ImgList` objects:

- `optflow_histo_analysis()`: performs optical flow histo analysis for all images in list and arbitrary number of PCS lines

2. Renamed method ‘‘get_main_flow_field_params‘ in `OpticalFlowFarneback` to ‘‘local_flow_params‘ (old name still also works)
3. A colour can now be assigned on init (using kwargs) to `LocalPlumeProperties`
4. Renamed classes:

- `OpticalFlowFarnebackSettings` -> `FarnebackSettings` (old name still works but gives a warning on class initiation)
- `OpticalFlowFarneback` -> `OptflowFarneback` (old name still works but gives a warning on class initiation)

7.5.15 24/03/2017

1. Improved tau-preview plot in `PlumeBackgroundModel`
2. Added option `save_tau_prev` to `correct_dilution_all()` in `ImgList` class: if True, then tau preview images are stored as well.

7.5.16 28-29/03/2017

1. Reviewed and optimised fitting routine in `MultiGaussFit`
2. Changed retrieval of main peak params and analysis of magnitude histogram using moments of distributions

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

P

R

S

T

U

V

W

X

Y

Z