fpipy Documentation

Release 0.1.0

The fpipy developers

Oct 23, 2019

CONTENTS:

Tools for reading and manipulating raw data from a Fabry-Perot interferometer based hyperspectral imager.

- Free software: MIT license
- Documentation: https://fpipy.readthedocs.io.

CHAPTER

ONE

INSTALLATION

1.1 Using PIP

For normal use, it is enough to install fpipy using pip from git:

```
pip install git+https://github.com/silmae/fpipy.git
```

Optionally, you may also specify dask as an extra dependency if you wish to enable parallel computation. Currently this is just for convenience (you will get the same result if you install dask separately):

```
pip install git+https://github.com/silmae/fpipy.git[dask]
```

If you need to access ENVI files (such as the included example hyperspectral datasets) you will need to install rasterio. Specifying ENVI as an extra dependency will install it if you have the required system libraries for GDAL:

```
pip install git+https://github.com/silmae/fpipy.git[ENVI]
```

However, due to the difficulty of installing the necessary libraries (especially on Windows) it is recommended you use Conda if you wish to access ENVI files.

1.2 Using Conda

Using conda, you can create an environment with rasterio and xarray, then install fpipy using pip:

```
conda create -n <env_name> rasterio xarray
pip install git+https://github.com/silmae/fpipy.git
```

There are also multiple ready-made environments under the *fpipy/envs* directory which may be used to create suitable conda environments using:

```
conda env create -n <env_name> --file fpipy/envs/<env_name>.yml
```

See *Contributing* for more info on using development environments.

CHAPTER

TWO

API REFERENCE

2.1 Main package

fpipy	Top-level package for Fabry-Perot Imaging in Python.

2.1.1 fpipy

Top-level package for Fabry-Perot Imaging in Python.

fpipy.**read_hdt**(*hdtfile*)
Load metadata from a .hdt header file (VTT format).

fpipy.**read_ENVI_cfa**(*filepath*, *raw_unit='dn'*, ***kwargs*)
Read ENVI format CFA data and metadata to an xarray Dataset.

For the VTT format raw ENVI files the ENVI metadata is superfluous and is discarded, with the actual metadata read from the separate VTT header file. Wavelength and fwhm data will be replaced with information from metadata and number of layers etc. are omitted as redundant. Gain and bayer pattern are assumed to be constant within each file.

Parameters

- **filepath** (*str*) – Path to the datafile to be opened, either with or without extension. Expects data and metadata to have extensions .dat and .hdt.
- **raw_unit** (*str, optional*) – Units for the raw data.
- ****kwargs** – Keyword arguments to be passed on to *xr.open_rasterio*.

Returns **dataset** – Dataset derived from the raw image data and accompanying metadata. If the ENVI data had an included dark layer, it is separated into its own data variable in the dataset.

Return type xarray.Dataset

fpipy.**raw_to_radiance**(*raw*, ***kwargs*)
Performs demosaicing and computes radiance from RGB values.

Parameters

- **raw** (*xarray.Dataset*) – A dataset containing the following variables: *c.sinv_data*, *c.wavelength_data´, 'c.fwhm_data c.camera_exposure c.cfa_data*, *c.dark_reference_data*,
- **dm_method** (*str, optional*) – **{'bilinear', 'DDFAPD', 'Malvar2004', 'Menon2007'}** Demosaicing method. Default is 'bilinear'. See the *colour_demosaicing* package for more info on the different methods.

- **keep_variables** (*list-like, optional*) – List of variables to keep in the result, default None. If you wish to keep the intermediate data, pass the relevant names from *fpipy.conventions*.

Returns **radiances** – Includes computed radiance sorted by wavelength along with original metadata.

Return type xarray.Dataset

fpipy.**raw_to_reflectance**(*raw*, *whiteraw*, *keep_variables=None*)

Performs demosaicing and computes radiance from RGB values.

Parameters

- **raw** (*xarray.Dataset*) – A dataset containing the following variables: *c.cfa_data*, *c.dark_reference_data*, *c.sinv_data*, *c.wavelength_data´, 'c.fwhm_data c.camera_exposure*
- **white** (*xarray.Dataset*) – Same as raw but for a cube that describes a white reference target.
- **keep_variables** (*list-like, optional*) – List of variables to keep in the result, default None. If you wish to keep the intermediate data, pass the relevant names from *fpipy.conventions*.

Returns **reflectance** – Includes computed radiance and reflectance as data variables sorted by wavelength or just the reflectance DataArray.

Return type xarray.Dataset or xarray.DataArray

fpipy.**radiance_to_reflectance**(*radiance*, *white*, *keep_variables=None*)

Computes reflectance from radiance and a white reference cube.

Parameters

- **radiance** (*xarray.Dataset*) – Dataset containing the image(s) to divide by the references.
- **white** (*xarray.Dataset*) – White reference image(s).
- **keep_variables** (*list-like, optional*) – List of variables to keep in the result, default None. If you wish to keep the intermediate data, pass the relevant names from *fpipy.conventions*.

Returns **reflectance** – Dataset containing the reflectance and the original metadata for both datasets indexed by measurement type.

Return type xarray.Dataset

fpipy.**demosaic**(*cfa*, *pattern*, *dm_method*)

Perform demosaicing on a DataArray.

Parameters

- **cfa** (*xarray.DataArray*) – Array containing a stack of CFA images.
- **pattern** (*BayerPattern or str*) – Bayer pattern used to demosaic the CFA.
- **dm_method** (*str*) –

Returns

Return type xarray.DataArray

fpipy.**subtract_dark**(*ds*, *keep_variables=None*)

Subtracts dark current reference from image data.

Subtracts a dark reference frame from all the layers in the given raw data and clamps any negative values in the result to zero. The result is stored in the dataset as the variable *c.dark_corrected_cfa_data* which is overwritten if it exists.

Parameters

- **ds** (*xarray.DataSet*) – Dataset containing the raw images in *fpipy.conventions.cfa_data* and the dark current reference measurement as *fpipy.conventions.dark_reference_data*.
- **keep_variables** (*list-like, optional*) – List of variables to keep in the result, default None. If you wish to keep the dark reference data and/or the original raw images, pass a list including the variable names.

Returns Dataset with the dark corrected data as *fpipy.conventions.dark_corrected_cfa_data*

Return type xarray.Dataset

2.2 Submodules

io	Functions for reading data from various formats.
raw	Functions for manipulating raw CFA data.
meta	Metadata parsing.
data	Test images.
conventions	Conventions for data, coordinate and attribute names.
simulate	Tools for simulating CFA data from radiance cubes.
testing	Test data generators for debugging and benchmarking.

2.2.1 fpipy.io

Functions for reading data from various formats.

Functions

read_ENVI_cfa(filepath[, raw_unit])	Read ENVI format CFA data and metadata to an xarray Dataset.
read_calibration(calibfile[, wavelength_unit])	Read a CSV calibration file to a structured dataset.
read_hdt(hdtfile)	Load metadata from a .hdt header file (VTT format).

fpipy.io.**read_ENVI_cfa**(*filepath*, *raw_unit='dn'*, ***kwargs*)

Read ENVI format CFA data and metadata to an xarray Dataset.

For the VTT format raw ENVI files the ENVI metadata is superfluous and is discarded, with the actual metadata read from the separate VTT header file. Wavelength and fwhm data will be replaced with information from metadata and number of layers etc. are omitted as redundant. Gain and bayer pattern are assumed to be constant within each file.

Parameters

- **filepath** (*str*) – Path to the datafile to be opened, either with or without extension. Expects data and metadata to have extensions .dat and .hdt.
- **raw_unit** (*str, optional*) – Units for the raw data.

- ****kwargs** – Keyword arguments to be passed on to *xr.open_rasterio*.

Returns **dataset** – Dataset derived from the raw image data and accompanying metadata. If the ENVI data had an included dark layer, it is separated into its own data variable in the dataset.

Return type xarray.Dataset

`fpipy.io.`**`read_calibration`**(*calibfile*, *wavelength_unit='nm'*)

Read a CSV calibration file to a structured dataset.

Parameters

- **calibfile** (*str*) – Filepath to the CSV file containing the metadata. The CSV is assumed to have the following columns (case-sensitive, in no specific order): ['Npeaks', 'SP1', 'SP2', 'SP3', 'PeakWL', 'FWHM', 'Sinv']
- **wavelength_unit** (*str, optional*) – Unit of the wavelength data in the calibration file.

Returns Dataset containing the calibration data in a structured format.

Return type xr.Dataset

`fpipy.io.`**`read_hdt`**(*hdtfile*)

Load metadata from a .hdt header file (VTT format).

2.2.2 fpipy.raw

Functions for manipulating raw CFA data.

Example

Calculating radiances from raw data and plotting them can be done as follows:

```
import xarray as xr
import fpipy as fpi
import matplotlib
import os.path as osp
from fpipy.data import house_raw

data = house_raw() # Example raw data (including dark current)
rad = fpi.raw_to_radiance(data)
rad.swap_dims({'band': 'wavelength'}).radiance.sel(wavelength=600,
                                                   method='nearest').plot()
```

Functions

demosaic(cfa, pattern, dm_method)	Perform demosaicing on a DataArray.
radiance_to_reflectance(radiance, white[, ...])	Computes reflectance from radiance and a white reference cube.
raw_to_radiance(raw, **kwargs)	Performs demosaicing and computes radiance from RGB values.
raw_to_reflectance(raw, whiteraw[, ...])	Performs demosaicing and computes radiance from RGB values.
subtract_dark(ds[, keep_variables])	Subtracts dark current reference from image data.

Classes

BayerPattern	Enumeration of the Bayer Patterns as used by FPI headers.

class fpipy.raw.**BayerPattern**
Enumeration of the Bayer Patterns as used by FPI headers.

BGGR = 2

BayerBG = 2

BayerGB = 0

BayerGR = 1

BayerRG = 3

GBRG = 0

GRBG = 1

RGGB = 3

bggr = 2

gbrg = 0

get = <bound method BayerPattern.get of <enum 'BayerPattern'>>

grbg = 1

rggb = 3

fpipy.raw.**demosaic**(*cfa*, *pattern*, *dm_method*)
Perform demosaicing on a DataArray.

Parameters

- **cfa** (*xarray.DataArray*) – Array containing a stack of CFA images.
- **pattern** (*BayerPattern or str*) – Bayer pattern used to demosaic the CFA.
- **dm_method** (*str*) –

Returns

Return type xarray.DataArray

fpipy.raw.**radiance_to_reflectance**(*radiance*, *white*, *keep_variables=None*)
Computes reflectance from radiance and a white reference cube.

Parameters

- **radiance** (*xarray.Dataset*) – Dataset containing the image(s) to divide by the references.
- **white** (*xarray.Dataset*) – White reference image(s).
- **keep_variables** (*list-like, optional*) – List of variables to keep in the result, default None. If you wish to keep the intermediate data, pass the relevant names from *fpipy.conventions*.

Returns **reflectance** – Dataset containing the reflectance and the original metadata for both datasets indexed by measurement type.

Return type xarray.Dataset

fpipy.raw.**raw_to_radiance**(*raw*, ***kwargs*)

Performs demosaicing and computes radiance from RGB values.

Parameters

- **raw** (*xarray.Dataset*) – A dataset containing the following variables: *c.sinv_data*, *c.wavelength_data´, 'c.fwhm_data c.camera_exposure c.cfa_data*, *c.dark_reference_data*,
- **dm_method** (*str, optional*) – **{'bilinear', 'DDFAPD', 'Malvar2004', 'Menon2007'}** Demosaicing method. Default is 'bilinear'. See the *colour_demosaicing* package for more info on the different methods.
- **keep_variables** (*list-like, optional*) – List of variables to keep in the result, default None. If you wish to keep the intermediate data, pass the relevant names from *fpipy.conventions*.

Returns **radiances** – Includes computed radiance sorted by wavelength along with original metadata.

Return type xarray.Dataset

fpipy.raw.**raw_to_reflectance**(*raw*, *whiteraw*, *keep_variables=None*)

Performs demosaicing and computes radiance from RGB values.

Parameters

- **raw** (*xarray.Dataset*) – A dataset containing the following variables: *c.cfa_data*, *c.dark_reference_data*, *c.sinv_data*, *c.wavelength_data´, 'c.fwhm_data c.camera_exposure*
- **white** (*xarray.Dataset*) – Same as raw but for a cube that describes a white reference target.
- **keep_variables** (*list-like, optional*) – List of variables to keep in the result, default None. If you wish to keep the intermediate data, pass the relevant names from *fpipy.conventions*.

Returns **reflectance** – Includes computed radiance and reflectance as data variables sorted by wavelength or just the reflectance DataArray.

Return type xarray.Dataset or xarray.DataArray

fpipy.raw.**subtract_dark**(*ds*, *keep_variables=None*)

Subtracts dark current reference from image data.

Subtracts a dark reference frame from all the layers in the given raw data and clamps any negative values in the result to zero. The result is stored in the dataset as the variable *c.dark_corrected_cfa_data* which is overwritten if it exists.

Parameters

- **ds** (*xarray.DataSet*) – Dataset containing the raw images in *fpipy.conventions.cfa_data* and the dark current reference measurement as *fpipy.conventions.dark_reference_data*.
- **keep_variables** (*list-like, optional*) – List of variables to keep in the result, default None. If you wish to keep the dark reference data and/or the original raw images, pass a list including the variable names.

Returns Dataset with the dark corrected data as *fpipy.conventions.dark_corrected_cfa_data*

Return type xarray.Dataset

2.2.3 fpipy.meta

Metadata parsing.

Functions

parse_image_meta(meta, layer)	Parse metadata for a given image (layer) in the FPI data.
parse_meta_to_ds(meta)	Parse FPI metadata from a configparser to a xarray.Dataset.
parse_peakmeta(s)	
parse_setpoints(s)	
parse_sinvs(s)	Parse an array of floats from a string.
parsevec(s)	Parse a vector of floats from a string.

fpipy.meta.**parse_image_meta**(*meta*, *layer*)
Parse metadata for a given image (layer) in the FPI data.

fpipy.meta.**parse_meta_to_ds**(*meta*)
Parse FPI metadata from a configparser to a xarray.Dataset.

fpipy.meta.**parse_peakmeta**(*s*)

fpipy.meta.**parse_setpoints**(*s*)

fpipy.meta.**parse_sinvs**(*s*)
Parse an array of floats from a string.

fpipy.meta.**parsevec**(*s*)
Parse a vector of floats from a string.

2.2.4 fpipy.data

Test images.

Functions

house_calibration()	Calibration sequence for the house dataset.
house_radiance(**kwargs)	Radiance image of a house and foliage.
house_raw(**kwargs)	Raw images of a house and nearby foliage.

fpipy.data.**house_radiance**(***kwargs*)
Radiance image of a house and foliage.

Radiances calculated by the VTT software from the *house_raw* dataset.

CC0

Parameters ****kwargs** – Parameters passed on to *xr.open_rasterio*.

Returns **house_rad** – (4, 400, 400) radiance cube with metadata.

Return type xarray.Dataset

fpipy.data.**house_raw**(***kwargs*)
Raw images of a house and nearby foliage.

Raw CFA data from the VTT FPI imager.

CC0

Parameters ****kwargs** – Keyword arguments passed on to *read_ENVI_cfa*.

Returns **house_raw** – (4, 400, 400) cube of CFA images with metadata.

Return type xarray.Dataset

fpipy.data.**house_calibration**()
Calibration sequence for the house dataset.

Calibration data for the *house_raw* dataset as read from the camera calibration file (instead of the VTT generated header).

2.2.5 fpipy.conventions

Conventions for data, coordinate and attribute names.

fpipy.conventions.**RGB_dims = ('y', 'x', 'colour')**
Convention for RGB images.

fpipy.conventions.**band_index = 'band'**
Index for wavelength band data.

fpipy.conventions.**camera_exposure = 'exposure'**
Camera exposure time in milliseconds.

fpipy.conventions.**camera_gain = 'gain'**
Camera analog gain value.

fpipy.conventions.**cfa_data = 'dn'**
DataArray containing a stack of raw CFA images.

fpipy.conventions.**cfa_dims = ('index', 'y', 'x')**
Convention for CFA image stacks.

fpipy.conventions.**cfa_pattern_data = 'bayer_pattern'**
String denoting the pattern (e.g. RGGB) of the colour filter array.

fpipy.conventions.**colour_coord = 'colour'**
Colour coordinate for values indexed w.r.t. CFA colours.

fpipy.conventions.**dark_corrected_cfa_data = 'dn_dark_corrected'**
DataArray containing a stack of raw images with dark current corrected

fpipy.conventions.**dark_ref_dims = ('y', 'x')**
Convention for dark reference image dimensions.

fpipy.conventions.**dark_reference_data = 'dark'**
DataArray containing a dark current reference.

fpipy.conventions.**fwhm_data = 'fwhm'**
Full Width at Half Maximum values

fpipy.conventions.**genicam_exposure = 'ExposureTime'**
Camera exposure time in microseconds as defined by GenICam.

fpipy.conventions.**genicam_pattern_data = 'PixelColorFilter'**
String denoting the pattern (e.g. BayerGB) of the colour filter array.

fpipy.conventions.**image_index = 'index'**
Index number of the image in a given stack.

fpipy.conventions.**number_of_peaks = 'npeaks'**
Number of passband peaks included in a given image.

fpipy.conventions.**peak_coord = 'peak'**
Index coordinate denoting passband peaks of the FPI.

fpipy.conventions.**radiance_data = 'radiance'**
DataArray containing a stack of radiance images.

fpipy.conventions.**radiance_dims = ('band', 'y', 'x')**
Convention for radiance image stacks.

fpipy.conventions.**reflectance_data = 'reflectance'**
DataArray containing a stack of reflectance images.

fpipy.conventions.**rgb_data = 'rgb'**
DataArray containing a stack of RGB images.

fpipy.conventions.**setpoint_coord = 'setpoint_index'**
Coordinate denoting the the setpoint (which voltage it denotes)

fpipy.conventions.**setpoint_data = 'setpoint'**
Values of the setpoints.

fpipy.conventions.**sinv_data = 'sinvs'**
Inversion coefficients for radiance calculation.

fpipy.conventions.**sinv_dims = ('index', 'peak', 'colour')**
Convention for inversion coefficient dimensions.

fpipy.conventions.**wavelength_data = 'wavelength'**
Passband center wavelength values.

2.2.6 fpipy.simulate

Tools for simulating CFA data from radiance cubes.

Functions

bayer_sensor(radiance, exposure, T_mosaic, ...)	Simulate a Bayer sensor image.
create_cfa(rad, S, pattern)	Simulate a colour filter array data from radiance data.
etalon_gap(wl, fsr, ng, theta)	Approximate value of the Fabry-Perot etalon gap.
finesse_coefficient(R)	Finesse coefficient of the etalon
fpi_bandpass_lims(d, n)	Bandpass filter limits for a single order of an FPI at given gap.
fpi_bayer_imager(radiance, T_fpi, exposure, ...)	Simulate a Fabry-Perot interferometer filtered Bayer sensor image.
fpi_bayer_spectral_signal(T_fpi, Q_eff, T_rgb)	Spectral signal for a given FPI gap and order.
fpi_gap(wl, fsr)	Approximate value of the FPI gap.
fpi_phase_difference(wl, l)	Phase difference between pairs of transmitted beams in the FPI
fpi_transmittance(wl, l, R)	Transmittance of the FPI at given wavelength, gap and mirror reflectance

Continued on next page

Table 8 – continued from previous page

fpi_triplet(wl, l)	Generate a triplet of etalon peaks (wavelengths) given the lowest.
free_spectral_range(wl, l, ng, theta)	Free spectral range of the Fabry-Perot etalon as
fsr_fpi(wl, l)	Free spectral range in the FPI.
mono_sensor(radiance, exposure, Qeff, pxformat)	Simulate a monochromatic sensor image.
mosaic_transmittances(shape, pattern, T_rgb)	Transmittances of a Bayer filter mosaic.
phase_difference(wl, n, l, theta)	Phase difference between pairs of transmitted beams in the etalon
quantize(im, pxformat)	Quantize a floating-point image to the desired pixel format.

fpipy.simulate.**bayer_sensor**(*radiance*, *exposure*, *T_mosaic*, *Q_eff*, *pxformat*)

Simulate a Bayer sensor image.

Parameters

- **radiance** (*array-like*) – (y, x, band) array of radiance values.
- **exposure** (*float*) – Exposure (integration time) in milliseconds.
- **T_mosaic** (*array-like*) – (y, x, band) array of spectral transmittances for the Bayer mosaic.
- **Q_eff** (*array-like*) – Quantum efficiencies of the sensor for each band/wavelength.
- **pxformat** (*str*) – Pixel format to discretize result to.
- **Result** –
- **------** –
- **np.ndarray** – (y, x) Bayer mosaic

fpipy.simulate.**create_cfa**(*rad*, *S*, *pattern*)

Simulate a colour filter array data from radiance data.

Parameters

- **rad** (*xarray.DataArray*) – Radiance datacube with wavelength information.
- **S** (*list of xarray.DataArray*) – Responses for different colours of the CFA for each wavelength
- **pattern** (*BayerPattern or str*) – Bayer pattern for the CFA.

Returns **cfa** – CFA images with the given pattern and responses.

Return type *xarray.Dataset*

Examples

Using a mockup response matrix to create a CFA from radiance:

```
import xarray as xr
import numpy as np
from fpipy.data import house_radiance
from fpipy.simulate import create_cfa

# load example radiance data
rad = house_radiance()
```

(continues on next page)

(continued from previous page)

```
rad = rad.swap_dims({'band':'wavelength'})

# create a mockup response matrix
S1 = xr.DataArray(
    np.eye(3),
    dims=('colour', 'wavelength'),
    coords={
        'colour':['R','G','B'],
        'wavelength': rad.wavelength.data[:-1]
        }
    )
S2 = xr.DataArray(
    np.eye(3),
    dims=('colour', 'wavelength'),
    coords={
        'colour':['R','G','B'],
        'wavelength': rad.wavelength.data[1:]
        }
    )

S = [S1, S2]

# Simulate a RGGB pattern CFA
simulated_raw = create_cfa(rad, S, 'RGGB')
```

fpipy.simulate.**etalon_gap**(*wl*, *fsr*, *ng*, *theta*)

Approximate value of the Fabry-Perot etalon gap.

Approximates the gap length l from the formula of the FSR as

$$l = \frac{\lambda(\lambda - \Delta\lambda)}{\Delta\lambda 2n\cos(\theta)}$$

Parameters

- **wl** (*np.float64*) – Peak wavelength
- **fsr** (*np.float64*) – Free spectral range near the peak.
- **ng** (*np.float64*) – Group refractive index of the media.
- **theta** (*np.float64*) – Angle of the light entering the etalon.

Returns **l** – Approximate gap length of the Fabry-Perot etalon.

Return type np.float64

fpipy.simulate.**finesse_coefficient**(*R*)

Finesse coefficient of the etalon

Parameters **R** (*np.float64*) – Reflectance of the etalon mirrors

Returns Finesse coefficient of the etalon

Return type np.float64

fpipy.simulate.**fpi_bandpass_lims**(*d*, *n*)

Bandpass filter limits for a single order of an FPI at given gap.

Parameters

- **d** (*float*) – Gap length of the Fabry-Perot etalon.

- **n** (*int*) – The order of the FPI peak included in the limits

Returns **(lmin, lmax)** – Minimum and maximum wavelengths that include the three FPI orders.

Return type tuple of float

fpipy.simulate.**fpi_bayer_imager**(*radiance*, *T_fpi*, *exposure*, *T_mosaic*, *Q_eff*, *pxformat*)
Simulate a Fabry-Perot interferometer filtered Bayer sensor image.

Parameters

- **radiance** (*array-like*) – (y, x, band) array of radiance values.
- **T_fpi** (*array-like*) – (a, b) array of Fabry-Perot interferometer transmittances for each band and gap length value a.
- **exposure** (*float*) – Exposure (integration time) in milliseconds.
- **T_mosaic** (*array-like*) – (y, x, band) array of spectral transmittances for the Bayer mosaic.
- **Q_eff** (*array-like*) – Quantum efficiencies of the sensor for each band/wavelength.
- **pxformat** (*str*) – Pixel format to discretize result to.
- **Result** –
- **------** –
- **np.ndarray** – (a, y, x) stack of Bayer mosaic images

fpipy.simulate.**fpi_bayer_spectral_signal**(*T_fpi*, *Q_eff*, *T_rgb*)
Spectral signal for a given FPI gap and order.

Parameters

- **T_fpi** (*array-like*) – (3, band) array of FPI transmittances for orders n, n+1 and n+2 for a given etalon gap length.
- **Q_eff** (*array-like*) – (band,) array of quantum efficiencies of the sensor
- **T_rgb** (*array-like*) – (3, band) array of tranmittances of the R, G and B pixels.

Returns **S** – (3, 3) matrix of effective transmittances for the FPI imager.

Return type np.ndarray

fpipy.simulate.**fpi_gap**(*wl*, *fsr*)
Approximate value of the FPI gap.

Special case of *etalon_gap* with $ng = 1$ and $\theta = 0$.

Parameters

- **wl** (*np.float64*) – Peak wavelength
- **fsr** (*np.float64*) – Free spectral range near the peak.

Returns **l** – Approximate gap length of the FPI.

Return type np.float64

fpipy.simulate.**fpi_phase_difference**(*wl*, *l*)
Phase difference between pairs of transmitted beams in the FPI

Parameters

- **wl** (*np.float64*) – Wavelength of the light in chosen units (matching gap length)

- **l** (*np.float64*) – Gap length in chosen units (matching wavelength)

Returns Phase difference in radians

Return type np.float64

fpipy.simulate.**fpi_transmittance**(*wl*, *l*, *R*)
Transmittance of the FPI at given wavelength, gap and mirror reflectance

Parameters

- **wl** (*np.float64*) – Wavelength in chosen units (matching gap length)
- **l** (*np.float64*) – Gap length in chosen units (matching wavelength)
- **R** (*np.float64*) – Reflectance of the FPI mirrors

Returns Reflectance of the Fabry-Perot interferometer

Return type np.float64

fpipy.simulate.**fpi_triplet**(*wl*, *l*)
Generate a triplet of etalon peaks (wavelengths) given the lowest.

Parameters

- **wl** (*np.float64*) – Lowest wavelength of the triplet.
- **l** (*np.float64*) – Gap of the etalon.

Returns **(wl1, wl2, wl3)** – Triplet of consecutive peaks of the Fabry-Perot etalon

Return type tuple of np.float64

fpipy.simulate.**free_spectral_range**(*wl*, *l*, *ng*, *theta*)
Free spectral range of the Fabry-Perot etalon as

$$\Delta\lambda = \frac{\lambda^2}{2nl\cos(\theta)}$$

Parameters

- **wl** (*np.float64*) – Wavelength of the nearest peak.
- **l** (*np.float64*) – Gap of the etalon.
- **ng** (*np.float64*) – Group refractive index of the media.
- **theta** (*np.float64*) – Angle of the ligth entering the etalon.

Returns **fsr** – Free spectral range of the Fabry-Perot etalon

Return type np.float64

fpipy.simulate.**fsr_fpi**(*wl*, *l*)
Free spectral range in the FPI.

Special case of *free_spectral_range* with $n_g = 1$ for air and collimated light at $\theta = 0$.

Parameters

- **wl** (*np.float64*) – Wavelength of the nearest peak.
- **l** (*np.float64*) – Gap of the etalon.

Returns FSR of the FPI at the given values.

Return type np.float64

fpipy.simulate.**mono_sensor**(*radiance*, *exposure*, *Qeff*, *pxformat*)
Simulate a monochromatic sensor image.

Simulates a linear monochromatic sensor response for a given radiance signal and exposure.

Parameters

- **radiance** (*array-like*) – (y, x, band) array of radiance values.
- **exposure** (*float*) – Exposure (integration time) in milliseconds.
- **Q_eff** (*array-like*) – Quantum efficiencies of the sensor for each band/wavelength.
- **pxformat** (*str*) – Pixel format to discretize result to.

Returns (y, x) monochromatic image.

Return type np.ndarray

fpipy.simulate.**mosaic_transmittances**(*shape*, *pattern*, *T_rgb*)
Transmittances of a Bayer filter mosaic.

Parameters

- **shape** (*pair of int*) – (y, x) Shape of the filter array.
- **pattern** (*BayerPattern or str*) – The Bayer filter pattern of the array.
- **T_rgb** (*array-like*) – (3, b) arrays of transmittances of the R, G and B pixels for each band.
- **Result** –
- **------** –
- **np.ndarray** – (y, x, b) array of mosaic responses for each band.

fpipy.simulate.**phase_difference**(*wl*, *n*, *l*, *theta*)
Phase difference between pairs of transmitted beams in the etalon

Parameters

- **wl** (*np.float64*) – Wavelength of the light in chosen units (matching gap length)
- **n** (*np.float64*) – Refractive index of the mirrors
- **l** (*np.float64*) – Gap length in chosen units (matching wavelength)
- **theta** (*np.float64*) – Angle of the beam entering the etalon in radians

Returns Phase difference in radians

Return type np.float64

fpipy.simulate.**quantize**(*im*, *pxformat*)
Quantize a floating-point image to the desired pixel format.

Simple quantization to maximum levels allowed by the pixel format and including the full range of the data.

Parameters

- **im** (*array-like*) – Array of floating point values.
- **pxformat** (*str*) – Pixel format string (as defined in GenICAM).

2.2.7 fpipy.testing

Test data generators for debugging and benchmarking.

Functions

cfa(size, pattern, R, G, B)	CFA data with given R, G and B values.
dark(shape, level)	Dark frame with given shape and level.
metadata(size, wl_range)	
rad(cfa, dark_level, exposure, metas, wl_range)	Radiance data corresponding to CFA with 1, 2 and 5 as R, G, and B.
raw(cfa, dark, pattern, exposure, gain, ...)	Raw data (CFA, dark and metadata).

fpipy.testing.**cfa**(*size*, *pattern*, *R*, *G*, *B*)
CFA data with given R, G and B values.

fpipy.testing.**dark**(*shape*, *level*)
Dark frame with given shape and level.

fpipy.testing.**metadata**(*size*, *wl_range*)

fpipy.testing.**rad**(*cfa*, *dark_level*, *exposure*, *metas*, *wl_range*)
Radiance data corresponding to CFA with 1, 2 and 5 as R, G, and B. and given constant dark level.

fpipy.testing.**raw**(*cfa*, *dark*, *pattern*, *exposure*, *gain*, *metas*, *wl_range*)
Raw data (CFA, dark and metadata).

CHAPTER

THREE

COMMAND LINE SCRIPTS

This is a list of command line utilities that are installed with fpipy.

3.1 raw2rad

Convert raw CFA data to radiances.

```
usage: raw2rad [-h] [-f output_format] [-o output_dir] [-p output_prefix]
               input file [input file ...]
```

input file
: List of files to process to radiance.

-h, --help
: show this help message and exit

-f {netCDF}, **--format** {netCDF}
: Output file format for the processed radiances, default "netCDF"

-o <output_dir>, **--odir** <output_dir>
: Directory to write output files to, default "."

-p <output_prefix>, **--prefix** <output_prefix>
: Filename prefix to add to output filenames, default "**RAD_**"

CHAPTER

FOUR

GLOSSARY

CFA Short for *Colour Filter Array*.

Colour Filter Array Array of various colour filter used to enable measurement of colour images using a monochromatic sensor by sampling different colours from neighbouring points. Usual arrangements red, green and blue pixels are referred to as Bayer filters.

DN Short for *Digital Number*.

Digital Number Values received from the sensor (after analog-to-digital conversion and possible sensor firmware modification).

Fabry-Perot interferometer Optical component composed of two parallel mirrors with a set air gap between them. Used as a tunable bandpass filter in hyperspectral imagers. See Wikipedia for more.

CHAPTER

FIVE

CHANGELOG

All notable changes to this project will be documented in this file.

5.1 [Unreleased]

CHAPTER

SIX

CONTRIBUTING

Contributions are welcome.

Report bugs or give feedback at https://github.com/silmae/fpipy/issues.

6.1 Get Started!

Ready to contribute? Here's how to set up *fpipy* for local development.

1. Fork the *fpipy* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/fpipy.git
```

3. Install the dependencies using the provided conda environment file and then the package itself using pip:

```
$ cd fpipy/
$ conda env create -f envs/development.yml
$ conda activate fpipy-dev
$ pip install -e .
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass tests:

```
$ flake8 fpipy tests
$ make test
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.2 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. Put any new functionality into a function with a docstring.
3. The pull request should work for all configurations. Check https://travis-ci.org/silmae/fpipy/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER

SEVEN

AUTHORS

7.1 Development lead

Developed at the Spectral Imaging Laboratory of the Faculty of Information Technology in the University of Jyväskylä by

- Matti A. Eskelinen <matti.a.eskelinen@student.jyu.fi>
- Jyri Hämäläinen <jyri.p.hamalainen@gmail.com>

7.2 Contributors

- Severi Jääskeläinen <severi.a.jaaskelainen@student.jyu.fi>
- Samuli Rahkonen <samuli.rahkonen@jyu.fi>
- Leevi Annala <leevi.a.annala@jyu.fi>

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

Symbols

B

C

D

E

F

G

H

I

M

N

P

Q

R

S

W