micropolarray

Herve Haudemand

Feb 08, 2024

CONTENTS:

CHAPTER

ONE

MICROPOLARRAY PACKAGE

1.1 Subpackages

1.1.1 micropolarray.processing package

1.1.1.1 Submodules

1.1.1.2 micropolarray.processing.chen_wan_liang_calibration module

`micropolarray.processing.chen_wan_liang_calibration.`**`calculate_chen_wan_lian_calibration`**(*polarizer_orientatio* *file-names_list*, *mi-cropol_phases_prev* *out-put_dir*, *oc-cul-ter=True*, *dark_filename=Non* *flat_filename=None*)

Performs calibration from Chen-Wang-Liang paper 2014

Parameters

- **`polarizer_orientations`** (`list[float]`) – List of polarizer orienataions
- **`filenames_list`** (`list[str]`) – List of filenames coupled with
- **`micropol_phases_previsions`** (`list[float]`) – Previsions of the micropolarizer orientations inside a superpixel
- **`output_dir`** (`str`) – output path for the calibration matrices
- **`occulter`** (`bool, optional`) – wether to exclude the occulter area. Defaults to True.
- **`dark_filename`** (`str, optional`) – path to the dark to be subtracted from the images. Defaults to None.
- **`flat_filename`** (`str, optional`) – path to the dark to be subtracted from the images. Defaults to None.

Raises

- **`ValueError`** – polarizer orientation list and filenames list do not have the same lenght

- **ValueError** – any of 0,45,90,-45 polarizarions is not included in the polarizer orientation list

`micropolarray.processing.chen_wan_liang_calibration.`**`chen_wan_liang_calibration`**(*data*, *calibration_matrices_dir: str*)

Calibrates the images using Chen-Wang-Liang 2014 paper calibration

Parameters

- **data** (*np.array*) – data to be calibrated
- **calibration_matrices_dir** (*str*) – path to the calibration matrices

Returns
calibrated data

Return type
np.array

1.1.1.3 micropolarray.processing.congrid module

`micropolarray.processing.congrid.`**`congrid`**(*a*, *newdims*, *kind='linear'*) → ndarray

Reshapes the data into any new lenght and width

Parameters

- **a** (*np.array*) – data to be reshaped
- **newdims** (*tuple | list*) – new lenght and width
- **kind** (*str, optional*) – interpolation type. Defaults to "linear".

Returns
numpy array of congridded image

Return type
ndarray

`micropolarray.processing.congrid.`**`micropolarray_jitcongrid`**(*data*, *width*, *height*, *scale*)

1.1.1.4 micropolarray.processing.convert module

`micropolarray.processing.convert.`**`average_rawfiles_to_fits`**(*filenames: list*, *new_filename: str*, *height: int*, *width: int*)

Saves the mean of a list of rawfiles to a new fits file.

Parameters

- **filenames** (*list*) – list of raw filenames
- **new_filename** (*str*) – new fits filename
- **height** (*int*) – image height in pix
- **width** (*int*) – image width in pix

Raises
ValueError – trying to save in a file that does not end with .fits

micropolarray.processing.convert.**convert_rawfile_to_fits**(*filename: str*, *height: int*, *width: int*, *remove_old: bool = False*)

Converts a raw file to a fits one, using default header

Parameters

- **filename** (*str*) – raw filename
- **height** (*int*) – file height
- **width** (*int*) – file width
- **remove_old** (*bool, optional*) – remove old raw file after conversion. Defaults to False.

Raises

ValueError – raised if the file does not end with ".raw"

micropolarray.processing.convert.**convert_set**(*filenames*, *new_filename*, *height*, *width*)

ANTARTICOR ONLY: Sums a set of filenames and converts them to one fits file.

Parameters

- **filenames** (*list*) – list of file names to be summed before being converted
- **new_filename** (*str*) – new .fits file name

micropolarray.processing.convert.**merge_rawfiles_to_fits**(*filenames: list*, *new_filename: str*, *height: int*, *width: int*, *mode='sum'*)

Saves the average or sum of a list of rawfiles to a new fits file.

Parameters

- **filenames** (*list*) – list of raw filenames
- **new_filename** (*str*) – new fits filename
- **height** (*int*) – image height in pix
- **width** (*int*) – image width in pix
- **mode** (*str*) – wether to "average" or "sum" the images. Defaults to "sum".

Raises

ValueError – trying to save in a file that does not end with .fits

micropolarray.processing.convert.**nparr_from_binary**(*filename*)

Converts a PolarCam binary file into a numpy array. Bytes are saved like this

- **24 bit (3 bytes)**
  1 | 3 | 2 111111111111 | 1111 | 11111111
- **2 numbers**
  First number 12bit | Second number (little endian) 8+4=12 bit

Parameters

filename (*str*) – name of the file to be converted

Raises

ValueError – file lenghts is indivisible by the number of chunks requested to parallelize operations

Returns

array of data from file

Return type
np.array

`micropolarray.processing.convert.`**`three_bytes_to_two_ints`**`(`*filecontent*`)`

Needed for parallelization, this will be run by each thread for a slice of the original array.

Returns
array of saved data

Return type
np.array

1.1.1.5 micropolarray.processing.demodulation module

`class micropolarray.processing.demodulation.`**`Demodulator`**`(`*demo_matrices_path: str*`)`

Bases: `object`

Demodulation class needed for MicropolImage demodulation.

`property Cij: ndarray`

`property eff: ndarray`

`flip``(`*axis*`)`

`property phi: ndarray`

`rebin``(`*binning*`)`

DO NOT USE THIS, calculate the tensor from the binned images

`rot90``(`*k=1*`)`

`show``(`*vmin=-1*, *vmax=1*, *cmap='Greys'*`)` → tuple

Shows the demodulation tensor

Parameters

- **vmin** (*int, optional*) – Minimum shown value. Defaults to -1.
- **vmax** (*int, optional*) – Maximum shown value. Defaults to 1.
- **cmap** (*str, optional*) – Colormap of the plot. Defaults to "Greys".

Returns
fig, ax tuple as returned by matplotlib.pyplot.subplots

Return type
tuple

`property tk: ndarray`

`micropolarray.processing.demodulation.`**`Malus`**`(`*angle*, *throughput*, *efficiency*, *phase*`)` micropolarray.processing.demodulation.**calculate_demodulation_tensor**(*polarizer_orientations: list*, *filenames_list: list*, *micropol_phases_previsions: list*, *gain: float*, *output_dir: str*, *binning: int = 1*, *occulter: list | None = None*, *procs_grid: list = [4, 4]*, *dark_filename: str | None = None*, *flat_filename: str | None = None*, *normalizing_S=None*, *tk_boundary: list | None = None*, *eff_boundary: list | None = None*, *DEBUG: bool = False*)

Calculates the demodulation tensor images and saves them. Requires a set of images with different polarizations to fit a Malus curve model.

Parameters

- **polarizer_orientations** (*list[float]*) – List containing the orientations of the incoming light for each image.
- **filenames_list** (*list[str]*) – List of input images filenames to read. Must include [0, 45, 90, -45].
- **micropol_phases_previsions** (*list[float]*) – Previsions for the micropolarizer orientations required to initialize fit.
- **gain** (*float*) – Detector [e-/DN], required to compute errors.
- **output_dir** (*str*) – output folder to save matrix to.
- **binning** (*int, optional*) – Output matrices binning. Defaults to 1 (no binning). Be warned that binning matrices AFTER calculation is an incorrect operation.
- **occulter** (*list, optional*) – occulter y, x center and radius to exclude from calculations. Defaults to None.
- **procs_grid** (*[int, int], optional*) – number of processors per side [Y, X], parallelization will be done in a Y x X grid. Defaults to [4,4] (16 procs in a 4x4 grid).
- **dark_filename** (*str, optional*) – Dark image filename to correct input images. Defaults to None.
- **flat_filename** (*str, optional*) – Flat image filename to correct input images. Defaults to None.
- **normalizing_S** (*float or np.ndarray, optional*) – maximum signal used to normalize single pixel signal. If not set will be estimated as the 4sigma of the signal distribution.
- **tk_boundary** (*list*) – if provided, sets the transmittancy [initial guess, boundary_inf, boundary_sup] of the Malus curve (max value). Defaults to [0.5, 0.1, 1.-1.e-6].
- **eff_boundary** (*list*) – if provided, sets the efficiency [initial guess, boundary_inf, boundary_sup] of the Malus curve (max value). Defaults to [0.5, 0.1, 1.-1.e-6].

Raises

ValueError – Raised if any among [0, 45, 90, -45] is not included in the input polarizations.

Notes

In the binning process the sum of values is considered, which is ok because data is normalized over the maximum S before being fitted.

micropolarray.processing.demodulation.**compute_demodulation_by_chunk**(*splitted_normalized_dara_arr*, *splitted_pixel_erorrs*, *splitted_occulter_flag*, *polarizer_orientations*, *rad_micropol_phases_previsions*, *tk_boundary*, *eff_boundary*, *DEBUG*)

Utility function to parallelize calculations.

1.1.1.6 micropolarray.processing.demodulation_errors module

class micropolarray.processing.demodulation_errors.**MicropolImageError**(*image:* MicropolImage, *image_error: ndarray*, *demodulator:* Demodulator)

Bases: `object`

micropolarray.processing.demodulation_errors.**get_error_on_AoLP**(*S: ndarray*, *sigma_S: ndarray*) → ndarray

micropolarray.processing.demodulation_errors.**get_error_on_DoLP**(*S: ndarray*, *sigma_S: ndarray*) → ndarray

micropolarray.processing.demodulation_errors.**get_error_on_Stokes**(*image_error: ndarray*, *demodulator:* Demodulator) → ndarray

Returns the error on the image, propagated through the demodulation matrix. If M[i, j] is the demodulation matrix, sigma_I[k] are the four pixel values in a superpixel, and S[i, j] is the Stokes vector, returns the matrix product sqrt(M^2 @ I^2)

Parameters

- **image_error** (*np.ndarray*) – array containing the pixel by pixel error to propagate.
- **demodulator** (`Demodulator`) – demodulator containing the demodulation matrix.

Returns

errors of the computed Stokes vector as a [3, y, x] array.

Return type

np.ndarray

micropolarray.processing.demodulation_errors.**get_error_on_pB**(*S: ndarray*, *sigma_S: ndarray*) → ndarray

1.1.1.7 micropolarray.processing.demosaic module

micropolarray.processing.demosaic.**demosaic**(*image_data*, *option='adjacent'*)

Returns a [4,n,m] array of polarized images, starting from a micropolarizer image array [n, m].

micropolarray.processing.demosaic.**demosaicadjacent**(*data*)

micropolarray.processing.demosaic.**demosaicmean**(*data*)

Loops over right polarization pixel location, takes 1/4 of that, stores it in the 2x2 superpixel. demo_images[0] = data[y=0, x=0] demo_images[1] = data[y=0, x=1] demo_images[2] = data[y=1, x=0] demo_images[3] = data[y=1, x=1]

micropolarray.processing.demosaic.**merge_polarizations**(*single_pol_images: ndarray*)

micropolarray.processing.demosaic.**split_polarizations**(*data: ndarray*)

1.1.1.8 micropolarray.processing.image_cleaning module

micropolarray.processing.image_cleaning.**auto_threshold**(*data: ndarray*) → float

Get the threshold following Otsu's algorithm. This assumes that there are two populations (noise + signal) and minimizes the intra- class variance

Parameters

data (*np.ndarray*) – array on which to perform the treshold

Returns

Otsu's threshold

Return type

float

micropolarray.processing.image_cleaning.**get_hot_pixels**(*image*, *threshold=100*)

micropolarray.processing.image_cleaning.**reject_outliers**(*data*, *m=2.0*)

micropolarray.processing.image_cleaning.**remove_outliers_simple**(*original*, *neighbours=2*)

EXPERIMENTAL DO NOT USE, for improving fitting on occulter position

1.1.1.9 micropolarray.processing.linear_roi module

micropolarray.processing.linear_roi.**DDA**(*start: list*, *end: list*) → ndarray

Digital_differential_analyzer algorithm for line rasterizing. Unlike bresenham, works in every quadrant. NOTE: even if the distance between start and end coordinates is the same, a different number of points is selected depending on the line slope, so the ratio between distance and number of points is also returned.

Parameters

- **start** (*list*) – starting point coordinates
- **end** (*list*) – ending point coordinates

Returns

interpolated points locations float: ratio between the distance from start to end point and the number of returned locations

Return type

np.ndarray

`micropolarray.processing.linear_roi.`**`bresenham`**`(`*start: list*, *end: list*`)` → ndarray

Bresenham algorithm for generating integers on a line. Efficient BUT works ONLY in the first octant

Parameters

- **`start`** (*`list`*) – starting point coordinates
- **`end`** (*`list`*) – ending point coordinates

Returns

coordinates of the points under the line from start to end

Return type

np.ndarray

`micropolarray.processing.linear_roi.`**`linear_roi`**`(`*data: ndarray*, *start: list*, *end: list*`)` → ndarray

Get values

Parameters

- **`data`** (*`np.ndarray`*) – _description_
- **`start`** (*`list`*) – _description_
- **`end`** (*`list`*) – _description_

Returns

description

Return type

np.ndarray

`micropolarray.processing.linear_roi.`**`linear_roi_from_polar`**`(`*data: ndarray*, *center: list*, *theta: float*, *r: list | None = None*`)` → list

Returns a linear roi starting from the center and extending to r or to the edge of the input data array. Angles start horizontally and rotate anti-clockwise (0deg corresponds to fixed y and increasing x).

Parameters

- **`data`** (*`np.ndarray`*) – input array from which to select a roi
- **`center`** (*`list`*) – center coordinates
- **`theta`** (*`float`*) – angle of the linear roi
- **`r`** (*`list, optional`*) – Maximum radius for the roi. Defaults to
- **`None.`** –

Returns

1-dimensional array containing the selected values from data np.ndarray: roi indexes along the first (y) dimension of data np.ndarray: roi indexes along the second (x) dimension of data float: ratio between the pixel lenght and the lenght of the returned roi (see linear_roi.DDA)

Return type

np.ndarray

1.1.1.10 micropolarray.processing.nrgf module

`micropolarray.processing.nrgf.`**`find_occulter_hough`**(*data: array*, *minr: float = 1*, *maxr: float | None = None*, ***kwargs*) → tuple

Uses Hough Gradient from cv2 and computes the coronagraph occulter coordinates. Returns Y, X, Radius of the occulter.

Parameters
`data` (*`np.array`*) – input data

Returns
occulter y, x, r

Return type
tuple

`micropolarray.processing.nrgf.`**`find_occulter_position`**(*data: array*, *method: str = 'sigmoid'*, *threshold: float = 4.0*)

Finds the occulter position of an image.

Parameters

- **`data`** (*`np.array`*) – input data
- **`method`** (*`str, optional`*) – Method to find occulter edges. If "sigmoid" it will try to fit four sigmoids at the image edges centers, inferring the occulter edges from the parameters. If "algo" it will start from the image edge center and infer the occulter position when DN[i] > threshold*mean(DN[:i]) Defaults to "sigmoid".
- **`threshold`** (*`float, optional`*) – Threshold for the algo method. Defaults to 4.0.

Raises
`UnboundLocalError` – couldn't converge

Returns
occulter y, occulter x, occulter radius

Return type
list

`micropolarray.processing.nrgf.`**`map_polar_coordinates`**(*height*, *width*, *center*)

`micropolarray.processing.nrgf.`**`nrgf`**(*data: array*, *y_center: int | None = None*, *x_center: int | None = None*, *rho_min: int | None = None*, *step: int = 1*, *phi_to_mean=[0.0, 360]*, *output_phi=[0.0, 360]*) → array

Performs nrgf filtering on the image, starting from center and radius. Mean is performed between phi_to_mean, 0 is horizontal right, anti-clockwise.

Parameters

- **`data`** (*`np.array`*) – input array
- **`y_center`** (*`int, optional`*) – pixel y coordinate of the nrgf center. Defaults to None (image y center).
- **`x_center`** (*`int, optional`*) – pixel x coordinate of the nrgf center. Defaults to (image x center).
- **`rho_min`** (*`int, optional`*) – minimun radius in pixels to perform nrgf to. Defaults to None (radius 0).

- **step** (*int, optional*) – step to which apply the nrgf from center, in pixels. Defaults to 1 pixel.
- **phi_to_mean** (*list[float, float], optional*) – polar angle to calculate the mean value from. Defaults to [0, 360].
- **output_phi** (*list[float, float], optional*) – polar angle to include in output data. Defaults to [0, 360].

Returns
nrgf-filtered input data

Return type
np.array

micropolarray.processing.nrgf.**roi_from_polar**(*data: array*, *center: list | None = None*, *rho: list | None = None*, *theta=[0, 360]*, *fill: float = 0.0*, *return_boolean: bool = False*, *include_superpixels: bool = True*) → array

Returns the input array in a circular selection, otherwise an arbitrary number. If a pixel is not in the selection the ENTIRE superpixel is considered out of selection. If return_boolean is True then a boolean array is returned instead (useful for mean/stdev operations).

Parameters

- **data** (*np.array*) – input data
- **center** (*list, optional*) – pixel coordinates of the circle center. Defaults to None (image center).
- **rho** (*list, optional*) – radius to exclude. Defaults to None (center to image border).
- **theta** (*list, optional*) – polar selection angle, 0 is horizonta, anti-clockwise direction. Defaults to [0, 360].
- **fill** (*float, optional*) – number to fill the outer selection. Defaults to 0.0.
- **return_boolean** (*bool, optional*) – if set to true, function returns a boolean array of the roi. Defaults to False.
- **include_superpixels** (*bool, optional*) – if set to true then exclude entire superpixel if one pixel is in the occulter

Returns
array containing the input data inside the selection, and fill otherwise

Return type
np.array

micropolarray.processing.nrgf.**sigmoid**(*x*, *max*, *min*, *slope*, *intercept*)

micropolarray.processing.nrgf.**tile_double**(*a*)

1.1.1.11 micropolarray.processing.rebin module

micropolarray.processing.rebin.**micropolarray_jitrebin**(*data*, *new_height*, *new_width*, *binning=2*)

Fast rebinning function for the micropolarray image. Needs to be wrapped to print info.

micropolarray.processing.rebin.**micropolarray_jitrebin_old**(*data*, *height*, *width*, *binning=2*)

Fast rebinning function for the micropolarray image.

micropolarray.processing.rebin.**micropolarray_rebin**(*data: ndarray*, *binning=2*)

Wrapper for the faster rebinning donw with numba. First deletes last row/column until binning is possible, then calls binning on the result shape.

Parameters

- **data** (*np.ndarray*) – data to rebin
- **height** (*int*) – lenght of first axis
- **width** (*int*) – lenght of second axis
- **binning** (*int, optional*) – Binning to be performed. Defaults to 2.

Returns

binned data, trimmed if necessary

Return type

ndarray

micropolarray.processing.rebin.**print_trimming_info**(*height*, *width*, *new_height*, *new_width*)

micropolarray.processing.rebin.**standard_jitrebin**(*data*, *new_height*, *new_width*, *binning=2*)

micropolarray.processing.rebin.**standard_rebin**(*data*, *binning: int*) → array

Rebins the data, binned each binningxbinning.

Parameters

- **image** (*np.array*) – data to be binned
- **binning** (*int*) – binning to be applied. A value of 2 will result in a 2x2 binning (1 pixel is a sum of 4 neighbour pixels)

Raises

KeyError – cannot divide image height/width by the binning value

Returns

binned data

Return type

np.array

micropolarray.processing.rebin.**trim_to_match_2xbinning**(*height: int*, *width: int*, *binning: int*)

Deletes the last image pixels until superpixel binning is compatible with new dimensions

Parameters

- **height** (*int*) – image height
- **width** (*int*) – image width
- **binning** (*int*) – image binning

Returns

image new height and width

Return type
int, int

`micropolarray.processing.rebin.`**`trim_to_match_binning`**`(`*height*, *width*, *binning*, *verbose=True*`)`

Deletes the last image pixels until simple binning is compatible with new dimensions

Parameters

- **`height`** (*`int`*) – image height
- **`width`** (*`int`*) – image width
- **`binning`** (*`int`*) – image binning
- **`verbose`** (*`bool, optional`*) – warns user of trimming. Defaults to True.

Returns
image new height and width

Return type
int, int

1.1.1.12 micropolarray.processing.shift module

`micropolarray.processing.shift.`**`shift`**`(`*data: ndarray*, *y: int*, *x: int*`)`

`micropolarray.processing.shift.`**`shift_micropol`**`(`*data: ndarray*, *y: int*, *x: int*`)`

Splits the image into single polarizations, shifts each of them by y,x and then merges them back.

Parameters

- **`data`** (*`np.ndarray`*) – array to shift
- **`y`** (*`int`*) – vertical shift (positive inside the image)
- **`x`** (*`int`*) – horizontal shift (positive inside the image)

Returns
shifted array

Return type
np.ndarray

1.1.1.13 Module contents

1.2 Submodules

1.3 micropolarray.cameras module

`class micropolarray.cameras.`**`Antarticor`**

Bases: `object`

`class micropolarray.cameras.`**`Camera`**

Bases: `object`

occulter_mask(*overoccult: int = 0*, *rmax: int | None = None*) → array

Returns an array of True inside the roi, False elsewhere. Useful for mean/std operations (where=occulter_mask).

Parameters

- **overoccult** (*int, optional*) – Pixels to overoccult. Defaults to 15.
- **rmax** (*int, optional*) – Maximum r of the ROI. Defaults to image nearest border.

Returns
Boolean roi array

Return type
np.array

occulter_roi(*data: array*, *fill: float = 0.0*, *overoccult: int = 0*) → array

Returns the array in the polar ROI, else fill

Parameters

- **data** (*np.array*) – Input array
- **fill** (*float, optional*) – Value for filling. Defaults to 0.0.
- **overoccult** (*int, optional*) – Pixels to overoccult. Defaults to 0.

Returns
Array if in ROI, fill elsewhere

Return type
np.array

class micropolarray.cameras.**Kasi**

Bases: *Camera*

class micropolarray.cameras.**PolarCam**

Bases: *Camera*

1.4 micropolarray.image module

class micropolarray.image.**Image**(*initializer: str | np.ndarray | Image*, *averageimages: bool = True*)

Bases: `object`

Basic image class. Can be initialized from a filename, a filenames list, a numpy array or another Image instance. If multiple filenames are provided, will perform the mean of them unless averageimages is False.

property data: ndarray

save_as_fits(*filename: str*, *fixto: str[float, float] = None*)

Saves image as fits with current header

Parameters

- **filename** (*str*) – output filename
- **fixto** (*str[float, float], optional*) – Set a maximum and minimum range for the data. Defaults to None.

Raises
ValueError – filename does not end with ".fits"

save_as_raw(*filename: str*)

Saves the image as a raw binary file

Parameters

filename (*str*) – output filename

Raises

ValueError – filename does not end with ".raw"

show(*cmap='Greys_r'*, *vmin=None*, *vmax=None*) → tuple

Shows the image data

Parameters

- **cmap** (*str, optional*) – figure colorbar. Defaults to "Greys_r".
- **vmin** (*_type_, optional*) – Minimum value to plot. Defaults to None.
- **vmax** (*_type_, optional*) – Maximum value to plot. Defaults to None.

Returns

fig, ax tuple as returned by matplotlib.pyplot.subplots

Return type

tuple

show_histogram(*split_pols=False*, ***kwargs*) → tuple

Print the histogram of the flattened image data

Parameters

****kwargs** (*int, optional*) – arguments to pass to numpy.histogram(), like bins and range.

Returns

fig, ax tuple as returned by matplotlib.pyplot.subplots

Return type

tuple

1.5 micropolarray.micropol_image module

class micropolarray.micropol_image.**MicropolImage**(*initializer: str | np.ndarray | list | MicropolImage*, *angle_dic: dict = None*, *dark: MicropolImage = None*, *flat: MicropolImage = None*, *averageimages: bool = True*)

Bases: *Image*

Micro-polarizer array image class. Can be initialized from a 2d array, a list of 1 or more file names (use the boolean keyword averageimages to select if sum or average is taken) or another MicropolImage. Dark and flat micropolarray images can also be provided to automatically correct the result.

property AoLP: *PolParam*

property DoLP: *PolParam*

property I: *PolParam*

property Q: *PolParam*

property U: *PolParam*

clean_hot_pixels(*flagged_hot_pix_map: MicropolImage*)

Returns a copy of the image with gaussian smeared pixels where flagged_hot_pix_map == 1.

Parameters
`flagged_hot_pix_map` (`MicropolImage`) – hot pixels map.

Returns
copy of the original image, gaussian smeared where flagged_hot_pix_map == 1

Return type
MicropolImage

congrid(*newdim_y: int, newdim_x: int*) → *MicropolImage*

correct_flat(*flat: MicropolImage*) → *MicropolImage*

Normalizes the flat and uses it to correStokes_vecct the image.

Parameters
`flat` (`MicropolImage`) – flat image, does not need to be normalized.

Returns
copy of input image corrected by flat

Return type
MicropolImage

correct_ifov() → *MicropolImage*

Corrects differences in single pixels fields of view inside each superpixel

Returns
image with data corrected for field of view differences

Return type
MicropolImage

demodulate(*demodulator: Demodulator, demosaicing: bool = False*) → *MicropolImage*

Returns a MicropolImage with polarization parameters calculated from the demodulation tensor provided.

Parameters

- `demodulator` (`Demodulator`) – Demodulator object containing the demodulation tensor components (see processing.new_demodulation)
- `demosaicing` (`bool, optional`) – wether to apply demosaicing to the image or not. Set it to False if demodulation matrices have half the dimension of the image. Defaults to True.

Raises
`ValueError` – raised if image and demodulator do not have the same dimension, for example in case of different binning

Returns
copy of the input imagreturn e with I, Q, U, pB, DoLP, AoLP calculated from the demodulation tensor.

Return type
MicropolImage

demosaic(*demosaic_mode='adjacent'*) → *MicropolImage*

Returns a demosaiced copy of the image with updated polarization parameters. Demoisacing is done IN PLACE and using the THEORETICAL MATRIX. If demodulation and demosaicing are required, please use demodulate(demosaic=True)

Parameters
`demosaic_mode` (*str, optional*) – demosaicing mode (see processing.demosaic). Defaults to "adjacent".

Returns
demosaiced image

Return type
MicropolImage

first_call = True

mask_occulter(*y: int = 919, x: int = 941, r: int = 536, overoccult: int = 0*) → None

Masks occulter for all image parameters

Parameters

- **y** (*int, optional*) – Occulter y position. Defaults to PolarCam().occulter_pos_last[0].
- **x** (*int, optional*) – Occulter x position. Defaults to PolarCam().occulter_pos_last[1].
- **r** (*int, optional*) – Occulter radius. Defaults to PolarCam().occulter_pos_last[2].
- **overoccult** (*int, optional*) – Pixels to overoccult. Defaults to 0.
- **camera** (*_type_, optional*) – Camera image type. Defaults to PolarCam().

Returns
None

property pB: *PolParam*

property pol0: *PolParam*

property pol45: *PolParam*

property pol90: *PolParam*

property pol_45: *PolParam*

property polparam_list: list

rebin(*binning: int*) → *MicropolImage*

Rebins the micropolarizer array image, binned each binningxbinning. Sum bins by default.

Parameters
binning (*int*) – binning to perform. A value of n will be translated in a nxn binning.

Raises
ValueError – negative binning provided

Returns
copy of the input image, rebinned.

Return type
MicropolImage

rotate(*angle: float*) → *MicropolImage*

Rotates an image of angle degrees, counter-clockwise.

save_all_pol_params_as_fits(*filename: str*) → None

Saves the image and all polarization parameters as fits file with the same name

Parameters
filename (*str*) – filename of the output image. Will be saved as filename_[I, Q, U, pB, AoLP, DoLP].fits

Raises
ValueError – filename is not a valid .fits file

save_demosaiced_images_as_fits(*filename: str*, *fixto: list[float, float] = None*) → None

Saves the four demosaiced images as fits files

Parameters

- **filename** (*str*) – filename of the output image. The four images will be saved as filename_POLXX.fits
- **fixto** (*list[float, float], optional*) – set the minimum and maximum value for the output images. Defaults to None.

Raises
ValueError – an invalid file name is provided

save_param_as_fits(*polparam: str*, *filename: str*, *fixto: list[float, float] = None*) → None

Saves chosen polarization parameter as a fits file

Parameters

- **polparam** (*str*) – polarization parameter to save. Can be one among [I, Q, U, pB, AoLP, DoLP]
- **filename** (*str*) – filename of the output image.
- **fixto** (*list[float, float], optional*) – set the minimum and maximum value for the output images. Defaults to None.

Raises
ValueError – filename is not a valid .fits file

save_single_pol_images(*filename: str*, *fixto: list[float, float] = None*) → None

Saves the four polarized images as fits files

Parameters

- **filename** (*str*) – filename of the output image. The four images will be saved as filename_POLXX.fits
- **fixto** (*list[float, float], optional*) – set the minimum and maximum value for the output images. Defaults to None.

Raises
ValueError – an invalid file name is provided

shift(*y: int*, *x: int*) → *MicropolImage*

Shifts image by y, x pixels and fills with 0 the remaining space. Positive numbers for up/right shift and negative for down/left shift. Image is split into polarizations, each one is shifted, then they are merged again.

Parameters

- **y** (*int*) – vertical shift in pix
- **x** (*int*) – horizontal shift in pix

Returns
shifted image copied from the original

Return type
MicropolImage

show_demo_images(*cmap='Greys_r'*, *vmin=None*, *vmax=None*, ***kwargs*)

Plots the four demosaiced images.

Parameters

- **cmap** (`str, optional`) – colormap for the plot. Defaults to "Greys_r".
- ****kwargs** – arguments passed to matplotlib.pyplot.imshow.

Returns
a (figure, axis) couple same as matplotlib.pyplot.subplots

Return type
tuple

show_histogram(*split_pols: bool = True*, ***kwargs*)

Print the histogram of the flattened image data

Parameters

- **split_pols** (`bool, optional`) – Whether to overplot histograms of same family pixels separately. Defaults to False.
- ****kwargs** (`int, optional`) – arguments to pass to numpy.histogram(), like bins and range.

Returns
fig, ax tuple as returned by matplotlib.pyplot.subplots

Return type
tuple

show_pol_param(*polparam: str*, *cmap='Greys_r'*, *vmin=None*, *vmax=None*, ***kwargs*)

Plots a single polarization parameter given as input

Parameters

- **polparam** (`str`) – image PolParam containing the parameter to plot. Can be one among [I, Q, U, pB, AoLP, DoLP]
- **cmap** (`str, optional`) – colormap for the plot. Defaults to "Greys_r".
- ****kwargs** – arguments passed to matplotlib.pyplot.imshow.

Returns
a (figure, axis) couple same as matplotlib.pyplot.subplots

Return type
tuple

show_single_pol_images(*cmap='Greys_r'*, ***kwargs*)

Plots the four polarizations images.

Parameters

- **cmap** (`str, optional`) – colormap for the plot. Defaults to "Greys_r".
- ****kwargs** – arguments passed to matplotlib.pyplot.imshow.

Returns
a (figure, axis) couple same as matplotlib.pyplot.subplots

Return type
tuple

`show_with_pol_params`(*cmap='Greys_r'*) → tuple

Returns a tuple containing figure and axis of the plotted data, and figure and axis of polarization parameters (3x2 subplots). User must callplt.show after this is called.

Parameters
cmap (*str, optional*) – colormap string. Defaults to "Greys_r".

Returns
a (figure, axis, figure, axis) couple same as matplotlib.pyplot.subplots for the image data and another for the six polarization parameters

Return type
tuple

`property single_pol_subimages`

`subtract_dark`(*dark:* MicropolImage) → *MicropolImage*

Correctly subtracts the input dark image from the image

Parameters
dark (MicropolImage) – dark to subtract

Returns
copy of input image with dark subtracted

Return type
MicropolImage

`class micropolarray.micropol_image.PolParam`(*ID: str*, *data: ndarray*, *title: str*, *measure_unit: str*, *fix_data: bool = False*)

Bases: `object`

Auxiliary class for polarization parameters.

Members:
ID (str): parameter identifier data (np.array): parameter image as numpy 2D array title (str): brief title of the parameter, useful for plotting measure_unit (str): initial measure units of the parameter fix_data (bool): controls whether data has to be constrained to [0, 4096] interval (not implemented yet)

`ID: str`

`data: ndarray`

`fix_data: bool = False`

`measure_unit: str`

`title: str`

`micropolarray.micropol_image.set_default_angles`(*angles_dic: dict*)

Sets the default micropolarizer orientations for images.

Parameters
angles_dic (*dict*) – dictionary {value : pos} where value is the angle in degrees from -90 to 90 and pos is the pixel position in superpixel, from 0 to 3 (position [y, x], fast index x)

1.6 micropolarray.polarization_functions module

micropolarray.polarization_functions.**AoLP**(*Stokes_vec_components*)

Angle of linear polarization in [rad]

micropolarray.polarization_functions.**DoLP**(*Stokes_vec_components*)

Degree of linear polarization in [%]

micropolarray.polarization_functions.**normalize2pi**(*angles_list*)

Normalizes the input angle list in the -90,90 range

Parameters

angles_list (*_type_*) – list of input angles in degrees

Returns

normalized angles

Return type

type

micropolarray.polarization_functions.**pB**(*Stokes_vec_components*)

Polarized brighness in [%]

1.7 micropolarray.utils module

micropolarray.utils.**align_keywords_and_data**(*header*, *data*, *sun_center*, *platescale*, *binning=1*)

Fixes antarticor keywords and data to reflect each other.

Parameters

- **header** (*dict*) – fits file header
- **data** (*ndarray*) – data as np array
- **platescale** (*float*) – plate scale in arcsec/pixel
- **binning** (*int, optional*) – binning applied to image. Defaults to 1 (no binning).

Returns

new fixed header and data

Return type

header, data

micropolarray.utils.**fix_data**(*data: array*, *min*, *max*)

micropolarray.utils.**get_Bsun_units**(*diffuser_I: float*, *texp_image: float = 1.0*, *texp_diffuser: float = 1.0*) → float

Returns the conversion unit for expressing brightness in units of sun brightness. Usage is data [units of B_sun] = data[DN] * get_Bsun_units(mean_Bsun_brightness, texp_image, texp_diffuser)

Parameters

- **mean_sun_brightness** (*float*) – diffuser mean in DN.
- **texp_image** (*float, optional*) – image exposure time. Defaults to 1.0.
- **texp_diffuser** (*float, optional*) – diffuser exposure time. Defaults to 1.0.

Returns
Bsun units conversion factor

Return type
float

`micropolarray.utils.`**`get_malus_normalization`**(*four_peaks_images*, *show_hist=False*)

`micropolarray.utils.`**`mean_minus_std`**(*data: array*, *stds_n: int = 1*) → float

Returns the value at the mean - standard deviation for the input data

Parameters

- **data** (*np.array*) – input data
- **stds_n** (*int, optional*) – number of standard deviations. Defaults to 1.

Returns
mean value - n*stdevs

Return type
float

`micropolarray.utils.`**`mean_plus_std`**(*data: array*, *stds_n: int = 1*) → float

Returns the value at the mean + standard deviation for the input data

Parameters

- **data** (*np.array*) – input data
- **stds_n** (*int, optional*) – number of standard deviations. Defaults to 1.

Returns
mean value + n*stdevs

Return type
float

`micropolarray.utils.`**`median_minus_std`**(*data: array*, *stds_n: int = 1*) → float

Returns the value at the median - median deviation for the input data

Parameters

- **data** (*np.array*) – input data
- **stds_n** (*int, optional*) – number of standard deviations. Defaults to 1.

Returns
median value - n*mediandevs

Return type
float

`micropolarray.utils.`**`median_plus_std`**(*data: array*, *stds_n: int = 1*) → float

Returns the value at the median + median deviation for the input data

Parameters

- **data** (*np.array*) – input data
- **stds_n** (*int, optional*) – number of standard deviations. Defaults to 1.

Returns
median value + n*mediandevs

Return type
float

`micropolarray.utils.`**`normalize2pi`**`(`*angles_list*`)`

`micropolarray.utils.`**`sigma_DN`**`(`*pix_DN*`)`

`micropolarray.utils.`**`timer`**`(`*func*`)`

Use this to time function execution

Parameters
`func` (*`function`*) – function of which to measure execution time

1.8 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

G

I

K

L

M

N

O

P

Q

R

S

T

U