

---

# **micropolarray**

**Herve Haudemand**

**May 08, 2024**



**CONTENTS:**

<b>1</b>	<b>micropolarray package</b>	<b>1</b>
1.1	Subpackages	1
1.1.1	micropolarray.processing package	1
1.1.1.1	Submodules	1
1.1.1.2	micropolarray.processing.chen_wan_liang_calibration module	1
1.1.1.3	micropolarray.processing.congrid module	2
1.1.1.4	micropolarray.processing.convert module	2
1.1.1.5	micropolarray.processing.demodulation module	4
1.1.1.6	micropolarray.processing.demodulation_errors module	6
1.1.1.7	micropolarray.processing.demosaic module	7
1.1.1.8	micropolarray.processing.image_cleaning module	7
1.1.1.9	micropolarray.processing.linear_roi module	7
1.1.1.10	micropolarray.processing.nrgf module	9
1.1.1.11	micropolarray.processing.rebin module	11
1.1.1.12	micropolarray.processing.shift module	12
1.1.1.13	Module contents	12
1.2	Submodules	12
1.3	micropolarray.cameras module	12
1.4	micropolarray.image module	13
1.5	micropolarray.micropol_image module	14
1.6	micropolarray.polarization_functions module	20
1.7	micropolarray.utils module	20
1.8	Module contents	22
<b>2</b>	<b>Indices and tables</b>	<b>23</b>
	<b>Python Module Index</b>	<b>25</b>
	<b>Index</b>	<b>27</b>



## 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\_orientations, filenames\_list, micropol\_phases\_previsions, output\_dir, occulter=True, dark\_filename=None, flat\_filename=None)*

Performs calibration from Chen-Wang-Liang paper 2014

#### Parameters

- **polarizer\_orientations** (*list[float]*) – List of polarizer orientations
- **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*) – whether 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 length

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

`micropolararray.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 micropolararray.processing.congrid module

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

Reshapes the data into any new length and width

### Parameters

- **a** (*np.array*) – data to be reshaped
- **newdims** (*tuple* / *list*) – new length and width
- **kind** (*str*, *optional*) – interpolation type. Defaults to “linear”.

### Returns

numpy array of congrided image

### Return type

*ndarray*

`micropolararray.processing.congrid.micropolararray_jitcongrid(data, width, height, scale)`

### 1.1.1.4 micropolararray.processing.convert module

`micropolararray.processing.convert.average_rawfiles_to_fits(filenamees: list, new_filename: str, height: int, width: int)`

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

### Parameters

- **filenamees** (*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(filenamees, new_filename, height, width)`

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

#### Parameters

- **filenamees** (*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(filenamees: 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

- **filenamees** (*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 MicropollImage 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: ndarraymicropolarray.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_errors,  
                                                                    splitted_occultor_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 micropolararray.processing.demosaic module

`micropolararray.processing.demosaic.demosaic(image_data, option='adjacent')`

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

`micropolararray.processing.demosaic.demosaicadjacent(data)`

`micropolararray.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]`

`micropolararray.processing.demosaic.merge_polarizations(single_pol_images: ndarray)`

`micropolararray.processing.demosaic.split_polarizations(data: ndarray)`

### 1.1.1.8 micropolararray.processing.image\_cleaning module

`micropolararray.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 threshold

#### Returns

Otsu's threshold

#### Return type

float

`micropolararray.processing.image_cleaning.get_hot_pixels(image, threshold=100)`

`micropolararray.processing.image_cleaning.reject_outliers(data, m=2.0)`

`micropolararray.processing.image_cleaning.remove_outliers_simple(original, neighbours=2)`

EXPERIMENTAL DO NOT USE, for improving fitting on occulter position

### 1.1.1.9 micropolararray.processing.linear\_roi module

`micropolararray.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_occulters_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*) – minimum 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 horizontal, 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 down 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*) – length of first axis
- **width** (*int*) – length 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  $\text{binning} \times \text{binning}$ .

#### 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*: MicropollImage)

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

**Parameters**

**flagged\_hot\_pix\_map** (MicropollImage) – hot pixels map.

**Returns**

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

**Return type**

*MicropollImage*

**congrid**(*newdim\_y*: int, *newdim\_x*: int) → *MicropollImage*

**correct\_flat**(*flat*: MicropollImage) → *MicropollImage*

Normalizes the flat and uses it to correct the image.

**Parameters**

**flat** (MicropollImage) – flat image, does not need to be normalized.

**Returns**

copy of input image corrected by flat

**Return type**

*MicropollImage*

**correct\_ifov**() → *MicropollImage*

Corrects differences in single pixels fields of view inside each superpixel

**Returns**

image with data corrected for field of view differences

**Return type**

*MicropollImage*

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

Returns a MicropollImage 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) – whether 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 image with I, Q, U, pB, DoLP, AoLP calculated from the demodulation tensor.

**Return type**

*MicropollImage*

**demosaic**(*demosaic\_mode*='adjacent') → *MicropollImage*

Returns a demosaiced copy of the image with updated polarization parameters. Demosaicing 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**

*MicropollImage*

**first\_call** = **True**

**mask\_occult**(*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*) → *MicropollImage*

Rebins the micropolarizer array image, binned each `binning`x`binning`. 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**

*MicropollImage*

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

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\_demosaiiced\_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*) → *MicropollImage*

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**

*MicropollImage*

**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 call plt.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 brightness 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

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### m

- `micropolarray`, 22
- `micropolarray.cameras`, 12
- `micropolarray.image`, 13
- `micropolarray.micropol_image`, 14
- `micropolarray.polarization_functions`, 20
- `micropolarray.processing`, 12
- `micropolarray.processing.chen_wan_liang_calibration`,  
1
- `micropolarray.processing.congrid`, 2
- `micropolarray.processing.convert`, 2
- `micropolarray.processing.demodulation`, 4
- `micropolarray.processing.demodulation_errors`,  
6
- `micropolarray.processing.demosaic`, 7
- `micropolarray.processing.image_cleaning`, 7
- `micropolarray.processing.linear_roi`, 7
- `micropolarray.processing.nrgf`, 9
- `micropolarray.processing.rebin`, 11
- `micropolarray.processing.shift`, 12
- `micropolarray.utils`, 20



## A

`align_keywords_and_data()` (in module `micropolar-ray.utils`), 20  
`Antarticator` (class in `micropolararray.cameras`), 12  
`AoLP` (`micropolararray.micropol_image.MicropollImage` property), 14  
`AoLP()` (in module `micropolar-ray.polarization_functions`), 20  
`auto_threshold()` (in module `micropolar-ray.processing.image_cleaning`), 7  
`average_rawfiles_to_fits()` (in module `micropolar-ray.processing.convert`), 2

## B

`bresenham()` (in module `micropolar-ray.processing.linear_roi`), 7

## C

`calculate_chen_wan_lian_calibration()` (in module `micropolar-ray.processing.chen_wan_liang_calibration`), 1  
`calculate_demodulation_tensor()` (in module `micropolararray.processing.demodulation`), 4  
`Camera` (class in `micropolararray.cameras`), 12  
`chen_wan_liang_calibration()` (in module `micropolar-ray.processing.chen_wan_liang_calibration`), 2  
`Cij` (`micropolararray.processing.demodulation.Demodulator` property), 4  
`clean_hot_pixels()` (`micropolar-ray.micropol_image.MicropollImage` method), 14  
`compute_demodulation_by_chunk()` (in module `micropolararray.processing.demodulation`), 6  
`congrid()` (in module `micropolar-ray.processing.congrid`), 2  
`congrid()` (`micropolar-ray.micropol_image.MicropollImage` method), 15

`convert_rawfile_to_fits()` (in module `micropolar-ray.processing.convert`), 2  
`convert_set()` (in module `micropolar-ray.processing.convert`), 3  
`correct_flat()` (`micropolar-ray.micropol_image.MicropollImage` method), 15  
`correct_ifov()` (`micropolar-ray.micropol_image.MicropollImage` method), 15

## D

`data` (`micropolararray.image.Image` property), 13  
`data` (`micropolararray.micropol_image.PolParam` attribute), 19  
`DDA()` (in module `micropolararray.processing.linear_roi`), 7  
`demodulate()` (`micropolar-ray.micropol_image.MicropollImage` method), 15  
`Demodulator` (class in `micropolar-ray.processing.demodulation`), 4  
`demosaic()` (in module `micropolar-ray.processing.demosaic`), 7  
`demosaic()` (`micropolar-ray.micropol_image.MicropollImage` method), 15  
`demosaicadjacent()` (in module `micropolar-ray.processing.demosaic`), 7  
`demosaicmean()` (in module `micropolar-ray.processing.demosaic`), 7  
`DoLP` (`micropolararray.micropol_image.MicropollImage` property), 14  
`DoLP()` (in module `micropolar-ray.polarization_functions`), 20

## E

`eff` (`micropolararray.processing.demodulation.Demodulator` property), 4

## F

`find_occulters_hough()` (in module `micropolar-`

`ray.processing.nrgf`), 9  
`find_occulters_position()` (in module `micropolar-  
ray.processing.nrgf`), 9  
`first_call` (*micropolar-  
ray.micropol\_image.MicropollImage* attribute),  
16  
`fix_data` (*micropolarray.micropol\_image.PolParam* at-  
tribute), 19  
`fix_data()` (in module `micropolarray.utils`), 20  
`flip()` (*micropolarray.processing.demodulation.Demodulator*  
method), 4

## G

`get_Bsun_units()` (in module `micropolarray.utils`), 20  
`get_error_on_AoLP()` (in module `micropolar-  
ray.processing.demodulation_errors`), 6  
`get_error_on_DoLP()` (in module `micropolar-  
ray.processing.demodulation_errors`), 6  
`get_error_on_pB()` (in module `micropolar-  
ray.processing.demodulation_errors`), 6  
`get_error_on_Stokes()` (in module `micropolar-  
ray.processing.demodulation_errors`), 6  
`get_hot_pixels()` (in module `micropolar-  
ray.processing.image_cleaning`), 7  
`get_malus_normalization()` (in module `micropolar-  
ray.utils`), 21

## I

`I` (*micropolarray.micropol\_image.MicropollImage* prop-  
erty), 14  
`ID` (*micropolarray.micropol\_image.PolParam* attribute),  
19  
`Image` (class in `micropolarray.image`), 13

## K

`Kasi` (class in `micropolarray.cameras`), 13

## L

`linear_roi()` (in module `micropolar-  
ray.processing.linear_roi`), 8  
`linear_roi_from_polar()` (in module `micropolar-  
ray.processing.linear_roi`), 8

## M

`Malus()` (in module `micropolar-  
ray.processing.demodulation`), 4  
`map_polar_coordinates()` (in module `micropolar-  
ray.processing.nrgf`), 9  
`mask_occulters()` (*micropolar-  
ray.micropol\_image.MicropollImage* method),  
16  
`mean_minus_std()` (in module `micropolarray.utils`), 21  
`mean_plus_std()` (in module `micropolarray.utils`), 21

`measure_unit` (*micropolar-  
ray.micropol\_image.PolParam* attribute),  
19  
`median_minus_std()` (in module `micropolarray.utils`),  
21  
`median_plus_std()` (in module `micropolarray.utils`), 21  
`merge_polarizations()` (in module `micropolar-  
ray.processing.demosaic`), 7  
`merge_rawfiles_to_fits()` (in module `micropolar-  
ray.processing.convert`), 3  
`micropolarray`  
module, 22  
`micropolarray.cameras`  
module, 12  
`micropolarray.image`  
module, 13  
`micropolarray.micropol_image`  
module, 14  
`micropolarray.polarization_functions`  
module, 20  
`micropolarray.processing`  
module, 12  
`micropolarray.processing.chen_wan_liang_calibration`  
module, 1  
`micropolarray.processing.congrid`  
module, 2  
`micropolarray.processing.convert`  
module, 2  
`micropolarray.processing.demodulation`  
module, 4  
`micropolarray.processing.demodulation_errors`  
module, 6  
`micropolarray.processing.demosaic`  
module, 7  
`micropolarray.processing.image_cleaning`  
module, 7  
`micropolarray.processing.linear_roi`  
module, 7  
`micropolarray.processing.nrgf`  
module, 9  
`micropolarray.processing.rebin`  
module, 11  
`micropolarray.processing.shift`  
module, 12  
`micropolarray.utils`  
module, 20  
`micropolarray_jitcongrid()` (in module `micropolar-  
ray.processing.congrid`), 2  
`micropolarray_jitrebin()` (in module `micropolar-  
ray.processing.rebin`), 11  
`micropolarray_jitrebin_old()` (in module `micropo-  
larray.processing.rebin`), 11  
`micropolarray_rebin()` (in module `micropolar-  
ray.processing.rebin`), 11



**MicropolImage** (class in micropolarray.micropol\_image), 14  
**MicropolImageError** (class in micropolarray.processing.demodulation\_errors), 6  
**module**  
     micropolarray, 22  
     micropolarray.cameras, 12  
     micropolarray.image, 13  
     micropolarray.micropol\_image, 14  
     micropolarray.polarization\_functions, 20  
     micropolarray.processing, 12  
     micropolarray.processing.chen\_wan\_liang\_calibration, 1  
     micropolarray.processing.congrid, 2  
     micropolarray.processing.convert, 2  
     micropolarray.processing.demodulation, 4  
     micropolarray.processing.demodulation\_errors, 6  
     micropolarray.processing.demosaic, 7  
     micropolarray.processing.image\_cleaning, 7  
     micropolarray.processing.linear\_roi, 7  
     micropolarray.processing.nrgf, 9  
     micropolarray.processing.rebin, 11  
     micropolarray.processing.shift, 12  
     micropolarray.utils, 20  
**N**  
**normalize2pi()** (in module micropolarray.polarization\_functions), 20  
**normalize2pi()** (in module micropolarray.utils), 22  
**nparr\_from\_binary()** (in module micropolarray.processing.convert), 3  
**nrgf()** (in module micropolarray.processing.nrgf), 9  
**O**  
**occulter\_mask()** (micropolarray.cameras.Camera method), 12  
**occulter\_roi()** (micropolarray.cameras.Camera method), 13  
**P**  
**pB** (micropolarray.micropol\_image.MicropollImage property), 16  
**pB()** (in module micropolarray.polarization\_functions), 20  
**phi** (micropolarray.processing.demodulation.Demodulator property), 4  
**pol10** (micropolarray.micropol\_image.MicropollImage property), 16  
**pol145** (micropolarray.micropol\_image.MicropollImage property), 16  
**pol190** (micropolarray.micropol\_image.MicropollImage property), 16  
**pol\_45** (micropolarray.micropol\_image.MicropollImage property), 16  
**PolarCam** (class in micropolarray.cameras), 13  
**PolParam** (class in micropolarray.micropol\_image), 19  
**polparam\_list** (micropolarray.micropol\_image.MicropollImage property), 16  
**print\_trimming\_info()** (in module micropolarray.processing.rebin), 11  
**Q**  
**q\_calibration** (micropolarray.micropol\_image.MicropollImage property), 14  
**R**  
**rebin()** (micropolarray.micropol\_image.MicropollImage method), 16  
**rebin()** (micropolarray.processing.demodulation.Demodulator method), 4  
**reject\_outliers()** (in module micropolarray.processing.image\_cleaning), 7  
**remove\_outliers\_simple()** (in module micropolarray.processing.image\_cleaning), 7  
**roi\_from\_polar()** (in module micropolarray.processing.nrgf), 10  
**rot90()** (micropolarray.processing.demodulation.Demodulator method), 4  
**rotate()** (micropolarray.micropol\_image.MicropollImage method), 16  
**S**  
**save\_all\_pol\_params\_as\_fits()** (micropolarray.micropol\_image.MicropollImage method), 16  
**save\_as\_fits()** (micropolarray.image.Image method), 13  
**save\_as\_raw()** (micropolarray.image.Image method), 13  
**save\_demosaiced\_images\_as\_fits()** (micropolarray.micropol\_image.MicropollImage method), 17  
**save\_param\_as\_fits()** (micropolarray.micropol\_image.MicropollImage method), 17  
**save\_single\_pol\_images()** (micropolarray.micropol\_image.MicropollImage method), 17  
**set\_default\_angles()** (in module micropolarray.micropol\_image), 19  
**shift()** (in module micropolarray.processing.shift), 12  
**shift()** (micropolarray.micropol\_image.MicropollImage method), 17

`shift_micropol()` (in module `micropolararray.processing.shift`), 12

`show()` (`micropolarray.image.Image` method), 14

`show()` (`micropolarray.processing.demodulation.Demodulator` method), 4

`show_demo_images()` (`micropolararray.micropol_image.MicropollImage` method), 18

`show_histogram()` (`micropolarray.image.Image` method), 14

`show_histogram()` (`micropolararray.micropol_image.MicropollImage` method), 18

`show_pol_param()` (`micropolararray.micropol_image.MicropollImage` method), 18

`show_single_pol_images()` (`micropolararray.micropol_image.MicropollImage` method), 18

`show_with_pol_params()` (`micropolararray.micropol_image.MicropollImage` method), 19

`sigma_DN()` (in module `micropolarray.utils`), 22

`sigmoid()` (in module `micropolarray.processing.nrgf`), 10

`single_pol_subimages` (`micropolararray.micropol_image.MicropollImage` property), 19

`split_polarizations()` (in module `micropolararray.processing.demosaic`), 7

`standard_jitrebin()` (in module `micropolararray.processing.rebin`), 11

`standard_rebin()` (in module `micropolararray.processing.rebin`), 11

`subtract_dark()` (`micropolararray.micropol_image.MicropollImage` method), 19

## T

`three_bytes_to_two_ints()` (in module `micropolararray.processing.convert`), 4

`tile_double()` (in module `micropolararray.processing.nrgf`), 10

`timer()` (in module `micropolarray.utils`), 22

`title` (`micropolarray.micropol_image.PolParam` attribute), 19

`tk` (`micropolarray.processing.demodulation.Demodulator` property), 4

`trim_to_match_2xbinning()` (in module `micropolararray.processing.rebin`), 11

`trim_to_match_binning()` (in module `micropolararray.processing.rebin`), 12

## U

`U` (`micropolarray.micropol_image.MicropollImage` property), 14