Pysiaf Documentation

Release 0.22.0

pysiaf

Mar 01, 2024

CONTENTS:

1 pysiaf **3**

1.1 Functionalities . . . 3
1.2 Where to Find pysiaf . . . 3
1.3 Installation . . . 3
1.3.1 How to install . . . 4
1.3.2 Known installation issue . . . 4
1.4 User Documentation . . . 4
1.4.1 Using sky transforms . . . 5
1.5 Reporting Issues / Contributing . . . 5
1.5.1 Coding and other guidelines . . . 5
1.5.2 How to make a code contribution . . . 5
1.6 Disclaimer . . . 6
1.7 Reference API . . . 6
1.7.1 aperture . . . 6
1.7.2 compare . . . 23
1.7.3 polynomial . . . 26
1.7.4 projection . . . 34
1.7.5 read . . . 35
1.7.6 rotations . . . 38
1.7.7 siaf . . . 46
1.7.8 tools . . . 49
1.7.9 write . . . 53

Python Module Index **55**

Index **57**

SIAF files contain detailed geometric focal plane descriptions and relationships for the science instruments. They are maintained in the JWST/HST PRD (Project Reference Database). pysiaf is a python package to access, interpret, maintain, and generate SIAF, in particular for JWST. Tools for applying the frame transformations, plotting, comparison, and validation are provided.

CHAPTER

ONE

PYSIAF

Handling of Science Instrument Aperture Files (SIAF) for space telescopes

1.1 Functionalities

- Captures current PRD content, i.e. pysiaf includes a copy of the SIAF XML files. These are maintained to be synchronized with the PRD.
- Transformations between the SIAF frames (Detector, Science, Ideal, Telelescope/V) are pre-loaded and easily accessible.
- Tools for plotting, validation, and comparison of SIAF apertures and files.
- Support for implementing transformations between celestial (RA, Dec) and telescope/V (V2, V3) coordinate systems is provided.
- Input/output: reading SIAF XML, writing XML/Excel/csv etc.
- Captures SI source data and code to generate the SIAF apertures
- Standard python package with installation script, unit tests, documentation.
- Supports working with HST SIAF (read-only).

1.2 Where to Find pysiaf

pysiaf is hosted and developed at https://github.com/spacetelescope/pysiaf

1.3 Installation

This package is being developed in a python 3.6 environment.

1.3.1 How to install

pysiaf is available on PyPI (https://pypi.org/project/pysiaf/) and is included in astroconda.

pip install pysiaf

Clone the repository: git clone https://github.com/spacetelescope/pysiaf Install pysiaf: cd pysiaf python setup.py install or pip install .

1.3.2 Known installation issue

If you get an error upon `import pysiaf` that traces back to `import lxml.etree as ET` and states

`ImportError [...] Library not loaded: libxml2.2.dylib Reason: Incompatible library version: etree.[...] requires version 12.0.0 or later, but libxml2.2.dylib provides version 10.0.0`,

this can probably be fixed by downgrading the version of lxml, e.g.

`pip uninstall lxml pip install lxml==3.6.4`

1.4 User Documentation

Example usage:

Check which PRD version is in use: `print(pysiaf.JWST_PRD_VERSION)`

Frame transformations (`det`, `sci`, `idl`, `tel` are supported frames):

```
import pysiaf
instrument = 'NIRISS'

# read SIAFXML
siaf = pysiaf.Siaf(instrument)

# select single aperture by name
nis_cen = siaf['NIS_CEN']

# access SIAF parameters
print('{} V2Ref = {}'.format(nis_cen.AperName, nis_cen.V2Ref))
print('{} V3Ref = {}'.format(nis_cen.AperName, nis_cen.V3Ref))

for attribute in ['InstrName', 'AperShape']:
    print('{} {} = {}'.format(nis_cen.AperName, attribute, getattr(nis_cen, attribute)))


# coordinates in Science frame
sci_x = np.array([0, 2047, 2047, 0])
sci_y = np.array([0, 0, 2047, 2047])

# transform from Science frame to Ideal frame
idl_x, idl_y = nis_cen.sci_to_idl(sci_x, sci_y)
```

1.4.1 Using sky transforms

Transformations to/from `sky` coordinates (RA, Dec) are also supported, but you have to first define and set an attitude matrix that represents the observatory orientation with respect to the celestial sphere. This can be done with `pysiaf.utils.rotations.attitude_matrix`:

```
# find attitude with some coordinates (v2,v3) pointed at (ra, dec) with a given pa
attmat = pysiaf.utils.rotations.attitude_matrix(v2, v3, ra, dec, pa)

# set that attitude for the transforms
nis_cen.set_attitude_matrix(attmat)

# transform from Science frame to Sky frame
sky_ra, sky_dec = nis_cen.sci_to_sky(sci_x, sci_y)
```

Sky coordinates are given in units of degrees RA and Dec.

1.5 Reporting Issues / Contributing

Do you have feedback and feature requests? Is there something missing you would like to see? Please open a new issue or new pull request at https://github.com/spacetelescope/pysiaf for bugs, feedback, or new features you would like to see. If there is an issue you would like to work on, please leave a comment and we will be happy to assist. New contributions and contributors are very welcome! This package follows the STScI Code of Conduct (https://github.com/spacetelescope/pysiaf/blob/master/CODE_OF_CONDUCT.md) strives to provide a welcoming community to all of our users and contributors.

1.5.1 Coding and other guidelines

We strive to adhere to the STScI Style Guides (https://github.com/spacetelescope/style-guides).

1.5.2 How to make a code contribution

The following describes the typical work flow for contributing to the pysiaf project (adapted from https://github.com/spacetelescope/jwql):

1. Do not commit any sensitive information (e.g. STScI-internal path structures, machine names, user names, passwords, etc.) to this public repository. Git history cannot be erased.
2. Create a fork off of the `spacetelescope pysiaf` repository on your personal github space.
3. Make a local clone of your fork.
4. Ensure your personal fork is pointing `upstream` to https://github.com/spacetelescope/pysiaf
5. Open an issue on `spacetelescope pysiaf` that describes the need for and nature of the changes you plan to make. This is not necessary for minor changes and fixes.
6. Create a branch on that personal fork.
7. Make your software changes.
8. Push that branch to your personal GitHub repository, i.e. to `origin`.
9. On the `spacetelescope pysiaf` repository, create a pull request that merges the branch into `spacetelescope:master`.

10. Assign a reviewer from the team for the pull request, if you are allowed to do so.
11. Iterate with the reviewer over any needed changes until the reviewer accepts and merges your branch.
12. Delete your local copy of your branch.

1.6 Disclaimer

All parameter values in pysiaf are subject to change. JWST values are preliminary until the JWST observatory commissioning has concluded.

Distortion and other transformations in pysiaf are of sufficient accuracy for operations, but do not necessarily have science-grade quality. For instance, generally only one filter solution is carried per aperture. For science-grade transformations, please consult the science pipelines and their reference files (see https://jwst-docs.stsci.edu/display/JDAT/JWST+Data+Reduction+Pipeline).

For science observation planning, the focal plane geometry implemented in the latest APT (http://www.stsci.edu/hst/proposing/apt) takes precedence.

The STScI Telescopes Branch provides full support of pysiaf for S&OC operational systems only.

1.7 Reference API

1.7.1 aperture

Module to handle SIAF apertures.

The aperture module defined classes and functions to support working with SIAF apertures. The main class is Aperture, JwstAperture and HstAperture inherit from it. The class methods support transformations between any of the 4 SIAF frames (detector, science/DMS, ideal, V-frame/telescope). Aperture attributes correspond to entries in the SIAF xml files in the Project Reference Database (PRD), and are checked for format compliance. Methods for aperture validation and plotting are provided. The nomenclature is adapted from the JWST SIAF document Cox & Lallo: JWST-STScI-001550.

Authors

- Johannes Sahlmann

Notes

Numerous contributions and code snippets by Colin Cox were incorporated. Some methods were adapted from the jwxml package (https://github.com/mperrin/jwxml). Some of the polynomial transformation code was adapted from (https://github.com/spacetelescope/ramp_simulator/).

class pysiaf.aperture.**Aperture**

A class for aperture definitions of astronomical telescopes.

HST and JWST SIAF entries are supported via class inheritance. Frames, transformations, conventions and property/attribute names are as defined for JWST in JWST-STScI-001550.

Transformations between four coordinate systems (“frames”) are supported:

- Detector (“det”) : units of pixels, according to detector read out axes orientation as defined by SIAF. This system generally differs from the JWST-DMS detector frame definition.

- Science ("sci") : units of pixels, corresponds to DMS coordinate system.
- Ideal ("idl") : units of arcseconds, usually a tangent plane projection with reference point at aperture reference location.
- Telescope or V2/V3 ("tel") : units of arcsecs, spherical coordinate system.

Examples

extract one aperture from a Siaf object: `ap = some_siaf['desired_aperture_name']`

convert pixel coordinates to V2V3 / tel coords (takes pixel coords, returns arcsec): `ap.det_to_tel(1024, 512)`

convert idl coords to sci pixels: `ap.idl_to_sci( 10, 3)`

Frames can also be defined by strings: `ap.convert(1024, 512, from_frame='det', to_frame='tel')`

Get aperture vertices/corners in tel frame: `ap.corners('tel')`

Get the reference point defined in the SIAF: `ap.reference_point('tel')`

plot coordinates in idl frame: `ap.plot('tel')`

There exist methods for all of the possible `{tel,idl,sci,det}_to_{tel,idl,sci,det}` combinations.

Set attributes required for PRD.

`closed_polygon_points`(*to_frame*, *rederive=True*)

Compute closed polygon points of aperture outline. Used for plotting and path generation.

Parameters

to_frame
[str] Name of frame.

rederive
[bool] Whether to rederive vertices from scratch (if True) or use idl values stored in SIAF.

Returns

points_x, points_y
[tuple of numpy arrays] x and y coordinates of aperture vertices

`complement`()

Complement the attributes of an aperture.

This method is useful when generating new apertures. The attributes that will be added are:

- X[Y]IdlVert1[2,3,4]
- X[Y]SciScale

`convert`(*x*, *y*, *from_frame*, *to_frame*)

Convert input coordinates from one frame to another frame.

Parameters

x
[float] first coordinate

y
[float] second coordinate

from_frame
[str] Frame in which x,y are given

to_frame
[str] Frame to transform into

Returns

x', y'
[tuple] Coordinates in to_frame

`corners`(*to_frame*, *rederive=True*)

Return coordinates of the aperture vertices in the specified frame.

The positions refer to the outside corners of the corner pixels, not the pixel centers.

Parameters

to_frame
[str] Frame of returned coordinates

rederive
[bool] If True, the parameters given in the aperture definition file are used to derive the vertices. Otherwise computations are made on the basis of the IdlVert attributes.

Returns

x, y
[tuple of float arrays] coordinates of vertices in to_frame

`correct_for_dva`(*v2_arcsec*, *v3_arcsec*, *verbose=False*)

Apply differential velocity aberration correction to input arrays of V2/V3 coordinates.

Currently only implemented for HST apertures.

Parameters

v2_arcsec
[float] v2 coordinates

v3_arcsec
[float] v3 coordinates

verbose
[bool] verbosity

Returns

v2,v3
[tuple ot floats] Corrected coordinates

`det_to_idl`(**args*)

Detector to ideal frame transformation.

`det_to_raw`(**args*)

Detector to raw frame transformation.

`det_to_sci`(*x_det*, *y_det*, **args*)

Detector to science frame transformation, following Section 4.1 of JWST-STScI-001550.

`det_to_sky`(**args*)

det_to_tel(**args*)

Detector to telescope frame transformation.

detector_transform(*from_system*, *to_system*, *angle_deg=None*, *parity=None*)

Generate transformation model to transform between det <-> sci.

DetSciYAngle_deg, DetSciParity can be defined externally to override aperture attribute.

Parameters

from_system
[str] Originating system

to_system
[str] System to transform into

angle_deg
[float, int] DetSciYAngle

parity
[int] DetSciParity

Returns

x_model, y_model
[tuple of astropy.modeling models]

distortion_transform(*from_system*, *to_system*, *include_offset=True*)

Return polynomial distortion transformation between sci<->idl.

Parameters

from_system
[str] Starting system (e.g. "sci", "idl")

to_system
[str] Ending coordinate system (e.g. "sci" , "idl")

include_offset
[bool] Whether to include the zero-point offset.

Returns

x_model
[astropy.modeling.Model] Correction in x

y_model
[astropy.modeling.Model] Correction in y

dms_corner()

Compute the pixel value of the lower left corner of an aperture.

This corresponds to the OSS corner position (x: ColCorner, y: RowCorner). The notation for OSS is 1-based, i.e. the lower left corner of a FULL subarray is (1,1)

get_polynomial_coefficients()

Return a dictionary of arrays holding the significant idl/sci coefficients.

get_polynomial_derivatives(*location='fiducial'*, *coefficient_seed='Sci2Idl'*)

Return derivative values for scale and rotation derivation.

The four partial derivatives of coefficients that transform between two frames E and R are:

b=(dx_E)/(dx_R) c=(dx_E)/(dy_R) e=(dy_E)/(dx_R) f=(dy_E)/(dy_R)

Parameters

location
[str or dict] if dict, has to have 'x' and 'y' elements

coefficient_seed

Returns

dict

`get_polynomial_linear_parameters`(*location='fiducial'*, *coefficient_seed='Sci2Idl'*)

Return linear polynomial parameters.

Parameters

location
coefficient_seed

Returns

`idl_to_det`(**args*)

Ideal to detector frame transformation.

`idl_to_raw`(**args*)

Ideal to raw frame transformation.

`idl_to_sci`(*x_idl*, *y_idl*)

Ideal to science frame transformation.

`idl_to_sky`(**args*)

`idl_to_tel`(*x_idl*, *y_idl*, *V3IdlYAngle_deg=None*, *V2Ref_arcsec=None*, *V3Ref_arcsec=None*, *method='planar_approximation'*, *input_coordinates='tangent_plane'*, *output_coordinates='tangent_plane'*, *verbose=False*)

Convert from ideal to telescope (V2/V3) coordinate system.

By default, this implements the planar approximation, which is adequate for most purposes but may not be for all. Error is about 1.7 mas at 10 arcminutes from the tangent point. See JWST-STScI-1550 for more details. For higher accuracy, set method='spherical_transformation' in which case 3D matrix rotations are applied. Also by default, the input coordinates are in a tangent plane with a reference points at the origin (0,0) of the ideal frame. input_coordinates can be set to 'spherical'.

Parameters

x_idl: float
ideal X coordinate in arcsec

y_idl: float
ideal Y coordinate in arcsec

V3IdlYAngle_deg: float
overwrites self.V3IdlYAngle

V2Ref_arcsec
[float] overwrites self.V2Ref

V3Ref_arcsec
[float] overwrites self.V3Ref

method
[str] must be one of ['planar_approximation', 'spherical_transformation', 'spherical']

input_coordinates
[str] must be one of ['tangent_plane', 'spherical', 'planar']

Returns

tuple of floats containing V2, V3 coordinates in arcsec

`path`(*to_frame*)

Return matplotlib path generated from aperture vertices.

Parameters

to_frame
[str] Coordinate frame.

Returns

path
[matplotlib.path.Path object] Path describing the aperture outline

`plot`(*frame='tel'*, *label=False*, *ax=None*, *title=False*, *units='arcsec'*, *show_frame_origin=None*, *mark_ref=False*, *fill=True*, *fill_color='cyan'*, *fill_alpha=None*, *label_rotation=0.0*, ***kwargs*)

Plot this aperture.

Parameters

frame
[str] Which coordinate system to plot in: 'tel', 'idl', 'sci', 'det'

label
[str or bool] Add text label. If True, text will be the default aperture name.

ax
[matplotlib.Axes] Desired destination axes to plot into (If None, current axes are inferred)

title
[str] Figure title. If set, add a label to the plot indicating which frame was plotted.

units
[str] one of 'arcsec', 'arcmin', 'deg'

show_frame_origin
[str or list] Plot frame origin (goes to plot_frame_origin()): None, 'all', 'det', 'sci', 'raw', 'idl', or a list of these.

mark_ref
[bool] Add marker for the (V2Ref, V3Ref) reference point defining this aperture.

fill
[bool] Whether to color fill the aperture

fill_color
[str] Fill color

fill_alpha
[float] alpha parameter for filled aperture

color
[matplotlib-compatible color] Color specification for this aperture's outline, passed through to matplotlib.Axes.plot

kwargs
[dict] Dictionary of optional parameters passed to matplotlib

.. todo:: plotting in Sky frame, requires attitude
#elif system == 'radec':
 # if attitude_ref is not None:
 # vertices = np.array(
 # siaf_rotations.pointing(attitude_ref, self.points_closed.T[0],
#self.points_closed.T[1])).T
 # self.path_radec = matplotlib.path.Path(vertices)
 # else:
 # error('Please provide attitude_ref')

`plot_detector_channels`(*frame*, *color='0.5'*, *alpha=0.3*, *evenoddratio=0.5*, *ax=None*)

Outline the detector readout channels.

These are depicted as alternating light/dark bars to show the regions read out by each of the output amps.

Parameters

frame
[str] Which coordinate system to plot in: "tel", "idl", "sci", "det"

color
[matplotlib-compatible color] Color specification for the amplifier shaded region, passed through to matplotlib.patches.Polygon as facecolor

alpha
[float] Opacity of odd-numbered amplifier region overlays (for even, see "evenoddratio")

evenoddratio
[float] Ratio of opacity between even and odd amplifier region overlays

ax
[matplotlib.Axes] Desired destination axes to plot into (If None, current axes are inferred from pyplot.)

`plot_frame_origin`(*frame*, *which='sci'*, *units='arcsec'*, *ax=None*)

Indicate the origins of the detector and science frames.

Red and blue squares indicate the detector and science frame origin, respectively.

Parameters

frame
[str] Which coordinate system to plot in: 'tel', 'idl', 'sci', 'det'

which
[str of list] Which origin to plot: 'all', 'det', 'sci', 'raw', 'idl', or a list

units
[str] one of 'arcsec', 'arcmin', 'deg'

ax
[matplotlib.Axes] Desired destination axes to plot into (If None, current axes are inferred from pyplot.)

`raw_to_det`(**args*)

Raw to detector frame transformation.

`raw_to_idl`(**args*)

Raw to raw ideal transformation.

raw_to_sci(*x_raw*, *y_raw*)

Convert from raw/native coordinates to SIAF-Science coordinates.

SIAF-Science coordinates are the same as DMS coordinates for FULLSCA apertures. Implements the fits_generator description described the table attached to https://jira.stsci.edu/browse/JWSTSIAF-25 (except for Guider 2) and implemented in https://github.com/STScI-JWST/jwst/blob/master/jwst/fits_generator/create_dms_data.py

Parameters

x_raw
[float] Raw x coordinate

y_raw
[float] Raw y coordinate

Returns

x_sci, y_sci
[tuple of science coordinates]

References

See https://jwst-docs.stsci.edu/display/JDAT/Coordinate+Systems+and+Transformations See also JWST-STScI-002566 and JWST-STScI-003222 Rev A

raw_to_tel(**args*)

Raw to telescope frame transformation.

reference_point(*to_frame*)

Return the defining reference point of the aperture in to_frame.

sci_to_det(*x_sci*, *y_sci*, **args*)

Science to detector frame transformation, following Section 4.1 of JWST-STScI-001550.

sci_to_idl(*x_sci*, *y_sci*)

Science to ideal frame transformation.

sci_to_raw(*x_sci*, *y_sci*)

Convert from Science coordinates to raw/native coordinates.

Implements the fits_generator description described the table attached to https://jira.stsci.edu/browse/JWSTSIAF-25 (except for Guider 2) and implemented in https://github.com/STScI-JWST/jwst/blob/master/jwst/fits_generator/create_dms_data.py

Parameters

x_sci
[float] Science x coordinate

y_sci
[float] Science y coordinate

Returns

x_raw, y_raw
[tuple of raw coordinates]

References

See https://jwst-docs.stsci.edu/display/JDAT/Coordinate+Systems+and+Transformations See also JWST-STScI-002566 and JWST-STScI-003222 Rev A

sci_to_sky(**args*)

sci_to_tel(**args*)

Science to telescope frame transformation.

set_attitude_matrix(*attmat*)

Set an atttude matrix, for use in subsequent transforms to sky frame.

set_distortion_coefficients_from_file(*file_name*)

Set polynomial from standardized file.

Parameters

file_name
[str] Reference file containing coefficients

Returns

set_polynomial_coefficients(*sci2idlx*, *sci2idly*, *idl2scix*, *idl2sciy*)

Set the values of polynomial coefficients.

Parameters

sci2idlx
[array] Coefficients

sci2idly
[array] Coefficients

idl2scix
[array] Coefficients

idl2sciy
[array] Coefficients

sky_to_det(**args*)

sky_to_idl(**args*)

sky_to_sci(**args*)

sky_to_tel(**args*)

Sky to Tel frame transformation. Requires an attitude matrix.

Input sky coords should be given in degrees RA and Dec, or equivalent astropy.Quantities

Results are returned as floats with implicit units of arcseconds, for consistency with the other *_to_tel functions.

tel_to_det(**args*)

Telescope to detector frame transformation.

tel_to_idl(*v2_arcsec*, *v3_arcsec*, *V3IdlYAngle_deg=None*, *V2Ref_arcsec=None*, *V3Ref_arcsec=None*, *method='planar_approximation'*, *output_coordinates='tangent_plane'*, *input_coordinates='tangent_plane'*)

Convert from telescope (V2/V3) to ideal coordinate system.

By default, this implements the planar approximation, which is adequate for most purposes but may not be for all. Error is about 1.7 mas at 10 arcminutes from the tangent point. See JWST-STScI-1550 for more details. For higher accuracy, set method='spherical_transformation' in which case 3D matrix rotations are applied.

Also by default, the output coordinates are in a tangent plane with a reference point at the origin (0,0) of the ideal frame.

Parameters

v2_arcsec
[float] V2 coordinate in arcsec

v3_arcsec
[float] V3 coordinate in arcsec

V3IdlYAngle_deg
[float] overwrites self.V3IdlYAngle

V2Ref_arcsec
[float] overwrites self.V2Ref

V3Ref_arcsec
[float] overwrites self.V3Ref

method
[str] must be one of ['planar_approximation', 'spherical_transformation']

output_coordinates
[str] must be one of ['tangent_plane', 'spherical']

Returns

tuple of floats containing x_idl, y_idl coordinates in arcsec

`tel_to_raw`(**args*)

Telescope to raw frame transformation.

`tel_to_sci`(**args*)

Telescope to science frame transformation.

`tel_to_sky`(**args*)

Tel to sky frame transformation. Requires an attitude matrix.

Note, sky coordinates are returned as floats with angles in degrees. It's easy to cast into an astropy SkyCoord and then convert into any desired other representation. For instance:

```
>>> sc = astropy.coordinates.SkyCoord( *aperture.tel_to_sky(xtel, ytel), unit=
↪'deg' )
>>> sc.to_string('hmsdms')
```

`telescope_transform`(*from_system*, *to_system*, *V3IdlYAngle_deg=None*, *V2Ref_arcsec=None*, *V3Ref_arcsec=None*, *verbose=False*)

Return transformation model between tel<->idl.

V3IdlYAngle_deg, V2Ref_arcsec, V3Ref_arcsec can be given as arguments to override aperture attributes.

Parameters

from_system
[str] Starting system ("tel" or "idl")

to_system
[str] Ending coordinate system ("tel" or "idl")

V3IdlYAngle_deg
[float] Angle

V2Ref_arcsec
[float] Reference coordinate

V3Ref_arcsec
[float] Reference coordinate

verbose
[bool] verbosity

Returns

x_model, y_model
[tuple of astropy.modeling models]

`validate()`

Verify that the instance's attributes fully qualify the aperture.

http: // ssb.stsci.edu / doc / jwst / _modules / jwst / datamodels / #wcs_ref_models.html # DistortionModel.validate

`verify()`

Perform internal verification of aperture parameters.

`class pysiaf.aperture.HstAperture`

Class for apertures of HST instruments.

Initialize HST aperture by inheriting from Aperture class.

`closed_polygon_points(to_frame, rederive=False)`

Compute closed polygon points of aperture outline.

`compute_tvs_matrix(v2_arcsec=None, v3_arcsec=None, pa_deg=None, verbose=False)`

Compute the TVS matrix from tvs-specific v2,v3,pa parameters.

Parameters

v2_arcsec
[float] offset

v3_arcsec
[float] offset

pa_deg
[float] angle

verbose
[bool] verbosity

Returns

tvs
[numpy matrix] TVS matrix

`corners`(*to_frame*, *rederive=False*)

Return coordinates of the aperture vertices in the specified frame.

`idl_to_tel`(*x_idl*, *y_idl*, *V3IdlYAngle_deg=None*, *V2Ref_arcsec=None*, *V3Ref_arcsec=None*, *method='planar_approximation'*, *input_coordinates='tangent_plane'*, *output_coordinates=None*, *verbose=False*)

Convert ideal coordinates to telescope (V2/V3) coordinates for HST.

INPUT COORDINATES HAVE TO BE FGS OBJECT SPACE CARTESIAN X,Y coordinates

For HST FGS, transformation is implemented using the FGS TVS matrix. Parameter names are overloaded for simplicity: tvs_pa_deg = V3IdlYAngle_deg tvs_v2_arcsec = V2Ref_arcsec tvs_v3_arcsec = V3Ref_arcsec

Parameters

x_idl
[float] ideal x coordinates in arcsec

y_idl
[float] ideal y coordinates in arcsec

V3IdlYAngle_deg
[float] angle

V2Ref_arcsec
[float] Reference coordinate

V3Ref_arcsec
[float] Reference coordinate

method
[str]

input_coordinates
[str]

Returns

v2, v3
[tuple of float] Telescope coordinates in arcsec

`rearrange_fgs_alignment_parameters`(*pa_deg_in*, *v2_arcsec_in*, *v3_arcsec_in*, *direction*)

Convert to/from alignment parameters make FGS look like a regular camera aperture.

Parameters

pa_deg_in
[float]

v2_arcsec_in
[float]

v3_arcsec_in
[float]

direction
[str] One of [‘fgs_to_camera’,]

Returns

pa, v2, v3
[tuple] re-arranged and sometimes sign-inverted alignment parameters

`set_idl_reference_point`(*v2_ref*, *v3_ref*, *verbose=False*)

Determine the reference point in the Ideal frame.

V2Ref and V3Ref are determined via the TVS matrix. The tvs parameters that determine the TVS matrix itself are derived and added to the attribute list.

Parameters

v2_ref
[float] reference coordinate

v3_ref
[float] reference coordinate

verbose
[bool] verbosity

`set_tel_reference_point`(*verbose=True*)

Recompute and set V2Ref and V3Ref to actual position in tel/V frame.

This is after using those V2Ref and V3Ref attributes for TVS matrix update.

`tel_to_idl`(*v2_arcsec*, *v3_arcsec*, *V3IdlYAngle_deg=None*, *V2Ref_arcsec=None*, *V3Ref_arcsec=None*, *method='planar_approximation'*, *output_coordinates='tangent_plane'*, *input_coordinates='tangent_plane'*)

Convert from telescope (V2/V3) to ideal coordinate system.

By default, this implements the planar approximation, which is adequate for most purposes but may not be for all. Error is about 1.7 mas at 10 arcminutes from the tangent point. See JWST-STScI-1550 for more details. For higher accuracy, set method='spherical_transformation' in which case 3D matrix rotations are applied.

Also by default, the output coordinates are in a tangent plane with a reference point at the origin (0,0) of the ideal frame.

Parameters

v2_arcsec
[float] V2 coordinate in arcsec

v3_arcsec
[float] V3 coordinate in arcsec

V3IdlYAngle_deg
[float] overwrites self.V3IdlYAngle

V2Ref_arcsec
[float] overwrites self.V2Ref

V3Ref_arcsec
[float] overwrites self.V3Ref

method
[str] must be one of ['planar_approximation', 'spherical_transformation']

output_coordinates
[str] must be one of ['tangent_plane', 'spherical']

Returns

tuple of floats containing x_idl, y_idl coordinates in arcsec

class pysiaf.aperture.**JwstAperture**

Class for apertures of JWST instruments.

Initialize JWST aperture by inheriting from Aperture.

class pysiaf.aperture.**NirspecAperture**(*tilt=None*, *filter_name='CLEAR'*)

Class for apertures of the JWST NIRSpec instrument.

Initialize NIRSpec aperture by inheriting from JWSTAperture.

corners(*to_frame*, *rederive=True*)

Return coordinates of aperture vertices.

det_to_sci(*x_det*, *y_det*, **args*)

Detector to science frame transformation. Use parent aperture if SLIT.

gwa_to_ote(*gwa_x*, *gwa_y*)

NIRSpec transformation from GWA sky side to OTE frame XAN, YAN.

Parameters

gwa_x
[float] GWA coordinate

gwa_y
[float] GWA coordinate

Returns

xan, yan
[tuple of floats] XAN, YAN in degrees

gwa_to_sci(*x_gwa*, *y_gwa*)

NIRSpec transformation from GWA detector side to Science frame.

Parameters

x_gwa
[float] GWA coordinate

y_sci
[float] GWA coordinate

Returns

x_sci, y_sci
[tuple of floats] Science coordinates

gwain_to_gwaout(*x_gwa*, *y_gwa*)

Transform from GWA detector side to GWA skyward side. This is the effect of the mirror.

Parameters

x_gwa
[float] GWA coordinate

y_gwa
[float] GWA coordinate

Returns

x_gwap, y_gwap
[tuple of floats] GWA skyward side coordinates

References

Giardino, Ferruit, and Alves de Oliveira (2014), ESA NTN-2014-005 Proffitt et al. (2018), JWST-STScI-005921

gwaout_to_gwain(*x_gwa*, *y_gwa*)

Transform from GWA skyward side to GWA detector side. Effect of mirror.

Parameters

x_gwa
[float] GWA coordinate

y_gwa
[float] GWA coordinate

Returns

x_gwap, y_gwap
[tuple of floats] GWA detector side coordinates

References

Equations for the reverse transform T. Keyes (private communication) will be documented in next update of JWST-STScI-005921.

idl_to_sci(*x_idl*, *y_idl*)

Transform to Science frame using special implementation for NIRSpec via tel frame.

Parameters

x_idl
[float] Ideal coordinate

y_idl
[float] Ideal coordinate

Returns

x_sci, y_sci
[tuple of floats] Science coordinates

ote_to_gwa(*ote_x*, *ote_y*)

NIRSpec transformation from OTE frame XAN, YAN to GWA sky side.

Parameters

ote_x
[float] XAN coordinate in degrees

ote_y
[float] YAN coordinate in degrees

Returns

gwa_x, gwa_y
[tuple of floats] GWA coordinates

sci_to_det(*x_sci*, *y_sci*, **args*)

Science to detector frame transformation. Use parent aperture if SLIT.

sci_to_gwa(*x_sci*, *y_sci*)

NIRSpec transformation from Science frame to GWA detector side.

Parameters

x_sci
[float] Science frame coordinate

y_sci
[float] Science frame coordinate

Returns

x_gwa, y_gwa
[tuple of floats] GWA detector side coordinates

sci_to_idl(*x_sci*, *y_sci*)

Transform to ideal frame, using special implementation for NIRSpec via tel frame.

Parameters

x_sci
[float] Science coordinate

y_sci
[float] Science coordinate

Returns

v2, v3
[tuple of floats] telescope coordinates

sci_to_tel(*x_sci*, *y_sci*)

Science to telescope frame transformation. Overwrite standard behaviour for NIRSpec.

tel_to_sci(*x_tel*, *y_tel*)

Telescope to science frame transformation for NIRSpec.

class pysiaf.aperture.**RomanAperture**

A class that defines the accepted aperture types for the Roman SIAF. This is built upon the pysiaf.aperture.Aperture base class.

Set attributes required for PRD.

class pysiaf.aperture.**SiafCoordinates**(*x*, *y*, *frame*)

A helper class to hold coordinate arrays associated with a SIAF frame.

Initialize object.

pysiaf.aperture.**compare_apertures**(*reference_aperture*, *comparison_aperture*, *absolute_tolerance=None*, *attribute_list=None*, *print_file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>*, *fractional_tolerance=1e-06*, *verbose=False*, *ignore_attributes=None*)

Compare the attributes of two apertures.

Parameters

reference_aperture
comparison_aperture
absolute_tolerance
attribute_list
print_file
fractional_tolerance
verbose
ignore_attributes

pysiaf.aperture.**get_hst_to_jwst_coefficient_order**(*polynomial_degree*)

Return array of indices that convert an aeeay of HST coefficients to JWST ordering.

This assumes that the coefficient orders are as follows HST: 1, y, x, y^2, xy, x^2, y^3, xy^2, x^2y, x^3 … (according to Cox's /grp/hst/OTA/alignment/FocalPlane.xlsx) JWST: 1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3 …

Parameters

polynomial_degree
[int] polynomial degree

Returns

conversion_index
[numpy array] Array of indices

pysiaf.aperture.**linear_transform_model**(*from_system*, *to_system*, *parity*, *angle_deg*)

Create an astropy.modeling.Model object for linear transforms: det <-> sci and idl <-> tel.

See sics_to_v2v3 (HST): x_v2v3 = v2_origin + parity * x_sics * np.cos(np.deg2rad(theta_deg)) + y_sics * np.sin(np.deg2rad(theta_deg)) y_v2v3 = v3_origin - parity * x_sics * np.sin(np.deg2rad(theta_deg)) + y_sics * np.cos(np.deg2rad(theta_deg))

Parameters

from_system
[str] Starting system

to_system
[str] End system

parity
[int] Parity

angle_deg
[float] Angle in degrees

Returns

xmodel, ymodel
[tuple of astropy models] Transformation models

pysiaf.aperture.**points_on_arc**(*x0*, *y0*, *radius*, *phi1_deg*, *phi2_deg*, *N=100*)

Compute points that lie on a circular arc between angles phi1 and phi2.

Parameters

x0
[float] offset

y0
[float] offset

radius
[float] Radius of circle

phi1_deg
[float] Start angle of pie in deg

phi2_deg
[float] End angle of pie in deg

N
[int] Number of sampled points

Returns

x, y
[tuple of floats] Coordinates on circle

pysiaf.aperture.**to_distortion_model**(*coefficients*, *degree=5*)

Create an astropy.modeling.Model object for distortion polynomial.

Parameters

coefficients
[array like] Coefficients from the ISIM transformations file.

degree
[int] Degree of polynomial. Default is 5 as in the ISIM file but many of the polynomials are of a smaller degree.

Returns

poly
[astropy.modeling.Polynomial2D] Polynomial model transforming one coordinate (x or y) between two systems.

1.7.2 compare

Functions to support comparisons between SIAF files.

Authors

Johannes Sahlmann

References

The format of the difference files was adapted from Colin Cox' matchcsv.py dict_compare was adapted from https://stackoverflow.com/questions/4527942/comparing-two-dictionaries-in-python/4527957

pysiaf.utils.compare.**compare_inspection_figures**(*comparison_siaf_input*, *reference_siaf_input=None*, *report_dir=None*, *selected_aperture_name=None*, *skipped_aperture_type=None*, *tags=None*, *mark_ref=False*, *xlimits=None*, *ylimits=None*, *filename_appendix=''*, *label=False*)

Visualize aperture of two SIAF files.

Parameters

comparison_siaf_input
[str (absolute file name) or pysiaf.Siaf object] The SIAF that will be compared to the reference_siaf

reference_siaf_input
[str (absolute file name) or pysiaf.Siaf object] The reference SIAF. Defaults to the current PRD content.

report_dir
selected_aperture_name str or list
aperture name(s) to include in plot

skipped_aperture_type
[str or list] aperture type(s) not to include in plot

tags
xlimits
[tuple of limits of output plots]

ylimits
[tuple of limits of output plots]

Returns

pysiaf.utils.compare.**compare_siaf**(*comparison_siaf_input*, *fractional_tolerance=0.0001*, *reference_siaf_input=None*, *report_file=None*, *report_dir=None*, *verbose=True*, *make_figures=False*, *selected_aperture_name=None*, *ignore_attributes=None*, *tags=None*)

Compare two SIAF files and write a difference file. Generate comparison figures showing the apertures if specified.

Parameters

comparison_siaf_input
[str (absolute file name) or pysiaf.Siaf object] The SIAF that will be compared to the reference_siaf

fractional_tolerance
[float] Value above which a fractional difference in parameter value will be written in the report

reference_siaf_input
[str (absolute file name) or pysiaf.Siaf object] The reference SIAF. Defaults to the current PRD content.

report_file
report_dir
verbose
make_figures
selected_aperture_name

pysiaf.utils.compare.**compare_transformation_roundtrip**(*comparison_siaf_input*, *fractional_tolerance=0.0001*, *reference_siaf_input=None*, *report_file=None*, *report_dir=None*, *verbose=True*, *make_figures=False*, *selected_aperture_name=None*, *skipped_aperture_type=None*, *instrument=None*, *make_plot=False*, *tags=None*)

Compare the forward-backward roundtrip transformations of two SIAF files. and write a difference file.

Parameters

comparison_siaf_input
[str (absolute file name) or pysiaf.Siaf object] The SIAF that will be compared to the reference_siaf

fractional_tolerance
[float] Value above which a fractional difference in parameter value will be written in the report

reference_siaf_input
[str (absolute file name) or pysiaf.Siaf object] The reference SIAF. Defaults to the current PRD content.

report_file
report_dir
verbose
make_figures
selected_aperture_name
[str or list] aperture name(s) to include in plot

skipped_aperture_type
[str or list] aperture type(s) not to include in plot

instrument
make_plot

Returns

roundtrip_table
[astropy.table.Table object] table containing roundtrip data

pysiaf.utils.compare.**dict_compare**(*dictionary_1*, *dictionary_2*)

Compare two dictionaries and return keys of the differing items. From https://stackoverflow.com/questions/4527942/comparing-two-dictionaries-in-python/4527957

Parameters

dictionary_1
[dict] first dictionary

dictionary_2
[dict] second dictionary

Returns

added, removed, modified, same
[set] Sets of dictionary keys that were added, removed, modified, or are the same

pysiaf.utils.compare.**show_save_plot**(*report_dir*)

1.7.3 polynomial

A collection of functions to manipulate polynomials and their coefficients.

Authors

- Colin Cox
- Johannes Sahlmann

pysiaf.utils.polynomial.**add_rotation**(*A*, *B*, *theta_deg*)

Add rotation after polynomial transformation.

Use when a distortion transformation using polynomials A and B is followed by a rotation.

u = A(x,y) v = B(x,y) followed by u2 = u*cos(theta) + v*sin(theta) v2 = -u*sin(theta) + v*cos(theta) This routine supplies a modified pair of polynomial which combine both steps i.e u2 = A2(x,y), v2 = B2(x,y)

Parameters

A
[array] Set of polynomial coefficients converting from (x,y) to a variable u

B
[array] set of polynomial coefficients converting from(x,y) to avariable v

theta_deg
[float] The angle in degrees of a rotationin the (u,v) plane

Returns

A2
[array] set of polynomial coefficiients providing combined steps from (x,y) to u2

B2
[array] set of polynomial coefficients providing combined steps from (x,y) to v2

Notes

Function formerly named Rotate or rotate_coefficients. Ported from makeSIAF.py by J. Sahlmann 2018-01-03.

pysiaf.utils.polynomial.**choose**(*n*, *r*)

Return number of ways of choosing r items from an array with n items.

Parameters

n
[int] number of items to choose from

r
[int] number of items to choose

Returns

combinations
[int] The number if ways of making the choice

pysiaf.utils.polynomial.**dpdx**(*a*, *x*, *y*)

Differential with respect to x.

The polynomial is defined as p(x,y) = a[i,j] * x**(i-j) * y**j summed over i and j. Then dp/dx = (i-j) * a[i,j] * x**(i-j-1) * y**j.

Parameters

a
[array] a linear array of polynomial coefficients in JWST order.

x
[array] an integer or float variable(or an array of same) representing pixel x positions

y
[array] a variable (or an array) representing pixel y positions. x and y must be of same shape.

Returns

differential
[array] float values of dp/dx for the given (x,y) point(s)

pysiaf.utils.polynomial.**dpdy**(*a*, *x*, *y*)

Differential with respect to y.

The polynomial is defined as p(x,y) = a[i,j] * x**(i-j) * y**j, summed over i and j Then dp/dy = j * a[i,j] * x**(i-j) * y**(j-1)

Parameters

a
[array] an array of polynomial coefficients in JWST arrangement. The number of coefficients must be (order+1)(order+2)/2

x
[array] an integer or float variable(or an array of same) representing pixel x positions

y
[array] a variable (or an array) representing pixel y positions

Returns

differential
[array] float value of dp/dy for the given (x,y) point(s) where p(x,y) is the value of the polynomial

pysiaf.utils.polynomial.**flatten**(*coefficients*)

Convert triangular layout to linear array.

For many of the polynomial operations the coefficients A[i,j] are contained in an array of dimension (order+1, order+1) but with all elements where j > i set equal to zero. This is what is called the triangular layout.

The flattened layout is a one-dimensional array containing only the elements where j <= i.

Parameters

coefficients
[array] a two-dimensional float array of shape (order+1, order+1) supplying the polynomial coefficients in JWST order. The coefficient at position [i,j] multiplies the value of x**(i-j) * y**j.

Returns

flate_coefficients
[array] a one-dimensional array including only those terms where i <= j

pysiaf.utils.polynomial.**flip_x**(*A*)

Change sign of all coefficients with odd x power.

Used when we have a polynomial expansion in terms of variables x and y and we wish to obtain one in which the sign of x is reversed

Parameters

A
[array] A set of polynomial coefficients given in the triangular layout as described in poly

Returns

AF
[array] Modified or flipped set of coefficients matching negated x values.

pysiaf.utils.polynomial.**flip_xy**(*A*)

Change sign for coeffs where sum of x and y powers is odd.

Used when we have a polynomial expansion in terms of variables x and y and we wish to obtain one in which the signs of x and y are reversed

Parameters

A
[array] A set of polynomial coefficients given in the triangular layout as described in the function poly

Returns

AF
[array] Modified or flipped set of coefficients matching negated x and y values.

pysiaf.utils.polynomial.**flip_y**(*A*)

Change sign of all coefficients with odd y power.

Used when we have a polynomial expansion in terms of variables x and y and we wish to obtain one in which the sign of y is reversed

Parameters

A
[array] A set of polynomial coefficients given in the triangular layout as described in the function poly

Returns

AF
[array] Modified or flipped set of coefficients matching negated y values.

pysiaf.utils.polynomial.**invert**(*A*, *B*, *u*, *v*, *verbose=False*)

Newton Raphson method in two dimensions.

Given that u = A[i,j] * x**(i-j) * y**j and v = B[i,j] * x**(i-j) * y**j find the values of x and y from the values of u and v

Parameters

A
[array] A set of polynomial coefficients given in the linear layout as described in the function poly converting (x,y) to u

B
[array] A set of polynomial coefficients given in the linear layout as described in the function poly converting (x,y) to v

u
[array] The result of applying the A coefficients to the (x,y) position

v
[array] The result of applying the B coefficients to the (x, y)position

verbose
[bool] Logical variable, set True if full text output required

Returns

x, y
[tuple of arrays] The pair of values which transform to (u,v)

err
[float] the standard deviation of the fit

iteration
[int] the number of iterations taken to determine the solution

pysiaf.utils.polynomial.**jacob**(*a*, *b*, *x*, *y*)

Calculate relative area using the Jacobian.

` ``` | da_dx db_dx | ``Jacobian = | | ``` | da_dy db_dy | ``` `

Then the relative area is the absolute value of the determinant of the Jacobian. x and y will usually be Science coordinates while u and v are Ideal coordinates

Parameters

a
[array] set of polynomial coefficients converting from (x,y) to u

b
[array] set of polynomial coefficients converting from (x,y) to v

x
[array] x pixel position or array of x positions

y
[array] y pixel position or array of y positions matching the y positions

Returns

area
[array] area in (u,v) coordinates matching unit area in the (x,y) coordinates.

pysiaf.utils.polynomial.**number_of_coefficients**(*poly_degree*)

Return number of coefficients corresponding to polynomial degree.

pysiaf.utils.polynomial.**poly**(*a*, *x*, *y*, *order=4*)

Polynomial evaluation.

pol = a[i,j] * x**(i-j) * y**j summed over i and j, where i runs from 0 to order. Then for each value of i, j runs from 0 to i. For many of the polynomial operations the coefficients A[i,j] are contained in an array of dimension (order+1, order+1) but with all elements where j > i set equal to zero. This is called the triangular layout. The flattened layout is a one-dimensional array containing copies of only the elements where j <= i.

The JWST layout is a[0,0] a[1,0] a[1,1] a[2,0] a[2,1] a[2,2] … The number of coefficients will be (n+1)(n+2)/2

Parameters

a
[array] float array of polynomial coefficients in flattened arrangement

x
[array] x pixel position. Can be integer or float or an array of integers or floats

y
[array] y pixel position in same layout as x positions.

order
[int] integer polynomial order

Returns

pol
[float] result as described above

pysiaf.utils.polynomial.**polyfit**(*u*, *x*, *y*, *order*)

Fit polynomial to a set of u values on an x,y grid.

u is a function u(x,y) being a polynomial of the form u = a[i, j] x**(i-j) y**j. x and y can be on a grid or be arbitrary values This version uses scipy.linalg.solve instead of matrix inversion. u, x and y must have the same shape and may be 2D grids of values.

Parameters

u
[array] an array of values to be the results of applying the sought after polynomial to the values (x,y)

x
[array] an array of x values

y
[array] an array of y values

order
[int] the polynomial order

Returns

coeffs: array
polynomial coefficients being the solution to the fit.

pysiaf.utils.polynomial.**polynomial_degree**(*number_of_coefficients*)

Return degree of the polynomial that has number_of_coefficients.

Parameters

number_of_coefficients
[int] Number of polynomial coefficients

Returns

polynomial_degree
[int] Degree of the polynomial

pysiaf.utils.polynomial.**prepend_rotation_to_polynomial**(*a*, *theta*, *verbose=False*)

Rotate axes of coefficients by theta degrees.

Used when a distortion transformation using polynomials A and B is preceded by a rotation. The set of polynomial coefficients a[i,j] transform (x,y) as u = a[i,j] * x**(i-j) * y**j Summation over repeated indices is implied.

If now we have a set of variables (xp,yp) rotated from (x,y) so that xp = x * cos(theta) - y * sin(theta) yp = x * sin(theta) + y * cos(theta) find a set of polynomial coefficients ap so that the same value of u is obtained from (xp,yp) i.e, u = ap[i,j] * xp**(i-j) * yp**j The rotation is opposite to the usual rotation as this routine was designed for the inverse transformation between Ideal and V2V3 or tel. Effectively the angle is reversed

Parameters

a
[array] Set of polynomial coefficients

theta
[float] rotation angle in degrees

verbose
[bool] logical variable set True only if print-out of coefficient factors is desired.

Returns

arotate
[array] set of coefficients derived as described above.

Notes

Function was formerly named RotateCoeffs.

pysiaf.utils.polynomial.**print_triangle**(*coefficients*)

Print coefficients in triangular layout.

A[0] A[1] A[2] A[3] A[4] A[5] … equivalent to A[0,0] A[1,0] A[1,1] A[2,0] A[2,1] A[2,2] … in [i,j] terms.

See method poly for details. This is just to display the coefficients. No calculation performed.

Parameters

coefficients
[array] polynomial float array in linear layout

pysiaf.utils.polynomial.**reorder**(*A*, *B*)

Change coefficient order from y**2 xy x**2 to x**2 xy y**2 in both A and B.

Parameters

A
[array] polynomial coefficients

B
[array] polynomial coefficients

Returns

A2, B2: numpy arrays
coefficients with changed order

pysiaf.utils.polynomial.**rescale**(*A*, *B*, *C*, *D*, *scale*)

Apply a scale to forward and inverse polynomial coefficients.

Parameters

A
[array] Polynomial coefficients

B
[array] Polynomial coefficients

C
[array] Polynomial coefficients

D
[array] Polynomial coefficients

scale
[float] Scale factor to apply

Returns

A_scaled, B_scaled, C_scaled, D_scaled
[tuple of numpy arrays] the scales coefficients

Notes

Ported from makeSIAF.py by J. Sahlmann 2018-01-03. J. Sahlmann 2018-01-04: fixed side-effect on ABCD variables

pysiaf.utils.polynomial.**rotation_from_derivatives**(*pc1*, *pc2*)

Return rotation estimate.

pysiaf.utils.polynomial.**rotation_scale_skew_from_derivatives**(*b*, *c*, *e*, *f*)

Compute rotations, scales, and skews from polynomial derivatives.

The four partial derivatives of coefficients that transform between two frames E and R are:

Parameters

b
[float] (dx_E)/(dx_R)

c
[float] (dx_E)/(dy_R)

e
[float] (dy_E)/(dx_R)

f
[float] (dy_E)/(dy_R)

Returns

pysiaf.utils.polynomial.**scale_from_derivatives**(*pc1*, *pc2*)

Return scale estimate.

pysiaf.utils.polynomial.**shift_coefficients**(*a*, *xshift*, *yshift*, *verbose=False*)

Calculate coefficients of polynomial when shifted to new origin.

Given a polynomial function such that u = a[i,j] * x**[i-j] * y**[j] summed over i and j. Find the polynomial function ashift centered at xshift, yshift i.e the same value of u = ashift[i,j] * (x-xshift)**(i-j) * (y-yshift)**j.

Parameters

a
[array] Set of coefficients for a polynomial of the given order in JWST order

xshift
[float] x position in pixels of new solution center

yshift
[float] y position in pixels of new solution center

verbose
[bool] logical variable to choose print-out of coefficient table - defaults to False

Returns

ashift
[array] shifted version of the polynomial coefficients.

pysiaf.utils.polynomial.**transform_coefficients**(*A*, *a*, *b*, *c*, *d*, *verbose=False*)

Transform polynomial coefficients.

This allows for xp = a*x + b*y yp = c*x + d*y

Parameters

A
[array] Polynomial coefficients

a
[float] factor

b
[float] factor

c
[float] factor

d
[float] factor

verbose
[bool] verbosity

Returns

AT
[array] Transformed coefficients

Notes

Designed to work with Sabatke solutions which included a linear transformation of the pixel coordinates before the polynomial distortion solution was calculated. *transform_coefficients* combines the two steps into a single polynomial.

pysiaf.utils.polynomial.**triangular_layout**(*coefficients*)

Convert linear array to 2-D array with triangular coefficient layout.

This is the reverse of the flatten method.

Parameters

coefficients
[array] float array of polynomial coefficients. Must be of dimension (order+1)(order+2)/2.

Returns

triangular_coefficients
[array] coefficients in triangular layout as described in poly.

pysiaf.utils.polynomial.**two_step**(*A*, *B*, *a*, *b*)

Combine linear step followed by a polynomial step into a single polynomial.

Designed to process Sabatke polynomials which had a linear transformation ahead of the polynomial fits.

Starting from a pair of polynomial arrays A and B such that u = A[i,j[* xp**(i-j) * yp**j v = B[i,j] * xp**(i-j) * yp**j in which xp = a[0] + a[1].x + a[2].y yp = b[0] + b[1].x + b[2].y find AP and BP such that the same u and v values are given by u = AP[i,j] * x**(i-j) * y**j v = BP[i,j] * x**(i-j) * y**j All input and output polynomials are flattened arrays of dimension (order+1)(order+2)/2 Internally they are processed as equivalent two dimensional arrays as described in the poly documentation.

Parameters

A
[array] polynomial converting from secondary xp and yp pixel positions to final coordinates u

B
[array] polynomial converting from secondary xp and yp pixel positions to final coordinates v

Aflat
[array] set of linear coefficients converting (x,y) to xp

Bflat
[array] set of linear coefficients converting (x,y) to yp

Returns

Aflat, Bflat
[tuple of arrays] polynomial coefficients as calculated

1.7.4 projection

A collection of functions to support tangent-plane de-/projections.

Authors

Johannes Sahlmann

pysiaf.utils.projection.**deproject_from_tangent_plane**(*x*, *y*, *ra_ref*, *dec_ref*, *scale=1.0*, *unwrap=True*)

Convert pixel coordinates into ra/dec coordinates using a tangent plane de-projection.

The projection's reference point has to be specified. See the inverse transformation radec2Pix_TAN.

Parameters

x
[float or array of floats] Pixel coordinate (default is in decimal degrees, but depends on value of scale parameter) x/scale has to be degrees.

y
[float or array of floats] Pixel coordinate (default is in decimal degrees, but depends on value of scale parameter) x/scale has to be degrees.

ra_ref
[float] Right Ascension of reference point in decimal degrees

dec_ref: float
declination of reference point in decimal degrees

scale
[float] Multiplicative factor that is applied to the input values. Default is 1.0

Returns

ra
[float] Right Ascension in decimal degrees

dec: float
declination in decimal degrees

pysiaf.utils.projection.**project_to_tangent_plane**(*ra*, *dec*, *ra_ref*, *dec_ref*, *scale=1.0*)

Convert ra/dec coordinates into pixel coordinates using a tangent plane projection.

Theprojection's reference point has to be specified. Scale is a convenience parameter that defaults to 1.0, in which case the returned pixel coordinates are also in degree. Scale can be set to a pixel scale to return detector coordinates in pixels

Parameters

ra
[float] Right Ascension in decimal degrees

dec: float
declination in decimal degrees

ra_ref
[float] Right Ascension of reference point in decimal degrees

dec_ref: float
declination of reference point in decimal degrees

scale
[float] Multiplicative factor that is applied to the returned values. Default is 1.0

Returns

x,y
[float] pixel coordinates in decimal degrees if scale = 1.0

1.7.5 read

Functions to read Science Instrument Aperture Files (SIAF) and SIAF reference files.

For JWST SIAF, reading XML and CSV format are supported. For HST SIAF, only .dat files can be read.

Authors

Johannes Sahlmann

References

Parts of read_hst_siaf were adapted from Matt Lallo's plotap.f. Parts of read_jwst_siaf were adapted from jwxml.

pysiaf.iando.read.**get_jwst_siaf_instrument**(*tree*)

Return the instrument specified in the first aperture of a SIAF xml tree.

Returns

instrument
[str] All Caps instrument name, e.g. NIRSPEC

pysiaf.iando.read.**get_siaf**(*input_siaf*, *observatory='JWST'*)

Return a Siaf object corresponding to input_siaf which can be a string path or a Siaf object.

Parameters

input_siaf
observatory

Returns

siaf_object: pysiaf.Siaf
Siaf object

pysiaf.iando.read.**month_name_to_number**(*month*)

Convert month name to digit.

pysiaf.iando.read.**read_hst_fgs_amudotrep**(*file=None*, *version=None*)

Read HST FGS amu.rep file which contain the TVS matrices.

Parameters

filepath
[str] Path to file.

Returns

data
[dict] Dictionary that holds the file content ordered by FGS number

pysiaf.iando.read.**read_hst_siaf**(*file=None*, *version=None*)

Read apertures from HST SIAF file and return a collection.

This was partially ported from Lallo's plotap.f.

Parameters

file
[str]

AperNames
[str list]

Returns

apertures: dict
Dictionary of apertures

pysiaf.iando.read.**read_jwst_siaf**(*instrument=None*, *filename=None*, *basepath=None*)

Read the JWST SIAF and return a collection of apertures.

Parameters

instrument
[str] instrument name (case-insensitive)

filename
[str] Absolute path to alternative SIAF xml file

basepath
[str] Directory containing alternative SIAF xml file conforming with standard naming convention

Returns

apertures
[dict] dictionary of apertures

pysiaf.iando.read.**read_roman_siaf**(*siaf_file=None*)

Returns

apertures (collections.OrderedDict)
An ordered dictionary of pysiaf.aperture objects containing each aperture from the Roman SIAF file that was read.

pysiaf.iando.read.**read_siaf_alignment_parameters**(*instrument*)

Return astropy table.

Parameters

instrument

Returns

: astropy table

pysiaf.iando.read.**read_siaf_aperture_definitions**(*instrument*, *directory=None*)

Return astropy table.

Parameters

instrument
[str] instrument name (case insensitive)

Returns

: astropy table
content of SIAF reference file

pysiaf.iando.read.**read_siaf_ddc_mapping_reference_file**(*instrument*)

Return dictionary with the DDC mapping.

Parameters

instrument
[str] instrument name (case insensitive)

Returns

: astropy table

pysiaf.iando.read.**read_siaf_detector_layout**()

Return the SIAF detector layout read from the SIAF reference file.

Returns

: astropy table

pysiaf.iando.read.**read_siaf_detector_reference_file**(*instrument*)

Return astropy table.

Parameters

instrument
[str] instrument name (case insensitive)

Returns

: astropy table

pysiaf.iando.read.**read_siaf_distortion_coefficients**(*instrument=None*, *aperture_name=None*, *file_name=None*)

Return astropy table.

Parameters

instrument
[str] instrument name (case insensitive)

aperture_name
[str] name of master aperture

file_name
[str] file name to read from, ignoring the first two arguments

Returns

: astropy table

pysiaf.iando.read.**read_siaf_xml_field_format_reference_file**(*instrument=None*)

Return astropy table.

Parameters

instrument
[str] instrument name (case insensitive)

Returns

: astropy table

1.7.6 rotations

A collection of basic routines for performing rotation calculations.

Authors

Colin Cox Johannes Sahlmann

pysiaf.utils.rotations.**attitude**(*v2*, *v3*, *ra*, *dec*, *pa*)

Return rotation matrix that transforms from v2,v3 to RA,Dec.

Makes a 3D rotation matrix which rotates a unit vector representing a v2,v3 position to a unit vector representing an RA, Dec pointing with an assigned position angle Described in JWST-STScI-001550, SM-12, section 5.1.

Parameters

v2
[float] a position measured in arc-seconds

v3
[float] a position measured in arc-seconds

ra
[float] Right Ascension on the sky in degrees

dec
[float] Declination on the sky in degrees

pa
[float] Position angle in degrees measured from North to V3 axis in North to East direction.

Returns

m
[numpy matrix] A (3 x 3) matrix represents the attitude of the telescope which points the given V2V3 position to the indicated RA and Dec and with the V3 axis rotated by position angle pa

pysiaf.utils.rotations.**attitude_matrix**(*nu2*, *nu3*, *ra*, *dec*, *pa*, *convention='JWST'*)

Return attitude matrix.

Makes a 3D rotation matrix that transforms between telescope frame and sky. It rotates a unit vector on the idealized focal sphere (specified by the spherical coordinates nu2, nu3) to a unit vector representing an RA, Dec pointing with an assigned position angle measured at nu2, nu3. See JWST-STScI-001550, SM-12, section 5.1.

Parameters

nu2
[float] an euler angle (default unit is arc-seconds)

nu3
[float] an euler angle (default unit is arc-seconds)

ra
[float] Right Ascension on the sky in degrees

dec
[float] Declination on the sky in degrees

pa
[float] Position angle of V3 axis at nu2,nu3 measured from North to East (default unit is degree)

Returns

m
[numpy matrix] the attitude matrix

pysiaf.utils.rotations.**axial_rotation**(*ax*, *phi*, *vector*)

Apply direct rotation to a vector using Rodrigues' formula.

Parameters

ax
[float array of size 3] a unit vector represent a rotation axis

phi
[float] angle in degrees to rotate original vector

vector
[float] array of size 3 representing any vector

Returns

v
[float] array of size 3 representing the rotated vectot

pysiaf.utils.rotations.**convert_quantity**(*x_in*, *to_unit*, *factor=1.0*)

Check if astropy quantity and apply conversion factor

Parameters

x_in
[float or quantity] input

to_unit
[astropy.units unit] unit to convert to

factor
[float] Factor to apply if input is not a quantity

Returns

x_out
[float] converted value

pysiaf.utils.rotations.**cross**(*a*, *b*)

Return cross product of two vectors c = a X b.

The order is significant. Reversing the order changes the sign of the result.

Parameters

a
[float array or list of length 3] first vector

b
[float array or list of length 3] second vector

Returns

c float array of length 3
the product vector

pysiaf.utils.rotations.**getv2v3**(*attitude*, *ra*, *dec*)

Return v2,v3 position of any RA and Dec using the inverse of attitude matrix.

Parameters

attitude
[3 by 3 float array] the telescope attitude matrix

ra
[float] RA of sky position

dec
[float] Dec of sky position

Returns

v2,v3
[tuple of floats] V2,V3 value at matching position

pysiaf.utils.rotations.**idl_to_tel_rotation_matrix**(*V2Ref_arcsec*, *V3Ref_arcsec*, *V3IdlYAngle_deg*)

Return 3D rotation matrix for ideal to telescope transformation.

Parameters

V2Ref_arcsec
V3Ref_arcsec
V3IdlYAngle_deg

Returns

pysiaf.utils.rotations.**pointing**(*attitude*, *v2*, *v3*, *positive_ra=True*, *input_cartesian=False*)

Calculate where a v2v3 position points on the sky using the attitude matrix.

Parameters

attitude
[3 by 3 float array] the telescope attitude matrix

v2
[float or array of floats] V2 coordinate in arc-seconds

v3
[float or array of floats] V3 coordinate in arc-seconds

positive_ra
[bool.] If True forces ra value to be positive

Returns

rd
[tuple of floats] (ra, dec) - RA and Dec in degrees

pysiaf.utils.rotations.**polar_angles**(*vector*, *positive_azimuth=False*)

Compute polar coordinates of an unit vector.

Parameters

vector
[float list or array of length 3] 3-component unit vector

positive_azimuth
[bool] If True, the returned nu2 value is forced to be positive.

Returns

nu2, nu3
[tuple of floats with astropy quantity] The same position represented by polar coordinates

pysiaf.utils.rotations.**posangle**(*attitude*, *v2*, *v3*)

Return the V3 angle at arbitrary v2,v3 using the attitude matrix.

This is the angle measured from North to V3 in an anti-clockwise direction i.e. North to East. Formulae from JWST-STScI-001550, SM-12, section 6.2. Subtract 1 from each index in the text to allow for python zero indexing.

Parameters

attitude
[3 by 3 float array] the telescope attitude matrix

v2
[float] V2 coordinate in arc-seconds

v3
[float] V3 coordinate in arc-seconds

Returns

pa
[float] Angle in degrees - the position angle at (V2,V3)

pysiaf.utils.rotations.**radec**(*vector*, *positive_ra=False*)

Return RA and Dec in degrees corresponding to the unit vector vector.

Parameters

vector
[a float array or list of length 3] represents a unit vector so should have unit magnitude if not, the normalization is forced within the method

positive_ra
[bool] indicating whether to force ra to be positive

Returns

ra , dec
[tuple of floats] RA and Dec in degrees corresponding to the unit vector vector

pysiaf.utils.rotations.**rodrigues**(*attitude*)

Interpret rotation matrix as a single rotation by angle phi around unit length axis.

Return axis, angle and matching quaternion. The quaternion is given in a slightly irregular order with the angle value preceding the axis information. Most of the literature shows the reverse order but JWST flight software uses the order given here.

Parameters

attitude
[3 by 3 float array] the telescope attitude matrix

Returns

axis
[float array of length 3] a unit vector which is the rotation axis

phi
[float] angle of rotation in degrees

q
[float array of length 4] the equivalent quaternion

pysiaf.utils.rotations.**rotate**(*axis*, *angle*)

Implement fundamental 3D rotation matrices.

Rotate a vector by an angle measured in degrees, about axis 1 2 or 3 in the inertial frame. This is an anti-clockwise rotation when sighted along the axis, commonly called a right-handed rotation.

Parameters

axis
[int] axis number, 1, 2, or 3

angle
[float] angle of rotation in degrees

Returns

r
[float array] a (3 x 3) matrix which performs the specified rotation.

pysiaf.utils.rotations.**rv**(*v2*, *v3*)

Rotate from v2,v3 position to V1 axis.

Rotate so that a V2,V3 position ends up where V1 started.

Parameters

v2
[float] V2 position in arc-sec

v3
[float] V3 position in arc-sec

Returns

rv
[a (3 x 3) array] matrix which performs the rotation described.

pysiaf.utils.rotations.**sky_posangle**(*attitude*, *ra*, *dec*)

Return the V3 angle at arbitrary RA and Dec using the attitude matrix.

This is the angle measured from North to V3 in an anti-clockwise direction.

Parameters

attitude
[3 by 3 float array] the telescope attitude matrix

ra
[float] RA position in degrees

dec
[float] Dec position in degrees

Returns

pa
[float] resulting position angle in degrees

pysiaf.utils.rotations.**sky_to_tel**(*attitude*, *ra*, *dec*, *verbose=False*)

Transform from sky (RA, Dec) to telescope (nu2, nu3) angles.

Return nu2,nu3 position on the idealized focal sphere of any RA and Dec using the inverse of attitude matrix.

Parameters

attitude
[3 by 3 float array] The attitude matrix.

ra
[float (default unit is degree)] RA of sky position

dec
[float (default unit is degree)] Dec of sky position

Returns

nu2, nu3
[tuple of floats with quantity] spherical coordinates at matching position on the idealized focal sphere

pysiaf.utils.rotations.**slew**(*v2t*, *v3t*, *v2a*, *v3a*)

Calculate matrix which slews from target (v2t,v3t) to aperture position (v2a, v3a) without a roll change.

Useful for target acquisition calculations.

Parameters

v2t
[float] Initial V2 position in arc-sec

v3t
[float] Initial V3 position in arc-sec

v2a
[float] Final V2 position in arc-sec

v3a
[float] Final V3 position in arc-sec

Returns

mv
[a (3 x 3) float array] The matrix that performs the rotation described

pysiaf.utils.rotations.**tel_to_sky**(*attitude*, *nu2*, *nu3*, *positive_ra=True*)

Calculate where a nu2,nu3 position points on the sky.

Parameters

attitude
[3 by 3 float array] the telescope attitude matrix

nu2
[float or array of floats (default unit is arcsecond)] V2 coordinate in arc-seconds

nu3
[float or array of floats (default unit is arcsecond)] V3 coordinate in arc-seconds

positive_ra
[bool.] If True forces ra value to be positive

Returns

rd
[tuple of floats with quantity] (ra, dec) - RA and Dec

pysiaf.utils.rotations.**unit**(*ra*, *dec*)

Convert inertial frame vector expressed in polar coordinates / Euler angles to unit vector components.

See Section 5 of JWST-STScI-001550 and Equation 4.1 of JWST-PLAN-006166.

Parameters

ra
[float or array of floats] RA of sky position in degrees

dec
[float or array of floats] Dec of sky position in degrees

Returns

vector
[float array of length 3] the equivalent unit vector in the inertial frame

pysiaf.utils.rotations.**unit_vector_from_cartesian**(*x=None*, *y=None*, *z=None*)

Return unit vector corresponding to two cartesian coordinates.

Array inputs are supported.

Parameters

x
[float or quantity] cartesian unit vector X coordinate in radians

y
[float or quantity] cartesian unit vector Y coordinate in radians

z
[float or quantity] cartesian unit vector Z coordinate in radians

Returns

unit_vector
[numpy.ndarray] Unit vector

pysiaf.utils.rotations.**unit_vector_hst_fgs_object**(*rho*, *phi*)

Return unit vector on the celestial sphere.

This is according to the HST object space angle definitions, CSC/TM-82/6045 1987, Section 4.1.2.2.4

Parameters

rho
[float or array of floats (default unit is degree)] RA of sky position

phi
[float or array of floats (default unit is degree)] Dec of sky position

Returns

vector
[float array of length 3] the equivalent unit vector in the inertial frame

pysiaf.utils.rotations.**unit_vector_sky**(*ra*, *dec*)

Return unit vector on the celestial sphere.

Parameters

ra
[float or array of floats (default unit is degree)] RA of sky position

dec
[float or array of floats (default unit is degree)] Dec of sky position

Returns

vector
[float array of length 3] the equivalent unit vector in the inertial frame

pysiaf.utils.rotations.**v2v3**(*vector*)

Compute v2,v3 polar coordinates corresponding to a unit vector in the rotated (telescope) frame.

See Section 5 of JWST-STScI-001550.

Parameters

vector
[float list or array of length 3] unit vector of cartesian coordinates in the rotated (telescope) frame

Returns

v2, v3
[tuple of floats] The same position represented by V2, V3 values in arc-sec.

1.7.7 siaf

Module to handle Science Instrument Aperture Files (SIAF).

The siaf module defined classes and functions to support working with SIAF files. The main class is ApertureCollection, and the Siaf class inherits from it. The class methods support basic operations and plotting.

ApertureCollection is essentially a container for a set of pysiaf
aperture objects.

Authors

- Johannes Sahlmann

References

Some of the Siaf class methods were adapted from the jwxml package (https://github.com/mperrin/jwxml).

`class pysiaf.siaf.ApertureCollection(aperture_dict=None)`

Structure class for an aperture collection, e.g. read from a SIAF file.

Initialize and generate table of contents.

`generate_toc(attributes=None)`

Generate a table of contents.

`list_apertures(instrument=None, shape=None)`

Print a list of apertures to screen.

`class pysiaf.siaf.Siaf(instrument, filename=None, basepath=None, AperNames=None)`

Science Instrument Aperture File class.

This is a class interface to SIAF information, e.g. stored in an XML file in the PRD. It enables apertures retrieval by name, plotting, and other functionality. See the Aperture class for the detailed implementation of the transformations.

Adapted from https://github.com/mperrin/jwxml

The HST case is treated here as an instrument, because it's single SIAF contains all apertures of all HST-instruments

Examples

fgs_siaf = SIAF('FGS') fgs_siaf.apernames # returns a list of aperture names ap = fgs_siaf['FGS1_FULL'] # returns an aperture object ap.plot(frame='Tel') # plot one aperture fgs_siaf.plot() # plot all apertures in this file

Attributes

observatory
[str] Name of observatory

Read a SIAF from disk.

Parameters

instrument
[string] one of 'NIRCam', 'NIRSpec', 'NIRISS', 'MIRI', 'FGS'; case-insensitive.

basepath
[string] Directory to look in for SIAF files

filename
[string, optional] Alternative method to specify a specific SIAF XML file.

`property apernames`

List of aperture names defined in this SIAF.

`delete_aperture`(*aperture_name*)

Remove an aperture from the Siaf.

Parameters
`aperture_name` – str or list

Returns

`plot`(*frame='tel'*, *names=None*, *label=False*, *units=None*, *clear=True*, *show_frame_origin=None*, *mark_ref=False*, *subarrays=True*, *ax=None*, ***kwargs*)

Plot all apertures in this SIAF.

Parameters

names
[list of strings] A subset of aperture names, if you wish to plot only a subset

subarrays
[bool] Plot all the minor subarrays if True, else just plot the "main" apertures

label
[bool] Add text labels stating aperture names

units
[str] one of 'arcsec', 'arcmin', 'deg'

clear
[bool] Clear plot before plotting (set to false to overplot)

show_frame_origin
[str or list] Plot frame origin (goes to plot_frame_origin()): None, 'all', 'det', 'sci', 'raw', 'idl', or a list of these.

mark_ref
[bool] Add markers for the reference (V2Ref, V3Ref) point in each apertyre

frame
[str] Which coordinate system to plot in: 'tel', 'idl', 'sci', 'det'

ax
[matplotlib.Axes] Desired destination axes to plot into (If None, current axes are inferred from pyplot.)

Other matplotlib standard parameters may be passed in via **kwargs to adjust the style of the displayed lines.

`plot_detector_channels`(*frame=None*, *ax=None*)

Mark on the plot the various detector readout channels.

These are depicted as alternating light/dark bars to show the regions read out by each of the output amps.

Parameters

frame
[str] Which coordinate system to plot in: 'Tel', 'Idl', 'Sci', 'Det' Optional if you have already called plot() to specify a coordinate frame.

ax
[matplotlib.Axes] Desired destination axes to plot into (If None, current axes are inferred from pyplot.)

plot_frame_origin(*frame=None*, *which='sci'*, *units='arcsec'*, *ax=None*)

Mark on the plot the frame's origin in Det and Sci coordinates.

Parameters

frame
[str] Which coordinate system to plot in: 'tel', 'idl', 'sci', 'det' Optional if you have already called plot() to specify a coordinate frame.

which
[str or list] Which origin to plot: 'all', 'det', 'sci', 'raw', 'idl', or a list

units
[str] one of 'arcsec', 'arcmin', 'deg'

ax
[matplotlib.Axes] Desired destination axes to plot into (If None, current axes are inferred from pyplot.)

pysiaf.siaf.**get_jwst_apertures**(*apertures_dict*, *include_oss_apertures=False*, *exact_pattern_match=False*)

Return ApertureCollection that corresponds to constraints specified in apertures_dict.

Parameters

apertures_dict
[dict] Dictionary of apertures

include_oss_apertures
[bool] Whether to include OSS apertures

exact_pattern_match
[bool]

Returns

ApertureCollection
[*ApertureCollection* object] Collection of apertures corresponding to selection criteria

Examples

apertures_dict = {'instrument':['FGS']} apertures_dict['pattern'] = ['FULL']*len(apertures_dict['instrument']) fgs_apertures_all = get_jwst_apertures(apertures_dict)

pysiaf.siaf.**plot_all_apertures**(*subarrays=True*, *showorigin=True*, *detector_channels=True*, ***kwargs*)

Plot all apertures.

pysiaf.siaf.**plot_main_apertures**(*label=False*, *darkbg=False*, *detector_channels=False*, *frame='tel'*, *attitude_matrix=None*, ***kwargs*)

Plot main/master apertures.

Parameters

frame
[string] Either 'tel' or 'sky'. (It does not make sense to plot apertures from multiple instruments in any of the other frames)

attitude_matrix
[3x3 ndarray] Rotation matrix representing observatory attitude. Needed for sky frame plots.

pysiaf.siaf.**plot_master_apertures**(***kwargs*)

Plot only master apertures contours.

1.7.8 tools

A collection of helper functions to support pysiaf calculations and functionalities.

Authors

- Johannes Sahlmann

pysiaf.utils.tools.**an_to_tel**(*xan_arcsec*, *yan_arcsec*)

Convert from XAN, YAN to V2, V3.

pysiaf.utils.tools.**compute_roundtrip_error**(*A*, *B*, *C*, *D*, *offset_x=0.0*, *offset_y=0.0*, *verbose=False*, *instrument=''*, *grid_amplitude=None*)

Return the roundtrip error of the distortion transformations specified by A,B,C,D.

Test whether the forward and inverse idl-sci transformations are consistent.

Parameters

A
[numpy array] polynomial coefficients

B
[numpy array] polynomial coefficients

C
[numpy array] polynomial coefficients

D
[numpy array] polynomial coefficients

offset_x
[float] Offset subtracted from input coordinate

offset_y
[float] Offset subtracted from input coordinate

verbose
[bool] verbosity

instrument
[str] Instrument name

Returns

error_estimation_metric, dx.mean(), dy.mean(), dx.std(), dy.std(), data
[tuple] mean and std of errors, data used in computations

pysiaf.utils.tools.**convert_polynomial_coefficients**(*A_in*, *B_in*, *C_in*, *D_in*, *oss=False*, *inverse=False*, *parent_aperture=None*)

Emulate some transformation made in nircam_get_polynomial_both.

Written by Johannes Sahlmann 2018-02-18, structure largely based on nircamtrans.py code by Colin Cox.

Parameters

A_in
[numpy array] polynomial coefficients

B_in
[numpy array] polynomial coefficients

C_in
[numpy array] polynomial coefficients

D_in
[numpy array] polynomial coefficients

oss
[bool] Whether this is an OSS aperture or not

inverse
[bool] Whether this is forward or backward/inverse transformation

parent_aperture
[str] Name of parent aperture

Returns

AR, BR, CR, DR, V3SciXAngle, V3SciYAngle, V2Ref, V3Ref
[tuple of arrays and floats] Converted polynomial coefficients

pysiaf.utils.tools.**correct_V3SciXAngle**(*V3SciXAngle_deg*)

Correct input angle.

Parameters

V3SciXAngle_deg

Returns

V3SciXAngle_deg
[float]

pysiaf.utils.tools.**correct_V3SciYAngle**(*V3SciYAngle_deg*)

Correct input angle.

Parameters

V3SciYAngle_deg

Returns

V3SciYAngle_deg_corrected
[float]

pysiaf.utils.tools.**get_grid_coordinates**(*n_side*, *centre*, *x_width*, *y_width=None*, *max_radius=None*)

Return tuple of arrays that contain the coordinates on a regular grid.

Parameters

n_side: int
Number of points per side. The returned arrays have n_side**2 entries.

centre: tuple of floats
Center coordinate

x_width: float
Extent of the grid in the first dimension

t_width: float
Extent of the grid in the second dimension

Returns

x
[array]

y
[array]

pysiaf.utils.tools.**is_ipython**()

Function that returns True if the user is in an ipython session and False if they are not

pysiaf.utils.tools.**jwst_fgs_to_fgs_matrix**(*direction='fgs2_to_fgs1'*, *siaf=None*, *verbose=False*)

Return JWST FGS1_OSS to FGS2_OSS transformation matrix as stored in LoadsPRD.

Parameters

siaf
[pysiaf.Siaf instance] JWST FGS SIAF content

Returns

R12
[ndarray] rotation matrix

References

pysiaf.utils.tools.**match_v2v3**(*aperture_1*, *aperture_2*, *verbose=False*, *match_v2_only=False*)

Modify the X[Y]DetRef,X[Y]SciRef attributes of aperture_2 such that V2Ref,V3Ref of both apertures match.

Also shift the polynomial coefficients to reflect the new reference point origin and for NIRCam recalculate angles.

Parameters

aperture_1
[`pysiaf.Aperture object`] Aperture whose V2,V3 reference position is to be used

aperture_2
[`pysiaf.Aperture object`] The V2,V3 reference position is to be altered to match that of aperture_1

verbose
[bool] verbosity

Returns

new_aperture_2: `pysiaf.Aperture object`
An aperture object derived from aperture_2 but with some parameters changed to match altered V2V3.

pysiaf.utils.tools.**revert_correct_V3SciXAngle**(*V3SciXAngle_deg*)

Return corrected V3SciXAngle.

Parameters

V3SciXAngle_deg
[float] Angle in deg

Returns

V3SciXAngle_deg
[float] Angle in deg

pysiaf.utils.tools.**revert_correct_V3SciYAngle**(*V3SciYAngle_deg*)

Return corrected V3SciYAngle.

Only correct if the original V3SciYAngle in [0,180) deg

Parameters

V3SciYAngle_deg
[float] angle in deg

Returns

V3SciYAngle_deg
[float] Angle in deg

pysiaf.utils.tools.**set_reference_point_and_distortion**(*instrument*, *aperture*, *parent_aperture*)

Set V2Ref and V3ref and distortion coefficients for an aperture with a parent_aperture.

Parameters

instrument
[str] Instrument name

aperture
[`pysiaf.Aperture` object] Aperture

parent_aperture
[`pysiaf.Aperture` object] Parent aperture

pysiaf.utils.tools.**tel_to_an**(*v2_arcsec*, *v3_arcsec*)

Convert from V2, V3 to XAN, YAN.

pysiaf.utils.tools.**v3sciyangle_to_v3idlyangle**(*v3sciyangle*)

Convert V3SciYAngle to V3IdlYAngle.

Parameters

v3sciyangle
[float] angle

Returns

v3sciyangle
[float] angle

pysiaf.utils.tools.**write_matrix_to_file**(*matrix*, *file*, *comments=None*, *format='jwst_fsw_patch_request'*)

Write the elements of a matrix to a text file.

Parameters

matrix
[ndarray] the matrix

file
[str] output file name

comments
[dict] comments to include in the commented file header

format
[str] Formatting of matrix elements in output

1.7.9 write

Functions to write Science Instrument Aperture Files (SIAF).

SIAF content in an aperture_collection object can be written to an xml file that can be ingested in the PRD. Format and order of the xml fields are defined in SIAF reference files. Writing to Microsoft Excel .xlsx format is supported. Writing to .csv and other formats supported by astropy.table.Table.write is enabled.

Authors

Johannes Sahlmann

pysiaf.iando.write.**write_jwst_siaf**(*aperture_collection*, *filename=None*, *basepath=None*, *label=None*, *file_format='xml'*, *verbose=True*)

Write the content of aperture_collection into xml and xlsx files that are PRD-compliant.

Parameters

aperture_collection
[ApertureCollection] dictionary of apertures

filename
[str] The file name and path if you do not wish to use the default naming

basepath
[str] If you wish to use the default naming, basepath allows you to set the path where the file will be saved

label
[str] Append default file name ("INSTR_SIAF") with this string

file_format
[str list] one of ['xml', 'xlsx', 'csv', and formats supported by astropy Table.write]

verbose

Returns

filenames
[list] list of the filenames written out

- genindex
- modindex

PYTHON MODULE INDEX

p

pysiaf.aperture, 6
pysiaf.iando.read, 35
pysiaf.iando.write, 53
pysiaf.siaf, 46
pysiaf.utils.compare, 23
pysiaf.utils.polynomial, 26
pysiaf.utils.projection, 34
pysiaf.utils.rotations, 38
pysiaf.utils.tools, 49

INDEX

A

add_rotation() (*in module pysiaf.utils.polynomial*), 26
an_to_tel() (*in module pysiaf.utils.tools*), 49
apernames (*pysiaf.siaf.Siaf property*), 47
Aperture (*class in pysiaf.aperture*), 6
ApertureCollection (*class in pysiaf.siaf*), 46
attitude() (*in module pysiaf.utils.rotations*), 38
attitude_matrix() (*in module pysiaf.utils.rotations*), 39
axial_rotation() (*in module pysiaf.utils.rotations*), 39

C

choose() (*in module pysiaf.utils.polynomial*), 26
closed_polygon_points() (*pysiaf.aperture.Aperture method*), 7
closed_polygon_points() (*pysiaf.aperture.HstAperture method*), 16
compare_apertures() (*in module pysiaf.aperture*), 21
compare_inspection_figures() (*in module pysiaf.utils.compare*), 23
compare_siaf() (*in module pysiaf.utils.compare*), 24
compare_transformation_roundtrip() (*in module pysiaf.utils.compare*), 24
complement() (*pysiaf.aperture.Aperture method*), 7
compute_roundtrip_error() (*in module pysiaf.utils.tools*), 49
compute_tvs_matrix() (*pysiaf.aperture.HstAperture method*), 16
convert() (*pysiaf.aperture.Aperture method*), 7
convert_polynomial_coefficients() (*in module pysiaf.utils.tools*), 49
convert_quantity() (*in module pysiaf.utils.rotations*), 40
corners() (*pysiaf.aperture.Aperture method*), 8
corners() (*pysiaf.aperture.HstAperture method*), 16
corners() (*pysiaf.aperture.NirspecAperture method*), 19
correct_for_dva() (*pysiaf.aperture.Aperture method*), 8
correct_V3SciXAngle() (*in module pysiaf.utils.tools*), 50
correct_V3SciYAngle() (*in module pysiaf.utils.tools*), 50
cross() (*in module pysiaf.utils.rotations*), 40

D

delete_aperture() (*pysiaf.siaf.Siaf method*), 47
deproject_from_tangent_plane() (*in module pysiaf.utils.projection*), 34
det_to_idl() (*pysiaf.aperture.Aperture method*), 8
det_to_raw() (*pysiaf.aperture.Aperture method*), 8
det_to_sci() (*pysiaf.aperture.Aperture method*), 8
det_to_sci() (*pysiaf.aperture.NirspecAperture method*), 19
det_to_sky() (*pysiaf.aperture.Aperture method*), 8
det_to_tel() (*pysiaf.aperture.Aperture method*), 8
detector_transform() (*pysiaf.aperture.Aperture method*), 9
dict_compare() (*in module pysiaf.utils.compare*), 25
distortion_transform() (*pysiaf.aperture.Aperture method*), 9
dms_corner() (*pysiaf.aperture.Aperture method*), 9
dpdx() (*in module pysiaf.utils.polynomial*), 26
dpdy() (*in module pysiaf.utils.polynomial*), 27

F

flatten() (*in module pysiaf.utils.polynomial*), 27
flip_x() (*in module pysiaf.utils.polynomial*), 28
flip_xy() (*in module pysiaf.utils.polynomial*), 28
flip_y() (*in module pysiaf.utils.polynomial*), 28

G

generate_toc() (*pysiaf.siaf.ApertureCollection method*), 46
get_grid_coordinates() (*in module pysiaf.utils.tools*), 50
get_hst_to_jwst_coefficient_order() (*in module pysiaf.aperture*), 22
get_jwst_apertures() (*in module pysiaf.siaf*), 48
get_jwst_siaf_instrument() (*in module pysiaf.iando.read*), 36
get_polynomial_coefficients() (*pysiaf.aperture.Aperture method*), 9

get_polynomial_derivatives() (*pysiaf.aperture.Aperture method*), 9
get_polynomial_linear_parameters() (*pysiaf.aperture.Aperture method*), 10
get_siaf() (*in module pysiaf.iando.read*), 36
getv2v3() (*in module pysiaf.utils.rotations*), 40
gwa_to_ote() (*pysiaf.aperture.NirspecAperture method*), 19
gwa_to_sci() (*pysiaf.aperture.NirspecAperture method*), 19
gwain_to_gwaout() (*pysiaf.aperture.NirspecAperture method*), 19
gwaout_to_gwain() (*pysiaf.aperture.NirspecAperture method*), 20

H

HstAperture (*class in pysiaf.aperture*), 16

I

idl_to_det() (*pysiaf.aperture.Aperture method*), 10
idl_to_raw() (*pysiaf.aperture.Aperture method*), 10
idl_to_sci() (*pysiaf.aperture.Aperture method*), 10
idl_to_sci() (*pysiaf.aperture.NirspecAperture method*), 20
idl_to_sky() (*pysiaf.aperture.Aperture method*), 10
idl_to_tel() (*pysiaf.aperture.Aperture method*), 10
idl_to_tel() (*pysiaf.aperture.HstAperture method*), 17
idl_to_tel_rotation_matrix() (*in module pysiaf.utils.rotations*), 40
invert() (*in module pysiaf.utils.polynomial*), 28
is_ipython() (*in module pysiaf.utils.tools*), 51

J

jacob() (*in module pysiaf.utils.polynomial*), 29
jwst_fgs_to_fgs_matrix() (*in module pysiaf.utils.tools*), 51
JwstAperture (*class in pysiaf.aperture*), 18

L

linear_transform_model() (*in module pysiaf.aperture*), 22
list_apertures() (*pysiaf.siaf.ApertureCollection method*), 46

M

match_v2v3() (*in module pysiaf.utils.tools*), 51
module
    pysiaf.aperture, 6
    pysiaf.iando.read, 35
    pysiaf.iando.write, 53
    pysiaf.siaf, 46
    pysiaf.utils.compare, 23
    pysiaf.utils.polynomial, 26
    pysiaf.utils.projection, 34
    pysiaf.utils.rotations, 38
    pysiaf.utils.tools, 49
month_name_to_number() (*in module pysiaf.iando.read*), 36

N

NirspecAperture (*class in pysiaf.aperture*), 19
number_of_coefficients() (*in module pysiaf.utils.polynomial*), 29

O

ote_to_gwa() (*pysiaf.aperture.NirspecAperture method*), 20

P

path() (*pysiaf.aperture.Aperture method*), 11
plot() (*pysiaf.aperture.Aperture method*), 11
plot() (*pysiaf.siaf.Siaf method*), 47
plot_all_apertures() (*in module pysiaf.siaf*), 48
plot_detector_channels() (*pysiaf.aperture.Aperture method*), 12
plot_detector_channels() (*pysiaf.siaf.Siaf method*), 47
plot_frame_origin() (*pysiaf.aperture.Aperture method*), 12
plot_frame_origin() (*pysiaf.siaf.Siaf method*), 48
plot_main_apertures() (*in module pysiaf.siaf*), 48
plot_master_apertures() (*in module pysiaf.siaf*), 49
pointing() (*in module pysiaf.utils.rotations*), 41
points_on_arc() (*in module pysiaf.aperture*), 22
polar_angles() (*in module pysiaf.utils.rotations*), 41
poly() (*in module pysiaf.utils.polynomial*), 29
polyfit() (*in module pysiaf.utils.polynomial*), 30
polynomial_degree() (*in module pysiaf.utils.polynomial*), 30
posangle() (*in module pysiaf.utils.rotations*), 41
prepend_rotation_to_polynomial() (*in module pysiaf.utils.polynomial*), 30
print_triangle() (*in module pysiaf.utils.polynomial*), 31
project_to_tangent_plane() (*in module pysiaf.utils.projection*), 35
pysiaf.aperture
    module, 6
pysiaf.iando.read
    module, 35
pysiaf.iando.write
    module, 53
pysiaf.siaf
    module, 46
pysiaf.utils.compare
    module, 23
pysiaf.utils.polynomial

module, 26
pysiaf.utils.projection
module, 34
pysiaf.utils.rotations
module, 38
pysiaf.utils.tools
module, 49

R

radec() (*in module pysiaf.utils.rotations*), 42
raw_to_det() (*pysiaf.aperture.Aperture method*), 12
raw_to_idl() (*pysiaf.aperture.Aperture method*), 12
raw_to_sci() (*pysiaf.aperture.Aperture method*), 12
raw_to_tel() (*pysiaf.aperture.Aperture method*), 13
read_hst_fgs_amudotrep() (*in module pysiaf.iando.read*), 36
read_hst_siaf() (*in module pysiaf.iando.read*), 36
read_jwst_siaf() (*in module pysiaf.iando.read*), 36
read_roman_siaf() (*in module pysiaf.iando.read*), 37
read_siaf_alignment_parameters() (*in module pysiaf.iando.read*), 37
read_siaf_aperture_definitions() (*in module pysiaf.iando.read*), 37
read_siaf_ddc_mapping_reference_file() (*in module pysiaf.iando.read*), 37
read_siaf_detector_layout() (*in module pysiaf.iando.read*), 37
read_siaf_detector_reference_file() (*in module pysiaf.iando.read*), 37
read_siaf_distortion_coefficients() (*in module pysiaf.iando.read*), 38
read_siaf_xml_field_format_reference_file() (*in module pysiaf.iando.read*), 38
rearrange_fgs_alignment_parameters() (*pysiaf.aperture.HstAperture method*), 17
reference_point() (*pysiaf.aperture.Aperture method*), 13
reorder() (*in module pysiaf.utils.polynomial*), 31
rescale() (*in module pysiaf.utils.polynomial*), 31
revert_correct_V3SciXAngle() (*in module pysiaf.utils.tools*), 51
revert_correct_V3SciYAngle() (*in module pysiaf.utils.tools*), 52
rodrigues() (*in module pysiaf.utils.rotations*), 42
RomanAperture (*class in pysiaf.aperture*), 21
rotate() (*in module pysiaf.utils.rotations*), 42
rotation_from_derivatives() (*in module pysiaf.utils.polynomial*), 32
rotation_scale_skew_from_derivatives() (*in module pysiaf.utils.polynomial*), 32
rv() (*in module pysiaf.utils.rotations*), 42

S

scale_from_derivatives() (*in module pysiaf.utils.polynomial*), 32
sci_to_det() (*pysiaf.aperture.Aperture method*), 13
sci_to_det() (*pysiaf.aperture.NirspecAperture method*), 20
sci_to_gwa() (*pysiaf.aperture.NirspecAperture method*), 20
sci_to_idl() (*pysiaf.aperture.Aperture method*), 13
sci_to_idl() (*pysiaf.aperture.NirspecAperture method*), 21
sci_to_raw() (*pysiaf.aperture.Aperture method*), 13
sci_to_sky() (*pysiaf.aperture.Aperture method*), 14
sci_to_tel() (*pysiaf.aperture.Aperture method*), 14
sci_to_tel() (*pysiaf.aperture.NirspecAperture method*), 21
set_attitude_matrix() (*pysiaf.aperture.Aperture method*), 14
set_distortion_coefficients_from_file() (*pysiaf.aperture.Aperture method*), 14
set_idl_reference_point() (*pysiaf.aperture.HstAperture method*), 17
set_polynomial_coefficients() (*pysiaf.aperture.Aperture method*), 14
set_reference_point_and_distortion() (*in module pysiaf.utils.tools*), 52
set_tel_reference_point() (*pysiaf.aperture.HstAperture method*), 18
shift_coefficients() (*in module pysiaf.utils.polynomial*), 32
show_save_plot() (*in module pysiaf.utils.compare*), 25
Siaf (*class in pysiaf.siaf*), 46
SiafCoordinates (*class in pysiaf.aperture*), 21
sky_posangle() (*in module pysiaf.utils.rotations*), 43
sky_to_det() (*pysiaf.aperture.Aperture method*), 14
sky_to_idl() (*pysiaf.aperture.Aperture method*), 14
sky_to_sci() (*pysiaf.aperture.Aperture method*), 14
sky_to_tel() (*in module pysiaf.utils.rotations*), 43
sky_to_tel() (*pysiaf.aperture.Aperture method*), 14
slew() (*in module pysiaf.utils.rotations*), 43

T

tel_to_an() (*in module pysiaf.utils.tools*), 52
tel_to_det() (*pysiaf.aperture.Aperture method*), 14
tel_to_idl() (*pysiaf.aperture.Aperture method*), 14
tel_to_idl() (*pysiaf.aperture.HstAperture method*), 18
tel_to_raw() (*pysiaf.aperture.Aperture method*), 15
tel_to_sci() (*pysiaf.aperture.Aperture method*), 15
tel_to_sci() (*pysiaf.aperture.NirspecAperture method*), 21
tel_to_sky() (*in module pysiaf.utils.rotations*), 44
tel_to_sky() (*pysiaf.aperture.Aperture method*), 15
telescope_transform() (*pysiaf.aperture.Aperture method*), 15
to_distortion_model() (*in module pysiaf.aperture*), 23

transform_coefficients() (*in module pysiaf.utils.polynomial*), 33
triangular_layout() (*in module pysiaf.utils.polynomial*), 33
two_step() (*in module pysiaf.utils.polynomial*), 33

U

unit() (*in module pysiaf.utils.rotations*), 44
unit_vector_from_cartesian() (*in module pysiaf.utils.rotations*), 44
unit_vector_hst_fgs_object() (*in module pysiaf.utils.rotations*), 45
unit_vector_sky() (*in module pysiaf.utils.rotations*), 45

V

v2v3() (*in module pysiaf.utils.rotations*), 45
v3sciyangle_to_v3idlyangle() (*in module pysiaf.utils.tools*), 52
validate() (*pysiaf.aperture.Aperture method*), 16
verify() (*pysiaf.aperture.Aperture method*), 16

W

write_jwst_siaf() (*in module pysiaf.iando.write*), 53
write_matrix_to_file() (*in module pysiaf.utils.tools*), 52