astrocut Documentation

Release 0.5

MAST Archive Developers

Jan 13, 2020

CONTENTS

Tools for making image cutouts from sets of images with shared footprints.

This package is under active development, and will ultimately grow to encompass a range of cutout activities relevant to images from many missions. Currently there are two modes of interaction:

- Solving the specific problem of creating image cutouts from sectors of TESS full frame images (FFIs) (`CubeFactory` and `CutoutFactory`)
- More generalized cutouts from sets of images with the same WCS/pixel scale (`fits_cut`)

Astrocut lives on GitHub at: github.com/spacetelescope/astrocut.

Part I

Documentation

CHAPTER

ONE

INSTALLATION

1.1 Requirements

Astrocut has the following requirements:

- Astropy
- NumPy

1.2 Installing astrocut

1.2.1 Using pip

The easiest way to install Astrocut is using pip:

```
pip install astrocut
```

1.2.2 From source

To install the bleading edge version from github without donloading, run the following command:

```
pip git+https://github.com/spacetelescope/astrocut.git
```

The latest development version of astrocut can be cloned from github using this command:

```
git clone https://github.com/spacetelescope/astrocut.git
```

To install astrocut (from the root of the source tree):

```
python setup.py install
```

CHAPTER

TWO

ASTROCUT DOCUMENTATION

2.1 Introduction

Astrocut contains tools for creating image cutouts from sets images with shared footprints. This package is under active development, and will ultimately grow to encompass a range of cutout activities relevant to images from many missions. Currently there are two modes of interaction:

- Solving the specific problem of creating image cutouts from sectors of TESS full frame images (FFIs) (`CubeFactory` and `CutoutFactory`)
- More generalized cutouts from sets of images with the same WCS/pixel scale (`fits_cut`)

2.2 TESS Full-Frame Image Cutouts

There are two parts of the package involved in this task, the `CubeFactory` class allows you to create a large image cube from a list of FFI files. This is what allows the cutout operation to be performed efficiently. The `CutoutFactory` class performs the actual cutout and builds a target pixel file (TPF) that is compatible with TESS pipeline TPFs.

The basic workflow is to first create an image cube from individual FFI files (this is one-time work), and then make individual cutout TPFs from this large cube file. If you are doing a small number of cutouts, it may make sense for you to use our tesscut web service: mast.stsci.edu/tesscut

2.2.1 Making image cubes

Making an image cube is a simple operation, but comes with a vert important limitation:

> **Warning: Memory Requirements**
>
> The entire cube file must be able to fit in your computer's memory!
>
> For a sector of TESS FFI images from a single camera/chip combination this is ~50 GB.

This operation can also take some time to run. For the 1348 FFI images of the TESS ete-6 simulated sector, it takes about 12 minutes to run on a computer with 65 GB of memory.

By default `make_cube` runs in verbose mode and prints out its progress, however setting verbose to false will silence all output.

```
>>> from astrocut import CubeFactory
>>> from glob import glob
>>> from astropy.io import fits
>>>
>>> my_cuber = CubeFactory()
>>> input_files = glob("data/*ffic.fits")
>>>
>>> cube_file = my_cuber.make_cube(input_files)
Completed file 0
Completed file 1
Completed file 2
.
.
.
Completed file 142
Completed file 143
Total time elapsed: 46.42 sec
File write time: 8.82 sec

>>> print(cube_file)
img-cube.fits

>>> cube_hdu = fits.open(cube_file)
>>> cube_hdu.info()
Filename: img-cube.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
0  PRIMARY       1 PrimaryHDU      28   ()
1                1 ImageHDU         9   (2, 144, 2136, 2078)   float32
2                1 BinTableHDU    302   144R x 147C   [24A, J, J, J, J, J, J, D, 24A, J, 24A, 24A, J, J,
↪ D, 24A, 24A, 24A, J, D, 24A, D, D, D, D, 24A, 24A, D, D, D, D, D, 24A, D, D, D, D, J, D, D, D, D, D,␣
↪D, D, D, D, D, D, D, J, J, D, J, J, J, J, J, J, J, J, J, J, D, J, J, J, J, J, J, D, J, J, J, J, J, J,␣
↪D, J, J, J, J, J, J, D, J, J, J, J, J, J, J, J, J, 24A, D, J, 24A, 24A, D, D, D, D, D, D, D, D, J, J, D,␣
↪D, D, D, D, D, J, J, D, D, D, D, D, D, D, D, D, D, D, D, 24A, J, 24A, 24A, J, J, D, 24A, 24A, J, J, D,
↪ D, D, D, J, 24A, 24A, 24A]
```

2.2.2 Making cutout target pixel files

To make a cutout, you must already have an image cube to cut out from. Assuming that that step has been completed, you simply give the central coordinate and cutout size (in either pixels or angular `Quanitity`) to the *cube_cut* function.

You can either specify a target pixel file name, or it will be built as: "<cube_file_base>_<ra>_<dec>_<cutout_size>_astrocut.fits". You can optionally also specify a output path, the directory in which the target pixel file will be saved, if unspecified it defaults to the current directory.

```
>>> from astrocut import CutoutFactory
>>> from astropy.io import fits
>>>
>>> my_cutter = CutoutFactory()
>>> cube_file = "img-cube.fits"
>>>
>>> cutout_file = my_cutter.cube_cut(cube_file, "251.51 32.36", 5, verbose=True)
Cutout center coordinate: 251.51,32.36
xmin,xmax: [26 31]
ymin,ymax: [149 154]
Image cutout cube shape: (144, 5, 5)
Uncertainty cutout cube shape: (144, 5, 5)
```

(continues on next page)

(continued from previous page)

```
Target pixel file: img_251.51_32.36_5x5_astrocut.fits
Write time: 0.016 sec
Total time: 0.18 sec

>>> cutout_hdu = fits.open(cutout_file)
>>> cutout_hdu.info()
Filename: img_251.51_32.36_5x5_astrocut.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
0  PRIMARY       1 PrimaryHDU      42   ()
1  PIXELS        1 BinTableHDU    222   144R x 12C   [D, E, J, 25J, 25E, 25E, 25E, 25E, J, E, E, 38A]
2  APERTURE      1 ImageHDU        45   (5, 5)   float64
```

Note: The 'TIME' column of the cutout table is formed by taking the average of the TSTART, TSTOP values from the corresponding FFI for each row.

A note about the cutout WCS object

TESS FFIs are large and therefore are described by WCS objects that have many non-linear terms. Astrocut creates a new simpler (linear) WCS object from the matched set of cutout pixel coordinates and sky coordinates (from the FFI WCS). This linear WCS object will generally work very well, however at larger cutout sizes (100-200 pixels per side and above) the linear WCS fit will start to be noticeably incorrect at the edges of the cutout. Three header keywords have been added to the PIXELS extension to give additional information about the cutout WCS:

- **WCS_FFI:** The name of the FFI file used to build the original WCS from which the cutout and cutout WCS were calculated.
- **WCS_MSEP:** The maximum separation in degrees between the cutout's linear WCS and the FFI's full WCS.
- **WCS_SIG:** The error in the cutout's linear WCS, calculated as sqrt((dist(Po_ij, Pl_ij)^2) where dist(Po_ij, Pl_ij) is the angular distance in degrees between the sky position of of pixel i,j in the original full WCS and the new linear WCS.

2.3 General fits image cutouts (beta!)*

These functions provide general purpose fits cutout functionality, returning the results either as fits cutout files, or as images (jpg or png). An image normalization (`normalize_img`) function is also available.

The function `fits_cut` takes one or more fits files and performs the same cutout in each, returning the result either in a single fits file or as one fits file per cutout. It is important to remember that while the expectation is that all input image are aligned and have the same pixel scale, no checking is done.

```
>>> from astrocut import fits_cut
>>> from astropy.io import fits
>>> from astropy.coordinates import SkyCoord

>>> input_files = ["https://archive.stsci.edu/pub/hlsp/candels/cosmos/cos-tot/v1.0/hlsp_candels_hst_acs_
↪cos-tot-sect23_f606w_v1.0_drz.fits",
...                "https://archive.stsci.edu/pub/hlsp/candels/cosmos/cos-tot/v1.0/hlsp_candels_hst_acs_
↪cos-tot-sect23_f814w_v1.0_drz.fits"]

>>> center_coord = SkyCoord("150.0945 2.38681", unit='deg')
>>> cutout_size = [200,300]

>>> cutout_file = fits_cut(input_files, center_coord, cutout_size, drop_after="", single_outfile=True)
```

(continues on next page)

(continued from previous page)

```
>>> print(cutout_file)
./cutout_150.094500_2.386810_200-x-300_astrocut.fits

>>> cutout_hdulist = fits.open(cutout_file)
>>> cutout_hdulist.info()
Filename: ./cutout_150.094500_2.386810_200-x-300_astrocut.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      11   ()
  1  CUTOUT        1 ImageHDU        44   (200, 300)   float32
  2  CUTOUT        1 ImageHDU        44   (200, 300)   float32
```

The function `img_cut` takes one or more fits files and performs the same cutout in each, returning the result either an image (jpg or png) per cutout, or a single color image. It is important to remember that while the expectation is that all input images are aligned and have the same pixel scale, no checking is done.

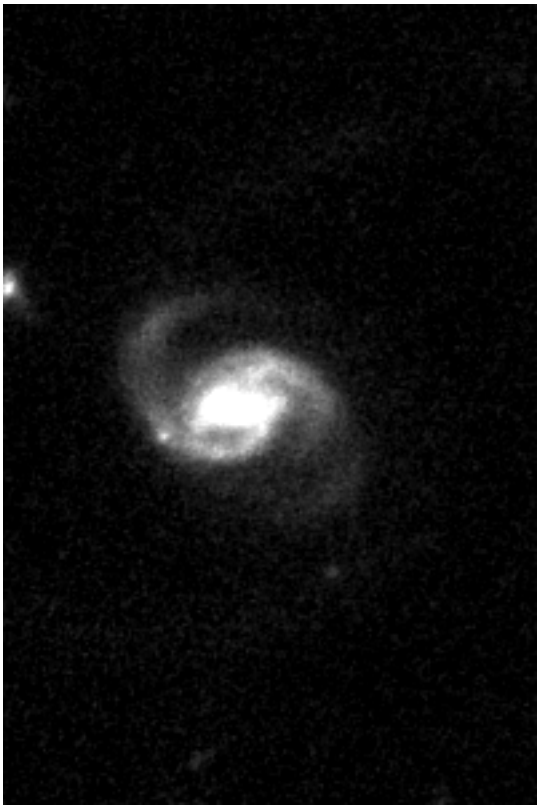
```
>>> from astrocut import img_cut
>>> from astropy.coordinates import SkyCoord
>>> from PIL import Image

>>> input_files = ["https://archive.stsci.edu/pub/hlsp/candels/cosmos/cos-tot/v1.0/hlsp_candels_hst_acs_
↪cos-tot-sect23_f606w_v1.0_drz.fits",
...                "https://archive.stsci.edu/pub/hlsp/candels/cosmos/cos-tot/v1.0/hlsp_candels_hst_acs_
↪cos-tot-sect23_f814w_v1.0_drz.fits"]

>>> center_coord = SkyCoord("150.0945 2.38681", unit='deg')
>>> cutout_size = [200,300]

>>> png_files = img_cut(input_files, center_coord, cutout_size, img_format='png', drop_after="")
>>> print(png_files[0])
./hlsp_candels_hst_acs_cos-tot-sect23_f606w_v1.0_drz_150.094500_2.386810_200-x-300_astrocut.png

>>> Image.open(png_files[1])
```



Color images can also be produced using `img_cut` given three input files, which will be treated as the R, G, and B channels respectively.

```
>>> from astrocut import img_cut
>>> from astropy.coordinates import SkyCoord
>>> from PIL import Image

>>> input_files = ["https://archive.stsci.edu/pub/hlsp/goods/v2/h_nz_sect14_v2.0_drz_img.fits",
...                "https://archive.stsci.edu/pub/hlsp/goods/v2/h_ni_sect14_v2.0_drz_img.fits",
...                "https://archive.stsci.edu/pub/hlsp/goods/v2/h_nv_sect14_v2.0_drz_img.fits"]

>>> center_coord = SkyCoord("189.51522 62.2865221", unit='deg')
>>> cutout_size = [200,300]

>>> color_image = img_cut(input_files, center_coord, cutout_size, colorize=True)
>>> print(color_image)
./cutout_189.515220_62.286522_200-x-300_astrocut.jpg

>>> Image.open(color_image)
```

* This is the newest functionality and as such should be considered to be in beta. It has been tested primarily on Hubble deep field drizzled images. We welcome issues and pull-requests to make it more widely functional.

2.4 astrocut Package

2.4.1 Functions

fits_cut(input_files, coordinates, cutout_size)	Takes one or more fits files with the same WCS/pointing, makes the same cutout in each file, and returns the result either in a single fitsfile with one cutout per extension or in individual fits files.
img_cut(input_files, coordinates, cutout_size)	Takes one or more fits files with the same WCS/pointing, makes the same cutout in each file, and returns the result either as a single color image or in individual image files.
normalize_img(img_arr[, stretch, . . .])	Apply given stretch and scaling to an image array.

fits_cut

astrocut.**fits_cut**(*input_files*, *coordinates*, *cutout_size*, *correct_wcs=False*, *drop_after=None*, *single_outfile=True*, *cutout_prefix='cutout'*, *output_dir='.'*, *verbose=False*)

Takes one or more fits files with the same WCS/pointing, makes the same cutout in each file, and returns the result either in a single fitsfile with one cutout per extension or in individual fits files.

Note: No checking is done on either the WCS pointing or pixel scale. If images don't line up the cutouts will also not line up.

Parameters

input_files
[list] List of fits image files to cutout from. The image is assumed to be in the first extension.

coordinates
[str or `SkyCoord` object] The position around which to cutout. It may be specified as a string ("ra dec" in degrees) or as the appropriate `SkyCoord` object.

cutout_size
[int, array-like, `Quantity`] The size of the cutout array. If `cutout_size` is a scalar number or a scalar `Quantity`, then a square cutout of `cutout_size` will be created. If `cutout_size` has two elements, they should be in `(ny, nx)` order. Scalar numbers in `cutout_size` are assumed to be in units of pixels. `Quantity` objects must be in pixel or angular units.

correct_wcs
[bool] Default False. If true a new WCS will be created for the cutout that is tangent projected and does not include distortions.

drop_after
[str or None] Default None. When creating the header for the cutout (and crucially, before building the WCS object) drop all header keywords starting with the one given. This is useful particularly for drizzle files that contain a multitude of extranious keywords and sometimes leftover WCS keywords that astropy will try to parse even thought they should be ignored.

single_outfile
[bool] Default True. If true return all cutouts in a single fits file with one cutout per extension, if False return cutouts in individual fits files. If returing a single file the filename will have the form: <cutout_prefix>_<ra>_<dec>_<size x>_<size y>.fits. If returning multiple files each will be named: <original filemame base>_<ra>_<dec>_<size x>_<size y>.fits.

cutout_prefix
[str] Default value "cutout". Only used if single_outfile is True. A prefix to prepend to the cutout filename.

output_dir
[str] Defaul value '.'. The directory to save the cutout file(s) to.

verbose
[bool] Default False. If true intermediate information is printed.

Returns

response
[str or list] If single_outfile is True returns the single output filepath. Otherwise returns a list of all the output filepaths.

img_cut

astrocut.**img_cut**(*input_files*, *coordinates*, *cutout_size*, *stretch='asinh'*, *minmax_percent=None*, *minmax_value=None*, *invert=False*, *img_format='jpg'*, *colorize=False*, *cutout_prefix='cutout'*, *output_dir='.'*, *drop_after=None*, *verbose=False*)

Takes one or more fits files with the same WCS/pointing, makes the same cutout in each file, and returns the result either as a single color image or in individual image files.

Note: No checking is done on either the WCS pointing or pixel scale. If images don't line up the cutouts will also not line up.

Parameters

input_files
[list] List of fits image files to cutout from. The image is assumed to be in the first extension.

coordinates
[str or `SkyCoord` object] The position around which to cutout. It may be specified as a string ("ra dec" in degrees) or as the appropriate `SkyCoord` object.

cutout_size
[int, array-like, `Quantity`] The size of the cutout array. If `cutout_size` is a scalar number or a scalar `Quantity`, then a square cutout of `cutout_size` will be created. If `cutout_size` has two elements, they should be in `(ny, nx)` order. Scalar numbers in `cutout_size` are assumed to be in units of pixels. `Quantity` objects must be in pixel or angular units.

stretch
[str] Optional, default 'asinh'. The stretch to apply to the image array. Valid values are: asinh, sinh, sqrt, log, linear

minmax_percent
[array] Optional, default [0.5,99.5]. Interval based on a keeping a specified fraction of pixels (can be asymmetric) when scaling the image. The format is [lower percentile, upper percentile], where pixel values below the lower percentile and above the upper percentile are clipped. Only one of minmax_percent and minmax_value should be specified.

minmax_value
[array] Optional. Interval based on user-specified pixel values when scaling the image. The format is [min value, max value], where pixel values below the min value and above the max value are clipped. Only one of minmax_percent and minmax_value should be specified.

invert
[bool] Optional, default False. If True the image is inverted (light pixels become dark and vice versa).

img_format
[str] Optional, default 'jpg'. The output image file type. Valid values are "jpg" and "png".

colorize
[bool] Optional, default False. If True a single color image is produced as output, and it is expected that three files are given as input.

cutout_prefix
[str] Default value "cutout". Only used when producing a color image. A prefix to prepend to the cutout filename.

output_dir
[str] Defaul value '.'. The directory to save the cutout file(s) to.

verbose
[bool] Default False. If true intermediate information is printed.

Returns

response
[str or list] If colorize is True returns the single output filepath. Otherwise returns a list of all the output filepaths.

normalize_img

astrocut.**normalize_img**(*img_arr*, *stretch='asinh'*, *minmax_percent=None*, *minmax_value=None*, *invert=False*)

Apply given stretch and scaling to an image array.

Parameters

img_arr
[array] The input image array.

stretch
[str] Optional, default 'asinh'. The stretch to apply to the image array. Valid values are: asinh, sinh, sqrt, log, linear

minmax_percent
[array] Optional. Interval based on a keeping a specified fraction of pixels (can be asymmetric) when scaling the image. The format is [lower percentile, upper percentile], where pixel values below the lower percentile and above the upper percentile are clipped. Only one of minmax_percent and minmax_value shoul be specified.

minmax_value
[array] Optional. Interval based on user-specified pixel values when scaling the image. The format is [min value, max value], where pixel values below the min value and above the max value are clipped. Only one of minmax_percent and minmax_value should be specified.

invert
[bool] Optional, default False. If True the image is inverted (light pixels become dark and vice versa).

Returns

response
[array] The normalized image array, in the form in an integer arrays with values in the range 0-255.

2.4.2 Classes

CubeFactory	Class for creating image cubes.
CutoutFactory()	Class for creating image cutouts.

CubeFactory

class astrocut.**CubeFactory**

Bases: object

Class for creating image cubes.

This class emcompasses all of the cube making functionality. In the current version this means creating image cubes fits files from TESS full frame image sets. Future versions will include more generalized cubing functionality.

Methods Summary

make_cube(self, file_list[, cube_file, ...])	Turns a list of fits image files into one large data-cube.

Methods Documentation

make_cube(*self*, *file_list*, *cube_file='img-cube.fits'*, *sector=None*, *verbose=True*)

Turns a list of fits image files into one large data-cube. Input images must all have the same footprint and resolution. The resulting datacube is transposed for quicker cutouts. This function can take some time to run and requires enough memory to hold the entire cube in memory. (For full TESS sectors this is about 40 GB)

Parameters

file_list
[array] The list of fits image files to cube. Assumed to have the format of a TESS FFI: - A primary HDU consisting only of a primary header - An image HDU containing the image - A second image HDU containing the uncertainty image

cube_file
[string] Optional. The filename/path to save the output cube in.

sector
[int] Optional. TESS sector to add as header keyword (not present in FFI files).

verbose
[bool] Optional. If true intermediate information is printed.

Returns

response: string or None
If successful, returns the path to the cube fits file, if unsuccessful returns None.

CutoutFactory

class astrocut.**CutoutFactory**

Bases: object

Class for creating image cutouts.

This class emcompasses all of the cutout functionality. In the current version this means creating cutout target pixel files from TESS full frame image cubes. Future versions will include more generalized cutout functionality.

Initiazation function.

Methods Summary

cube_cut(self, cube_file, coordinates, ...)	Takes a cube file (as created by CubeFactory), and makes a cutout target pixel file of the given size around the given coordinates.

Methods Documentation

cube_cut(*self*, *cube_file*, *coordinates*, *cutout_size*, *target_pixel_file=None*, *output_path='.'*, *verbose=False*)

Takes a cube file (as created by CubeFactory), and makes a cutout target pixel file of the given size around the given coordinates. The target pixel file is formatted like a TESS pipeline target pixel file.

Parameters

cube_file
[str] The cube file containing all the images to be cutout. Must be in the format returned by ~astrocut.make_cube.

coordinates
[str or astropy.coordinates.SkyCoord object] The position around which to cutout. It may be specified as a string ("ra dec" in degrees) or as the appropriate SkyCoord object.

cutout_size
[int, array-like, Quantity] The size of the cutout array. If cutout_size is a scalar number or a scalar Quantity, then a square cutout of cutout_size will be created. If cutout_size has two elements, they should be in (ny, nx) order. Scalar numbers in cutout_size are assumed to be in units of pixels. Quantity objects must be in pixel or angular units.

target_pixel_file
[str] Optional. The name for the output target pixel file. If no name is supplied, the file will be named: <cube_file_base>_<ra>_<dec>_<cutout_size>_astrocut.fits

output_path
[str] Optional. The path where the output file is saved. The current directory is default.

verbose
[bool] Optional. If true intermediate information is printed.

Returns

response: string or None
If successfull, returns the path to the target pixel file, if unsuccessful returns None.

CHAPTER

THREE

LICENSE

3.1 AstroCut License

AstroCut is licensed under a 3-clause BSD style license:

Copyright (c) 2018, MAST Archive Developers All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Astropy Team nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

PYTHON MODULE INDEX

INDEX