disco_stu Documentation

Release 1.3.6

Chris Simpson

Oct 25, 2019

CONTENTS

CHAPTER

ONE

OVERVIEW OF DISCO-STU

Disco-Stu (*DIStortion COrrection and STacking Utility*) has been written to aid in the stacking of images taken with the Gemini South Adaptive Optics Imager, GSAOI, but can be used to align and stack images from other instruments. It aims to produce science-quality images from partially-processed (dark-subtracted and flat-fielded) input frames.

Disco-Stu undertakes four steps:

1. Modifying the world coordinate system (WCS) of the input (GSAOI) images to account for the *static distortion* in the instrument focal plane.
2. Astrometric matching between the different input images (and, if supplied, an external reference catalog) to determine the *variable distortion*.
3. Reprojecting the input images to a common astrometric frame.
4. Stacking the images (with bad pixel rejection and inverse-variance weighting, if desired).

The user can specify options on the command line to control Disco-Stu's operation, including the parameters of the source matching, whether to perform sky subtraction, and some parameters of the output image (orientation and pixel scale or, alternatively, copying the world coordinate system from an existing image), although the defaults are expected to be the best options in most cases.

Disco-Stu was written by Chris Simpson, based on and incorporating code from an unreleased package called *gsaoi_discorr*, written by Mark Simpson.

CHAPTER

TWO

INSTALLING DISCO-STU

Note: These instructions assume that you are running python version 2.7. If you are running a different version, the software will be installed in a different "pythonX.Y" subdirectory, rather than "python2.7". You can determine your python version by typing `python --version` from the command line.

It is recommended to install the software in a location other than the standard Python location for modules (the default 'site-packages'). This is also the only solution if you do not have write permission to the default 'site-packages'. Here is how you install the software somewhere other than the default location:

```
$ python setup.py install --prefix=/your/favorite/location
```

`/your/favorite/location` must already exist. This command will install executable scripts in a 'bin' subdirectory, the documentation in a `share` subdirectory, and the modules in a `lib/python2.7/site-packages` subdirectory.

Because you are not using the default location, you will need to add two paths to your environment. You might want to add the following to your .cshrc or .bash_profile, or equivalent shell configuration script.

C shell(csh, tcsh):

```
setenv PATH /your/favorite/location/bin:${PATH}
setenv PYTHONPATH /your/favorite/location/lib/python2.7/site-packages:${PYTHONPATH}
```

Bourne shells (sh, bash, ksh, . . .):

```
export PATH=/your/favorite/location/bin:${PATH}
export PYTHONPATH=/your/favorite/location/lib/python2.7/site-packages:${PYTHONPATH}
```

If you added those lines to your shell configuration script, make sure you source the file to activate the new setting.

For csh/tcsh:

```
$ source ~/.cshrc
$ rehash
```

For bash:

```
$ source ~/.bash_profile
```

Disco-Stu uses a number of python packages, such as numpy and astropy, which must be installed and visible within your `PYTHONPATH`. These are included in the Ureka distribution. Testing has been performed with numpy v1.9.1 and astropy v1.0, as distributed with Ureka, on CentOS 7 and Mac OSX El Capitan, and with astropy v1.2. Note that the parallelization may not work under OSX and may need to be switched off using the `--serial` flag. Disco-Stu will check your numpy build at runtime and warn you if it believes you need to set this flag.

Disco-Stu calculates a variable distortion correction by matching sources between images. These source catalogs are created using SExtractor, which must therefore be installed on your system. Version 2.8.6 is the minimum requirement. Alternatively, you may provide source catalogs for the input images as described later in this document.

Disco-Stu has been tested with python-2.7 and 3.5.

CHAPTER

THREE

PREPARING DATA

Disco-Stu is not an end-to-end data reduction package, and expects its input data to have been processed, including dark subtraction (if needed), flat-fielding, and sky subtraction (if flat-fielding has not flattened the sky). Furthermore, in order to maintain the photometric integrity of the data when stacking, all images, and all extensions in multi-extension images, should be on a single photometric scale, i.e., pixels with the same data values should always represent the same flux.

3.1 Specific steps for preparing GSAOI data

The following procedure using Gemini IRAF will appropriately prepare raw GSAOI images. Alternative reduction methods are available, and possibly superior.

In the following steps, it is assumed that your images can be divided into *flatfield images*, *science images*, and *sky images*. For uncrowded fields, the sky images may be the same as the science images.

1. Run all files through `gprepare`.
2. Combine the *flatfield images* using `gaflat` with `statextn="DETECTOR"`. This is not the default parameter setting, and normalizes the flatfield across all four arrays, thereby accounting for the detector-to-detector quantum efficiency variations.
3. Combine the *sky images* using `gasky`.
4. Reduce all *science images* using `gareduce`, with `fl_flat=yes`, `fl_sky=yes`, and `fl_mult=no`. Setting `fl_mult=no` keeps the pixel values in ADU, which is essential because the gain variation between arrays has been accounted for by the flatfield. You should set `fl_autosky=no` if you want your final images to be sky-subtracted.

You should set `fl_vardq=yes` in all these tasks if you want to propagate the variance and data quality information.

CHAPTER

FOUR

USING DISCO-STU

You run Disco-Stu using the command `disco <filenames>` where `<filenames>` is a wildcard-enabled list of FITS files you wish to process. The behaviour can be extensive controlled using the various command-line options described in detail below, with additional control possible through editing the `lookups/general_parameters.py` file.

Output files will be automatically overwritten.

4.1 Command-line options

4.1.1 Output image

-o <name>, --output <name> Set the output filename. The `.fits` extension is not required. If not supplied, the final image will be written to the file `disco_stack.fits`.

-a <angle>, --pa <angle> Set the position angle of the positive y-axis of the output image. By default this is set to be the same as that of the first input image.

-p <scale>, --pixel_scale <scale> Set the pixel scale of the output image. The value should be given in arcseconds per pixel and defaults to 0.0195.

-w <image>, --wcs <image> Adopt the world coordinate system from an existing image. This overrides the `-a` and `-p` flags and will produce an image of the same size and with the same WCS as the specified image. If the file contains multiple images, the first will be used. If the output image extends beyond the region of sky covered by the specified image, it will be cropped.

--reference <image> Use the specified image as the reference image. All other images will have their object catalogs matched to that of the reference image. If not reference image is specified, the first file in the input list will be used.

4.1.2 Reference catalog

-r <file>, --refcat <file> Use the supplied file as an astrometric reference catalog. This must be readable by `astropy.table.Table.read()`

--refcat_format <format> Format of the reference catalog, if it cannot be auto-identified. This will be passed as a parameter directly to `Table.read()` and so must be one of the built-in table reader formats

--refcat_columns <ra,dec> Names of the table columns (separated by a comma) to use for the right ascension and declination of reference sources. By default these are RA and DEC. Note that SExtractor uses X_WORLD and Y_WORLD.

4.1.3 Source matching

-d <degree>, --degree <degree> Degree (order) of the polynomial used to calculate the variable distortion correct by matching object catalogs. Default is 3.

-m <number>, --min_matches <number> Minimum number of matches required between each input's source catalog and the reference catalog for a variable distortion correction to be applied. In order for the fit to be constrained, this *must* be equal to or greater than (d+1)(d+2)/2, where d is the degree of the polynomial.

--search_radius <r1[,r2]> Radii (in arcseconds) for reference catalog matching. The first value is used to determine the systemic pointing offset via cross-correlation. The second value is optional and is used to determine one-to-one source matches to calculate the variable distortion correction. Defaults are 5.0 and 0.5. Note that the matching radii between images are controlled by a lookup file, as detailed in the next chapter.

--ignore_objcat If present, this flag will cause a new object catalog to be created for each input image, even if such catalogs are present in the input data.

4.1.4 Stacking

-c <operation>, --combine <operation> Mathematical operation for combining the input images. These correspond directly to methods of the `numpy.MaskedArray` class and allowable values are `mean`, `average` (weighted mean), or `median`. In the case of `median`, the variance will not be propagated. The default is `average`.

--no_align Stack the images *only*. No alignment or sky-subtraction will be performed, and only the `-c`, `-l`, and `-o` options are relevant. This flag is intended for use with input images that have already been aligned and reprojected to investigate the results of different stacking operations without repeating those computationally-intensive steps.

--no_skysub By default, Disco-Stu will estimate the sky level in each array and subtract it from the data. Set this flag to turn off that behavior. Note that if the input images have a `SKYMIDPT` header keyword (which is inserted by `gareduce` if it is run with `fl_autosky=yes`), then the reprojected image will retain this background level.

--no_stack Set this flag to turn off image stacking, and only perform distortion correction and reprojection of the input images.

4.1.5 Bad pixel mitigation

--clean During the reprojection, a single highly discrepant pixel can have a significant effect on its neighbors due to 'ringing' from the interpolation. To eliminate this effect, it is possible to clean the input images by replacing bad pixels with the median value of their neighbors. The pixel is still flagged as bad and will not be used in the stacking.

--clean_iter <iterations> Maximum number of cleaning iterations to perform. If this is specified, cleaning will occur even if `-clean` is not invoked. The default is 1.

--clean_radius <radius> The radius within which to select pixels for the median filter. The default value of 1.5 selects all pixels within a 3x3 box. The bad pixel itself is excluded.

4.1.6 Miscellaneous

--serial By default, Disco-Stu will spawn subprocesses during the image reprojection, one per detector, which significantly speeds up its operation. Setting this flag disables this behavior, which is necessary when running on OSX if your numpy installation has been built against the Accelerate framework. You can check whether this is the case on your system with `numpy.__config__.show()`

-l <file>, --logfile <file> Change the name of the output log file.

-v, --version Display the version of Disco-Stu being run, and exit.

4.2 Object catalogs

Although Disco-Stu can produce its own object catalogs (via SExtractor) to determine the variable distortion, it is possible to include object catalogs with the input images that will be used instead (unless the `--ignore_objcat` flag is used). If you wish to do this, the catalog *must* be included in the input images, with each SCI extension having a corresponding OBJCAT extension containing a table with at least two columns called `X_IMAGE` and `Y_IMAGE`. If such an extension cannot be found, Disco-Stu will create its own object catalog. In addition, if a reference catalog is supplied, it is recommended that at least the catalogs for the first input image have a `FLUX_AUTO` column, to enable the brightest sources to be selected for matching to the reference catalog, as described in the Detailed Operation section.

CHAPTER

FIVE

DETAILED OPERATION

This section describes the operation of Disco-Stu. It is highly recommended reading for all users of the software as you should understand the processing steps being applied to your data.

5.1 General ideas

Disco-Stu works by creating a reversible chain of transformations or mappings between coordinate frames (all instances of `astropy.model.Model`). Some of these frames have a clear meaning (e.g., pixel coordinates, or celestial coordinates) while others are simply intermediate steps. Fundamentally, there are four major parts to this chain:

1. **Input pixel frame to nominal celestial frame**. For GSAOI, this is determined via a lookup table describing the static distortion, coupled with various image header keywords describing the telescope pointing and instrument position angle. For other instruments, this will simply be each image's world coordinate system (WCS), as read from the image header.
2. **Nominal celestial frame to reference image celestial frame**. This is calculated by matching individual sources between each image and the first image in the input list. A two-dimensional polynomial is used, with the user able to select the order. Source matching is undertaken in two distinct steps:
    1. A cross-correlation between the reference catalog and each image's source catalog to determine the systemic RA and DEC offsets.
    2. A nearest-source direct matching between catalogs, after applying the above offset.

    As each image is matched, the reference image's source catalog is augmented with unmatched sources. The coordinates of these sources in the reference image celestial frame are used. This allows the next step to use sources which lie outside the field of view of the first input image.
3. **Reference image celestial frame to reference catalog celestial frame**. If an external reference catalog is supplied, a match is undertaken between the reference catalog and the augmented source catalog. This follows the same two-step process outlined above. Since it will often be the case that the science images are much deeper than the reference catalog, the source catalog for each image is culled to include only the brightest $2n$ sources, where n is the number of sources in the reference catalog. This should prevent (bright) reference sources being matched to much fainter sources in the input image.
4. **Reference catalog celestial frame to output pixel frame**. This is handled simply as a gnomonic (tangent) projection, with the reference point being the nominal telescope pointing and user-specifiable orientation and pixel scale.

Linking these three transformations into a chain provides a mapping between input pixels and output pixels, and vice versa. Each output image is then constructed by interpolation across the input image at the coordinates that map to each pixel in the output.

5.2 Specific issues

This section provides additional details on certain aspects of Disco-Stu's operation.

5.2.1 Sky subtraction

Sky subtraction, if requested, is undertaken by subtracting a constant value from each extension. Unflagged pixels in the SCI plane (i.e., those with DQ=0) are 1-in-10 sampled (for speed) and a Gaussian function is fit to the histogram of these values between (-5,+1) standard deviations from the median. The mean of this Gaussian is adopted as the sky level, which is subtracted when the data are reprojected.

5.2.2 Catalog matching

Catalog matching (where OBJCAT-to-OBJCAT or OBJCAT-to-REFCAT) is performed in two stages. First, an overall offset is determine via cross-correlation, and then individual sources are matched between frames. All matching is performed in the celestial coordinate frame.

The cross-correlation method involves creating a synthetic 'landscape' image, where a Gaussian source is placed on the image at the location of each entry in the reference catalog. Initially, this synthetic image has a pixel scale equal to 1/20 of the offset search radius, and each Gaussian has a FWHM of 23.5 pixels (sigma of 10 pixels). Over the range of offsets under consideration, the quality of fit is determined by summing the pixels in the landscape image at the locations off all the shifted input source catalog positions. The offset with the highest quality of fit is adopted and applied to all the input source positions.

If the pixel scale of this synthetic image is larger than the required precision (set to the size of a GSAOI pixel, i.e., 0.02 arcseconds), then a new landscape is constructed with pixels five times smaller (in linear dimension), but with Gaussians the same size in pixels. This process is repeated as required.

The second stage is a simple nearest-neighbor source match. This provides the reference coordinates at a series of input coordinates, to which a transformation function can be fit.

There are four parameters used to determine the alignment transformation:

1. `OFFSET_RADIUS`: The search radius for determining the systemic offset between the two source catalogs.
2. `MATCH_RADIUS`: The search radius for finding individual source-to-source nearest-neighbor matches.
3. `POLY_DEGREE`: The order of the 2D polynomial used to represent the transformation. Such a polynomial has (d+1)(d+2)/2 parameters, and so there must be at least this many source matches for the fit to be constrained. If not, the order is repeatedly lowered until the fit is constrained.
4. `MIN_MATCHES`: The minimum number of source-to-source matches required to compute a transformation. No transformation is applied if there are fewer matches than this, and it can therefore be used to provided a minimum polynomial order. For fits of order 1,2,3,4,5 the required number of matches are 3,6,10,15,21. A value of `None` allows the polynomial to reduce to first order and is effectively equivalent to a value of 3 (as are values less than 3).

These parameters have different values for OBJCAT-to-OBJCAT transformations and OBJCAT-to-REFCAT transformations. For OBJCAT-to-OBJCAT (where the uncertainty should be due inaccuracies in the telescope offsetting if all the input images were taken in a single observation) the default values are 1.0,0.3,3,None while for OBJCAT-to-REFCAT they are 5,0.5,2,None. Command-line switches can change the polynomial order for OBJCAT-to-OBJCAT matches and the other values for OBJCAT-to-REFCAT matches. Further alteration of these parameters requires editing the `lookups/general_parameters.py` file.

5.2.3 Interpolation

The pixel values in each reprojected output image are interpolated from the corresponding input image using the `scipy.ndimage.geometric_transform` function. This differs from the method of Drizzle but is appropriate since variations in the sky coverage of each pixel are accounted for by flatfielding, and therefore the input images represent the average surface brightness seen by each pixel, rather than the total flux. The default interpolation is a cubic spline, but this can be changed by editing the value of `INTERPOLATION_ORDER` in the `lookups/general_parameters.py` file.

Since `geometric_transform` does not handle bad pixel masks, the spline interpolation it uses can result in bad pixels with very discrepant values influencing the pixels around them. To mitigate this effect, it is possible to clean the image prior to interpolation, replacing the value of pixels flagged as 'bad' (i.e., with the lowest DQ bit set) with the median value of the pixels around them. Two parameters, `-clean_radius` and `-clean_iter`, control the cleaning. The first sets the radius within which to use pixels for the median operation (the bad pixel itself is not used), while the second allows multiple iterations of the cleaning to take place. After each iteration, if fewer than half the pixels used to calculate the median were bad, the pixel will no longer be marked as bad. Multiple iterations therefore allow large groups of connected bad pixels to have their values replaced by eating in from the outside. However, the original bad pixel mask is preserved so these new pixel values are not used in the final science image.

5.2.4 Bad pixels

The data quality (DQ) planes in Gemini data have individual bits set to represent different issues with the data. The lowest (1) bit represents a known bad pixel, the next (2) bit indicates a pixel is in the non-linear regime, while the 4 bit indicates it is saturated, etc. The DQ image is therefore effectively a combination of many images, one for each bit that is used, and each of these should be transformed independently and then recombined. This would significantly increase the computation time needed, however. Furthermore, when stacking the images, pixels will either be flagged as 'good' (so included in the stack) or 'bad' (so not) and the reason why they are flagged is irrelevant. It therefore makes sense to make this good/bad selection prior to transformation, and a pixel will be flagged as bad if any bit other than the non-linear (2) bit is set. This is controlled via the `DQ_BITMASK` value in `lookups.general_parameters`, which is combined with the individual DQ pixel values via a bitwise AND, and a non-zero result treated as a bad pixel.

Since there is no one-to-one mapping between input and output pixels, there is no definitive way to flag bad pixels in the output image. The option chosen here is to transform the DQ image (having set all bad pixels to a value of 1) in the same way as the SCI image, and then flag output pixels whose values excess a threshold, defined by the `MIN_DQ_BADNESS` value in `lookups.general_parameters`. This has the default value 0.02.

5.2.5 Object catalogs

The individual input images will all have object catalogs, either provided by the user or (more commonly) produced by SExtractor during the task's execution. These are stored as FITS extensions named `OBJCAT`, one per array, following the Gemini convention. When the images are reprojected, they are merged into a single table, and additional columns are added that may aid in debugging if the alignment and/or reprojection is unsatisfactory.

- `RA`, `DEC` are the celestial coordinates determined from the static transform *only*.
- `mRA`, `mDEC` are the celestial coordinates of the object to which this source has been matched. For the first input image, these will be the coordinates of the source in the reference catalog if one has been supplied, otherwise they will all have dummy values of `-99`. For subsequent images, these will be the fully-transformed coordinates of the source in the first image.
- `tRA`, `tDEC` are the fully-transformed coordinates in the final output image, incorporating the static correction, the alignment to the first input image, and the alignment to the reference catalog.
- `rRA`, `rDEC` are the residuals in arcseconds, namely the difference between the `m` and `t` columns.
- `tX_IMAGE`, `tY_IMAGE` are the pixel coordinates in the transformed (output) image.

- oX_IMAGE, oY_IMAGE are the pixel coordinates in the original image that map to these coordinates in the output image. These should (obviously) be very close to the original X_IMAGE, Y_IMAGE values.

Only the individual reprojected output images, and not the final stacked image, have OBJCAT tables.