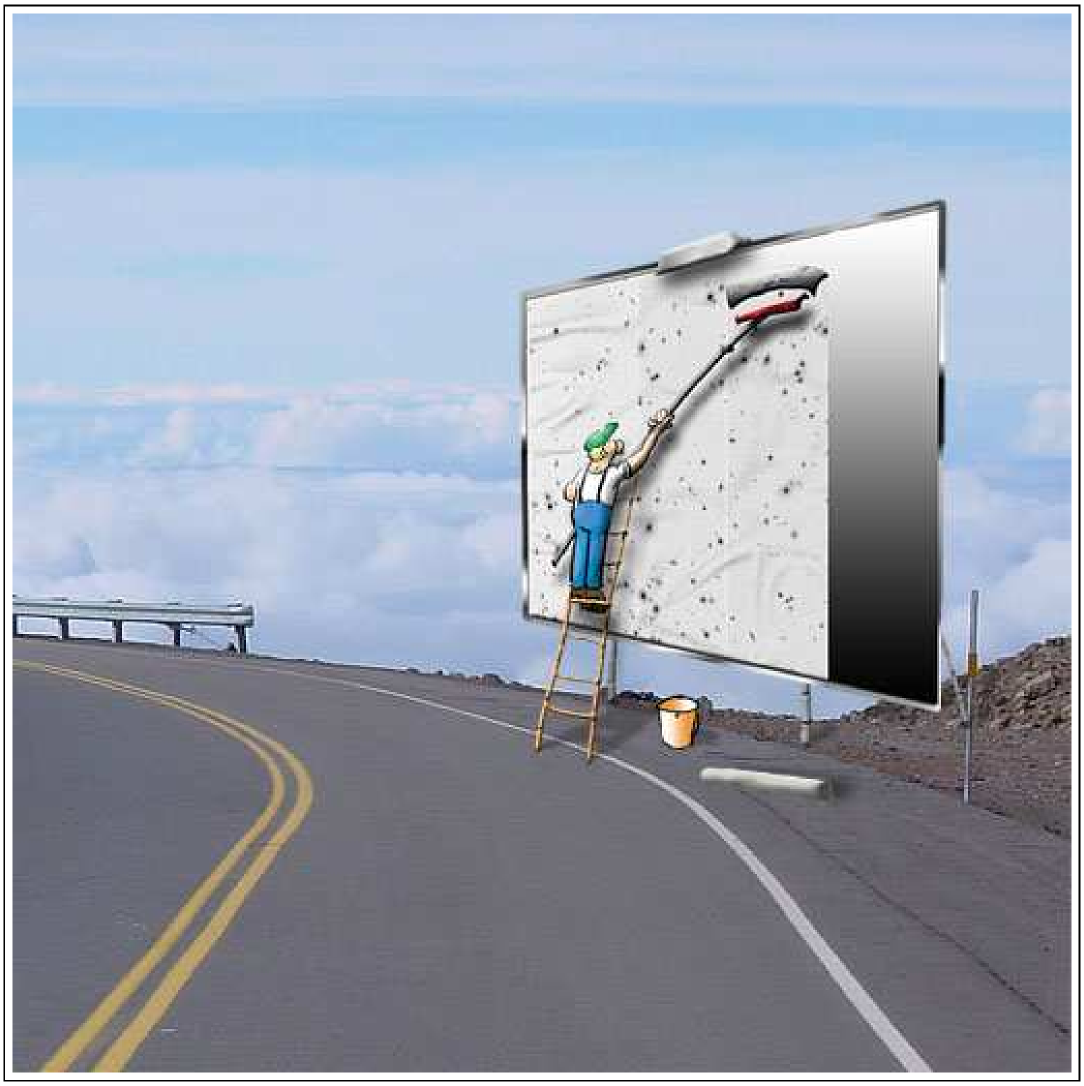
SWarp
v1.21
User's guide

E. BERTIN
Institut d'Astrophysique & Observatoire de Paris

April 17, 2001



Contents

1 What is SWarp?

SWarp is a program that resamples and co-adds together FITS images using any arbitrary astrometric projection defined in the WCS standard[1]. The main features of SWarp are:

- FITS format (including multi-extensions) in input and output,
- Full handling of weight-maps in input and output,
- Ability to work with very large images (up to 500 Mpixels on 32-bit machines and 10^6 Tpixels with 64-bits), thanks to customized virtual-memory-mapping and buffering,
- Works with arrays in up to 9 dimensions (including or not two spherical coordinates),
- Selectable high-order interpolation method (up to 8-tap filters) in any dimension,
- Compatible with WCS and TNX (IRAF) astrometric descriptions,
- Support for equatorial, galactic and equatorial coordinate systems,
- Astrometric and photometric parameters are read from FITS headers or external ASCII files,
- Built-in background subtraction,
- Built-in noise-level measurement for automatic weighting,
- Automatic centering and sizing functions of the output field.

2 Skeptical Sam's questions

Skeptical Sam doesn't have time to test software extensively but is always keen on asking agressive questions to the author to find out if a program could fit his needs.

S.Sam: What's the point in releasing another image co-addition software? We already have the *Drizzle* and *MSCRED* (IRAF) packages, for instance.

Author: Co-addition is most-certainly the most critical step in reducing modern CCD mosaic data. Although several powerful packages are already available, they did not meet all the requirements we have at the TERAPIX data center here in Paris.

S.Sam: SWarp doesn't perform the astrometric calibration, does it?

Author: No it doesn't. The astrometric calibration software of the TERAPIX pipeline shall be released in future versions of the software. In the meanwhile, any polynomial description will do; you may use IRAF for instance.

S.Sam: I am the kind of guy who does high precision astrometry and photometry. My sources are detectable on raw frames. Resampling, you know, would just wreck the signal, so I prefer to combine measurements from individual images than to co-add pixels.

Author: It is true that resampling distorts slightly both signal and noise. However, if done properly, and if the data are not undersampled (Full Width at Half Maximum > 2.5 pixels), the degradation is generally very small. For instance, resampling a single ground-based image

[1] see `http://www.cv.nrao.edu/fits/documents/wcs/wcs.html`

(atmospheric seeing of FWHM 3 pixels, and conversion factor $2e^-/\mathrm{ADU}$) twice with SWarp, and comparing fluxes and positions measured using PSF-fitting, one gets for bright stars *rms* differences of less than 5.10^{-4} mag and 10^{-3} pixel between the original and the twice-resampled images. This is already much better than what photon-noise allows for. Hence apart from situations of strongly non-stationary noise or undersampled data, the consequences of resampling are expected to be negligible.

3 Installing the software

3.1 Software and hardware requirements

SWarp has been developed on Unix machines (Compaq True-64 and GNU/Linux), and should compile on any POSIX-compliant system. The software is run in (ANSI) text-mode from a shell. A window system is therefore unnecessary with present versions.

(Silicon) memory requirements are fairly modest in most cases, as they do not depend on the size of the output images. 100MB is sufficient when co-adding images from current CCD chips ($2k\times4.5k$). More memory may be helpful for co-adding bigger maps. Although the built-in virtual-memory feature will almost always allow one to work with any image size, the performance hit caused by file-swapping may be important in some cases.

3.2 Obtaining SWarp

The easiest way to obtain SWarp is to download it from an internet site. The current official anonymous FTP site is `ftp://ftp.iap.fr/pub/from_users/bertin/swarp/`. There can be found the latest versions of the program as standard `.tar.gz` Unix archives, including the documentation.

3.3 Installation

To install, you must first uncompress and unarchive the archive:

```
gzip -dc swarp_x.x.tar.gz | tar xvf -
```

A new directory called `swarpx.x` should now appear at the current position on your disk. You should then just enter the directory and follow the instructions in the file called "`INSTALL`".

4 Using SWarp

SWarp is run from the shell with the following syntax:

% `swarp` *Input_image1*`[`*,Input_image2,...*`]` `-c` *configuration-file* `[ -`*Parameter1 Value1* `]` `[ -`*Parameter2 Value2 ...*`]`

The part enclosed within brackets is optional. Any "`-`*Parameter Value*" statement in the command-line overrides the corresponding definition in the configuration-file or any default value (see below).

4.1 The Configuration file

Each time SWarp is run, it looks for a configuration file. If no configuration file is specified in the command-line, it is assumed to be called "`default.swarp`" and to reside in the current directory.

4.1.1 Format of the configuration file

The format is ASCII. There must be only one parameter set per line, following the form:

Config-parameter *Value(s)*

Extra spaces or linefeeds are ignored. Comments must begin with a "#" and end with a linefeed. Values can be of different types: strings (can be enclosed between double quotes), floats, integers, keywords or boolean (`Y/y` or `N/n`). Some parameters accept zero or several values, which must then be separated by commas. Integers can be given as decimals, in octal form (preceded by digit `0`), or in hexadecimal (preceded by `0x`). The hexadecimal format is particularly convenient for writing multiplexed bit values such as binary masks. Environment variables, written as `$HOME` or `${HOME}` are expanded.

4.1.2 Configuration parameter list

Here is a list of all the parameters known to SWarp. Please refer to next section for a detailed description of their meaning.

Parameter	default	type	Description
`BACK_DEFAULT`	0.0	*floats* ($n \leq n_{\rm ima}$)	Default background value to be subtracted in `BACK_TYPE MANUAL` mode.
`BACK_FILTERSIZE`	—	*integers* ($n \leq n_{\rm ima}$)	Size (in background meshes) of the background-filtering mask.
`BACK_FILTTHRESH`	—	*integers* ($n \leq n_{\rm ima}$)	Difference threshold (in ADUs) for the background-filtering.
`BACK_SIZE`	—	*integers* ($n \leq n_{\rm ima}$)	Size (in pixels) of a background mesh.
`BACK_TYPE`	`AUTO`	*keywords* ($n \leq n_{\rm ima}$)	What background is subtracted from the images:
		`AUTO`	– the internal interpolated background-map,
		`MANUAL`	– a user-supplied constant value provided in `BACK_DEFAULT`.
`CELESTIAL_TYPE`	`NATIVE`	*keyword*	Celestial coordinate system in output:
		`NATIVE`	– Same as first input file,
		`EQUATORIAL`	– Equatorial α, δ coordinates,
		`GALACTIC`	– Galactic l, b coordinates
		`ECLIPTIC`	– Ecliptic λ, β coordinates
`CENTER_TYPE`	`ALL`	*keywords* ($n \leq n_{\rm dim}$)	The way SWarp centers the output frame:
		`ALL`	– Center on the region that contains all input fields,
		`MOST`	– Center on the region with most overlap between input fields,

		MANUAL	– Manual centering using the CENTER parameter.
CENTER	0.0	*strings* ($n \le n_{\rm dim}$)	Position of the center in CENTER_TYPE MANUAL mode. Can be given in floating point notation, in *hh:mm:ss* (for right ascension/longitude), or *dd:mm:ss* (for declination/latitude).
COMBINE_TYPE	MEDIAN	*keyword*	The way SWarp combines resampled images:
		MEDIAN	– Take the median of pixel values,
		AVERAGE	– Take the average,
		MIN	– Take the minimum,
		MAX	– Take the maximum,
		WEIGHTED	– Take the weighted average.
		CHI2	– Take the weighted, quadratic sum.
DELETE_TMPFILES	Y	*boolean*	If true, resampled, temporary image files are deleted.
FSCALASTRO_TYPE	FIXED	*keyword*	The way SWarps computes the astrometric part of the flux-scaling:
		NONE	– Ignore the effects of re-projection,
		FIXED	– Apply a fixed correction based on the ratio of pixel scales.
FSCALE_DEFAULT	1.0	*floats* ($n \le n_{\rm ima}$)	Default fluxscale to adopt for each image if the FSCALE_KEYWORD keyword is not found in the FITS header.
FSCALE_KEYWORD	FLXSCALE	*string*	FITS keyword that should contain the flux scale in input images.
GAIN_DEFAULT	0.0	*floats* ($n \le n_{\rm ima}$)	Default gain (conversion factor in e^-/ADU) to adopt for each image if the GAIN_KEYWORD keyword is not found in the FITS header. 0 means "infinite".
GAIN_KEYWORD	GAIN	*string*	FITS keyword that should contain the gain in input images.
HEADER_SUFFIX	.head	*string*	Extension of the external ASCII "headers" that shall be seeked to override internal FITS parameters.
IMAGEOUT_NAME	coadd.fits	*string*	Name of the output image file.
IMAGE_SIZE	—	*integers* ($n \le n_{\rm dim}$)	Dimensions of the output image (in PIXELSCALE_TYPE MANUAL or FIT mode).
MEM_MAX	128	*integer*	Maximum amount of megabytes allowed for (silicon) memory storage.
OVERSAMPLING	0	*integers* ($n \le n_{\rm dim}$)	Amount of oversampling in each dimension (0 means "automatic".
PIXEL_SCALE	—	*floats* ($n \le n_{\rm dim}$)	Step between pixels in each dimension (in PIXELSCALE_TYPE MANUAL mode). For angular coordinates, it must be expressed in arcseconds.

PIXELSCALE_TYPE	MEDIAN	*keywords* ($n \leq n_{\rm dim}$)	The way SWarp sets the output pixel size:
		MEDIAN	– Take the median of pixel scales at the center of input frames,
		MIN	– Take the minimum of pixel scales at the center of input frames,
		MAX	– Take the maximum of pixel scales at the center of input frames,
		MANUAL	– User-defined pixel scale at image center (with the PIXEL_SCALE keyword),
		FIT	– Compute the pixel scale in order to have the full output field fitting the user-defined IMAGE_SIZE.
PROJECTION_TYPE	TAN	*string*	Projection system used in output, in standard WCS notation (see Table 1).
RESAMPLING_TYPE	LANCZOS3	*keywords* ($n \leq n_{\rm dim}$)	Resampling method:
		NEAREST	– Take the nearest neighbour,
		BILINEAR	– Bi-linear interpolation,
		LANCZOS2	– Lanczos-2 4-tap filter,
		LANCZOS3	– Lanczos-3 6-tap filter,
		LANCZOS4	– Lanczos-4 8-tap filter.
SUBTRACT_BACK	Y	*booleans* ($n \leq n_{\rm ima}$)	If true, input images are background-subtracted prior to resampling.
VMEM_DIRECTORY	/tmp	*string*	Path of the directory where virtual-memory and other temporary files are written.
VMEM_MAX	2048	*integer*	Maximum amount of megabytes allowed for virtual-memory storage.
WEIGHT_IMAGE		*strings* ($n \leq n_{\rm ima}$)	List of input weight-maps.
WEIGHTOUT_NAME	coadd.fits	*string*	File name of the output weight-map.
WEIGHT_THRESH		*floats* ($n \leq n_{\rm ima}$)	Threshold below or above which input weights are equivalent to zero (infinite variance, i.e. a bad pixel).
WEIGHT_TYPE	NONE	*keywords* ($n \leq n_{\rm ima}$)	Type of input weight-maps:
		NONE	– no weighting,
		MAP_WEIGHT	– relative weights (i.e. inverse variance),
		MAP_VARIANCE	– relative variance,
		MAP_RMS	– absolute standard deviation.
WRITE_FILEINFO	Y	*boolean*	If true, extended information about input files is written in the header of the output FITS image.
VERBOSE_TYPE	NORMAL	*keyword*	How much SWarp comments its operations:
		QUIET	– run silently,
		NORMAL	– display warnings and limited info concerning the work in progress,
		FULL	– display more complete information.

5 How SWarp works

5.1 Overview of the software

What SWarp does is basically to read a set of input FITS images, resample and combine them, and finally save the resultant FITS image to disk. The work can be decomposed in several steps. In a first pass, input image headers are read and checked for content. If configured in fully automatic mode, SWarp will set the characteristics of the output frame based on this information. In a second pass, input images (and their weight-maps, if available) are read one-by-one. A background-map is built, and subtracted from the image if required. In the third and last pass, the image is resampled and projected into a subsection of the output frame, which is saved as a temporary FITS file. A temporary, "projected" weight-map is created too, even if no weight-map was given in input. Finally, a co-added output image is created using the information stored in the temporary weight-maps, and consisting of a composite of the resampled sub-sections. A composite output weight-map is also written in the process.

The global layout of SWarp is presented in Fig. 1. Let us now describe each of the important steps.

5.2 Image mapping and memory constraints

How does SWarp projects input images into the output frame space? There are two ways of applying a geometric transformation to an image (see Wolberg 1992). The most intuitive is called "forward mapping". It consists in scanning the input image pixel-per-pixel, line-by-line. Each pixel is simply "thrown" to the position it is supposed to occupy in the output grid. Although this technique can be used for geometric resampling or "drizzling" (Fruchter & Hook 1997), it is totally cumbersome with high order interpolation techniques. "Inverse mapping" is far more efficient in this case. In this procedure the output frame is scanned pixel-per-pixel and line-by-line. Using the inverse projection, each output pixel center is associated a position in the input frame, at which the image is interpolated. This technique has been implemented in SWarp (Fig. 2); it possesses several advantages. The output image is accessed sequentially, and thus can be arbitrarily large. Also, only positions corresponding to pixels (or sub-pixels) within the output frame have to be mapped.

The most potentially critical part is the pseudo-random access in the input image. In most cases, it will be an individual imaging array (like an individual CCD) and will therefore fit into memory. For much larger input images however we rely on the efficiency of virtual-memory-mapping. Swarp's virtual memory engine works in the following way: each input image, stored as a single precision, 4-byte array is loaded in physical memory if the required amount of megabytes doesn't exceed `MEM_MAX`. If it does, a temporary file called *vmxxxxx_xxxxx.tmp* is written in the directory specified by `VMEM_DIRECTORY`. The program exits with an error message if this file would exceed `VMEM_MAX` Megabytes. The 3 "memory" parameters are mostly hardware-dependent. It is advised to set `MEM_MAX` to 50-100% of the actual amount of memory present in your machine. If disk-space is not the limiting factor, `VMEM_MAX` should be set to a higher value, 2048 on a 32-bit machine, or even more on a 64-bit machine. The choice of the `VMEM_DIRECTORY` is critical. First, as this write-enabled directory is used to store all temporary files, it must be large enough to contain all resampled input images in floating-point format. Second, it is strongly recommended to have the data on a fast disk. Note that the default path for `VMEM_DIRECTORY` is */tmp*, which on many systems is on a partition simply not large enough to handle the typical quantities of data to process. An alternative is to use ".", the current directory, at the expense of disk thrashing!

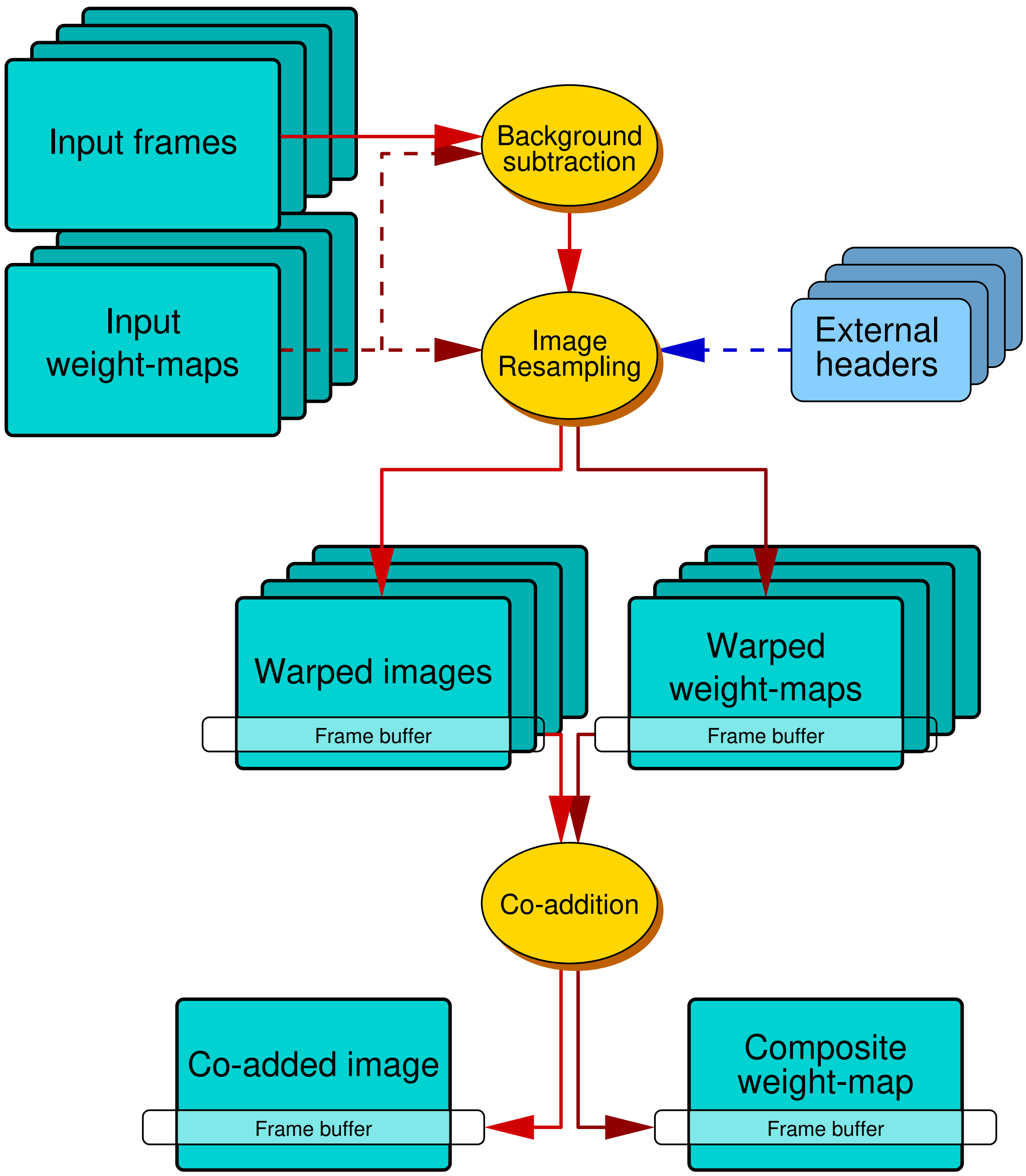


Figure 1: Global Layout of SWarp.

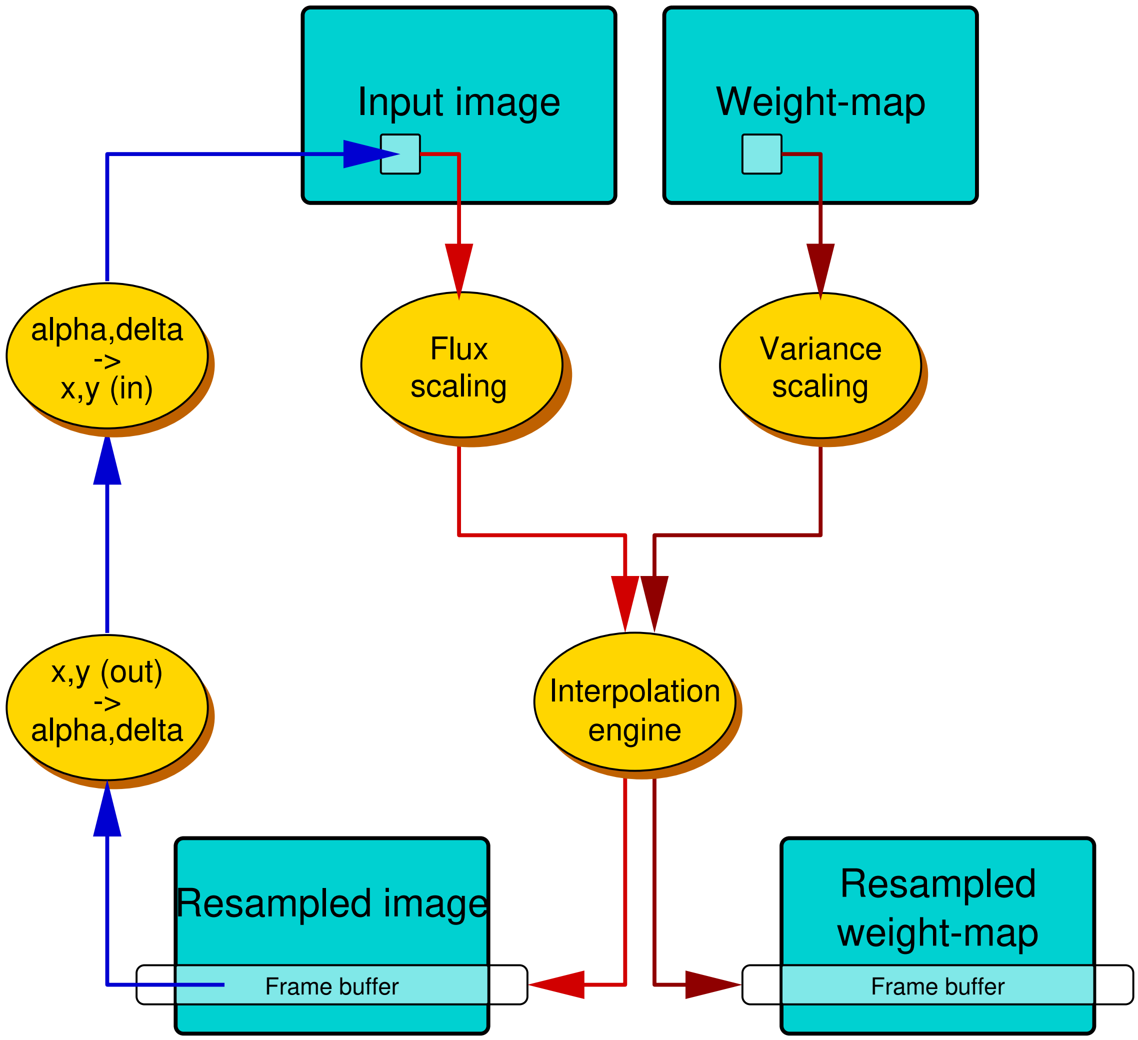


Figure 2: Layout of the image mapping section.

De-projecting and co-adding simultaneously all input images would frequently imply many files open at the same time, and large amounts of (virtual) memory. SWarp takes a more sequential approach: each input image is mapped in a tightly-fitting rectangular subsection of the output frame. All subsections are written to disk (in the `VMEM_DIRECTORY`) as *swarp.xxx.fits* FITS files, and read back later during the co-addition phase, to be stacked together. Individual subsection files are automatically removed after processing or abortion. It is possible to disable the deletion by setting the `DELETE_TMPFILES` configuration parameter to `N` (the default is `Y`). This can be useful for diagnostic purposes.

Note that although SWarp does not use much memory, the amount of temporary disk-space needed during processing can be quite large. In addition to the output image and weight-map, one should provide disk-space for the individual projected images and their weight-maps. In the case of mappings done at unit scale, this involves storing more than twice the amount of input pixels as temporary data.

5.3 Astrometry

The astrometric engine at the heart of SWarp is based on M. Calabretta's *WCSlib* library[2], to which we added the handling of polynomial distorsion parameters (FITS keywords `PV_xx_xx`) as proposed in the latest WCS documents[3]. We included IRAF's `TNX` astrometric projection too (for inputs only), although it is not part of the WCS standard.

All celestial coordinate computations are performed in the equatorial system. Galactic or ecliptic coordinates are supported in input and output.

5.3.1 Input frames

(De-)projection parameters of input images are extracted from their respective FITS headers. These are the usual `CTYPEx` `CRVALx`, `CRPIXx`, `CDELTx` and/or `CD_xx_xx` WCS parameters. External "header" files can also be provided by the user; for every input *xxxx.fits* image, SWarp looks for a *xxxx.head* header file, and loads it if present. A *.head* suffix is the default; it can be changed using the `HEADER_SUFFIX` configuration parameter. External headers may either be real FITS header cards (no carriage-return), or ASCII files containing lines in FITS-like format, with the final line starting with "`END␣␣␣␣␣`". Multiple extensions must be separated by an "`END␣␣␣␣␣`" line. External "headers" need not contain all the FITS keywords normally required. The keywords present in external headers are only there to override their counterparts in the original image headers, or add new ones.

5.3.2 Output frames

The celestial pair of components of the output coordinate system is specified with the `CELESTIAL_TYPE` configuration parameter, and can be selected among `NATIVE`, `EQUATORIAL`, `GALACTIC` and `ECLIPTIC`. In `NATIVE` mode, the output celestial coordinate system is taken from that of the *first* file of the input list.

The output (celestial) projection is set by the `PROJECTION_TYPE` configuration parameter. The list of all presently supported projections is shown in Table 1, and illustrated in Fig. 3 and 4 using a gridded map of the Earth.

Now, what projection is the best? With "small" fields (<10 degrees in their maximum dimension), the choice is not critical as long as the projection center lies within the frame. For compatibility reasons, it is advised to stay with the traditional gnomonic (`TAN`, for tangential) projection in such cases. With larger fields the pure tangential projection is inappropriate, and one is faced with the usual problems confronted by cartographers. It would be outside the scope of this document to discuss the merits of each projection. For detailed information about the different projection systems, the user should refer to the latest WCS document. Let us just mention that equal-area projections (those that conserve relative areas) are often to be preferred for mapping large sky surveys, because they naturally conserve surface-brightness and/or allow summing pixel values to measure fluxes. The following are equal-area projections: `ZEA`, `CEA`, `COE`, `BON`, `GLS`, `PAR`, `MOL`, `AIT`, `QSC`. `AIT` (Aitoff) is one of the most popular projections for all-sky maps.

Note that some of the projections (`CYP`, `CEA`, `COD`, `COE`, `COO`, `COP` and `BON`) *require* additional `PV_xx_xx` parameters. These parameters can easily be included in a *xxxx.head* header file with

[2] Available at `http://www.cv.nrao.edu/fits/src/wcs/`

[3] `http://www.cv.nrao.edu/fits/documents/wcs/wcs.html`

the same prefix as the output coadded image (which is *coadd.fits* by default, see the example at the end of this document).

Table 1: Valid PROJECTION_TYPEs in SWarp

Zenithal projections	
AZP	Zenithal perspective
TAN	Distorted tangential
STG	Stereographic
SIN	Slant orthographic
ARC	Zenithal equidistant
ZPN	Zenithal polynomial
ZEA	Zenithal equal-area
AIR	Airy

Cylindrical projections	
CYP	Cylindrical perspective
CEA	Cylindrical equal-area
CAR	Plate carrée
MER	Mercator

Conic projections	
COP	Conic perspective
COE	Conic equal-area
COD	Conic equidistant
COO	Conic orthomorphic

Pseudoconic and polyconic projections	
BON	Bonne's equal-area
PCO	Polyconic

Pseudocylindrical projections	
GLS	Global sinusoidal (Sanson-Flamsteed)
PAR	Parabolic
MOL	Mollweide
AIT	Hammer-Aitoff

Quad-cube projections	
TSC	Tangential spherical cube
CSC	COBE quadrilateralized spherical cube
QSC	Quadrilateralized spherical cube

Centering of the output frame is controlled by the CENTER_TYPE parameter. There are three centering modes:

- ALL: the field is centered in a way that all input images fit into the output frame. This is the default.
- MOST: the field is centered on the zone of maximum overlap between input images.
- MANUAL: manual centering with the CENTER parameter.

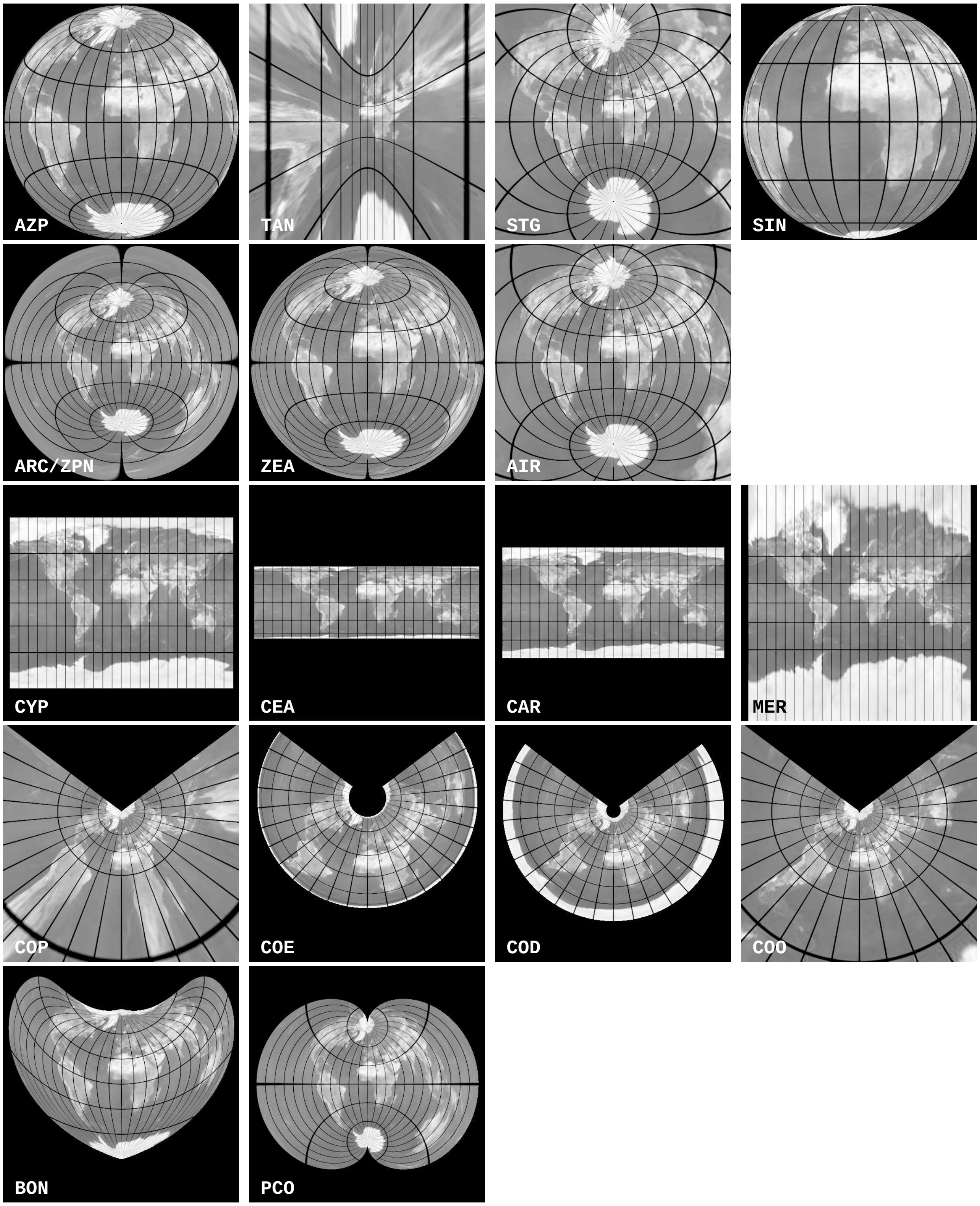


Figure 3: Graphic illustration of projections available in the WCS library (see text).

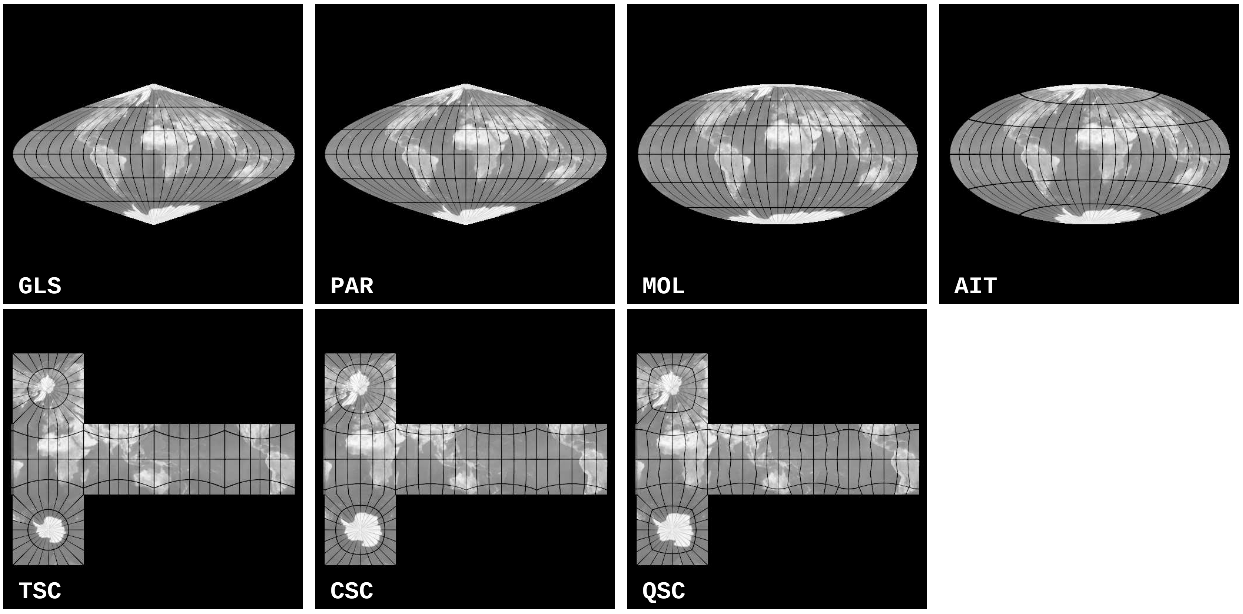


Figure 4: Graphic illustration of projections available in the WCS library (continued from Fig. 3).

A different centering mode can be used in each dimension; for instance, in 2D images with α, δ coordinates, "CENTER_TYPE ALL,MOST" will apply the ALL mode in α and the MOST mode in δ. If a single mode is specified, it is applied to all available dimensions.

The CENTER parameters are active in CENTER_TYPE MANUAL mode only, and must be used to specify the actual center of the output field, in world units. In the case of angular coordinates, both the floating point (in degrees) and sexagedecimal formats are accepted: right ascension/longitude may be written as *hh:mm:ss.ss*, and declination/latitude as $\pm$*dd:mm:ss.ss*.

The pixel "scale" (which is the step between pixels at the center of the output frame) can be computed automatically in each dimension by SWarp. There are five modes specified by the PIXELSCALE_TYPE configuration parameter:

- MEDIAN (the default): the median value of all pixel scales at the center of input frames is taken as the output pixel scale.
- MIN: the smallest of all pixel scales at the center of input frames is taken.
- MAX: the largest of all pixel scales at the center of input frames is taken.
- MANUAL: manual scaling with the PIXEL_SCALE configuration parameter.
- FIT: Pixel scales are automatically computed to have the projected data fitting the output frame dimensions specified with the IMAGE_SIZE configuration parameter.

When right ascension/longitude and declination/latitude are both present, pixel scales computed by SWarp are made equal in both dimensions to avoid anamorphosis.

The PIXEL_SCALE parameters are active in PIXELSCALE_TYPE MANUAL mode only, and must be used to specify the actual pixel step for each dimension, in world units. Note that in the case of angular coordinates, PIXEL_SCALE values are read in arcseconds, not degrees.

The dimensions of the output frame (in number of pixels per axis) are set using the IMAGE_SIZE configuration parameter. A value of 0 for any axis results in an automatic dimensioning of this axis; but obviously this is not possible in PIXELSCALE_TYPE FIT mode.

Note that the current automatic centering and scaling routines can get confused rather easily with some very wide field projections. In particular, it is recommended to turn off automatic settings when making all-sky projections.

5.4 Resampling

The action of projecting a grid of pixels on another grid is called resampling. Ideal image resampling involves both filtering and interpolation between pixels. In SWarp, filtering is "naturally" implemented by oversampling the destination grid. Because SWarp uses reverse-mapping, interpolation is made on the input images.

5.4.1 Image data

At each position $\mathbf{x}$, the dot-product between a local kernel $\mathbf{k}(\mathbf{x})$ and neighbouring pixel values $\mathbf{f}$ yields a local, interpolated value:

$$\tilde{f}(\mathbf{x}) = \mathbf{k}(\mathbf{x}).\mathbf{f} \tag{1}$$

The kernel is derived locally from an *interpolation function* h:

$$k_i(\mathbf{x}) = h(\mathbf{x} - \mathbf{x}_i) \tag{2}$$

The RESAMPLING_TYPE configuration parameter allows the user to choose among several symmetric, compact interpolation functions:

- NEAREST: a "square box" response function, with width 1 pixel. Applying this function produces *nearest-neighbour interpolation* (also known as *sample-and-hold*). The kernel extends over a single input pixel.
- BILINEAR: a pyramidal response function with Full-Width at Half Maximum 1 pixel. This results in a *bilinear interpolation*. The kernel extends over $2^{n_{\rm dim}}$ pixels.
- LANCZOS2: a $\prod_d \mathrm{sinc}(\pi x_d)\mathrm{sinc}(\frac{\pi}{2}x_d)$ response function with $(-2 < x_d \leq 2)$ (Lanczos2 window). The kernel extends over $4^{n_{\rm dim}}$ pixels.
- LANCZOS3: a $\prod_d \mathrm{sinc}(\pi x_d)\mathrm{sinc}(\frac{\pi}{3}x_d)$ response function with $(-3 < x_d \leq 3)$ (Lanczos3 window). The kernel extends over $6^{n_{\rm dim}}$ pixels.
- LANCZOS4: a $\prod_d \mathrm{sinc}(\pi x_d)\mathrm{sinc}(\frac{\pi}{4}x_d)$ response function with $(-4 < x_d \leq 4)$ (Lanczos4 window). The kernel extends over $8^{n_{\rm dim}}$ pixels.

As demonstrated in Fig. 5, the Lanczos4 interpolation function provides the best resampling for correctly sampled data. In theory one could use an even larger kernel to get a closer-to-perfect resampling. However in practice large kernels with a sharply-limited bandpass carry more problems than advantages. Artifacts, image borders or undersampled data generate extended ripples (*Gibbs' phenomenon*). These ripples are obvious on the saturation trail and the cosmic ray impact of the Lanczos interpolations in Fig. 5. In addition, the computational cost becomes prohibitive with multi-dimensional data.

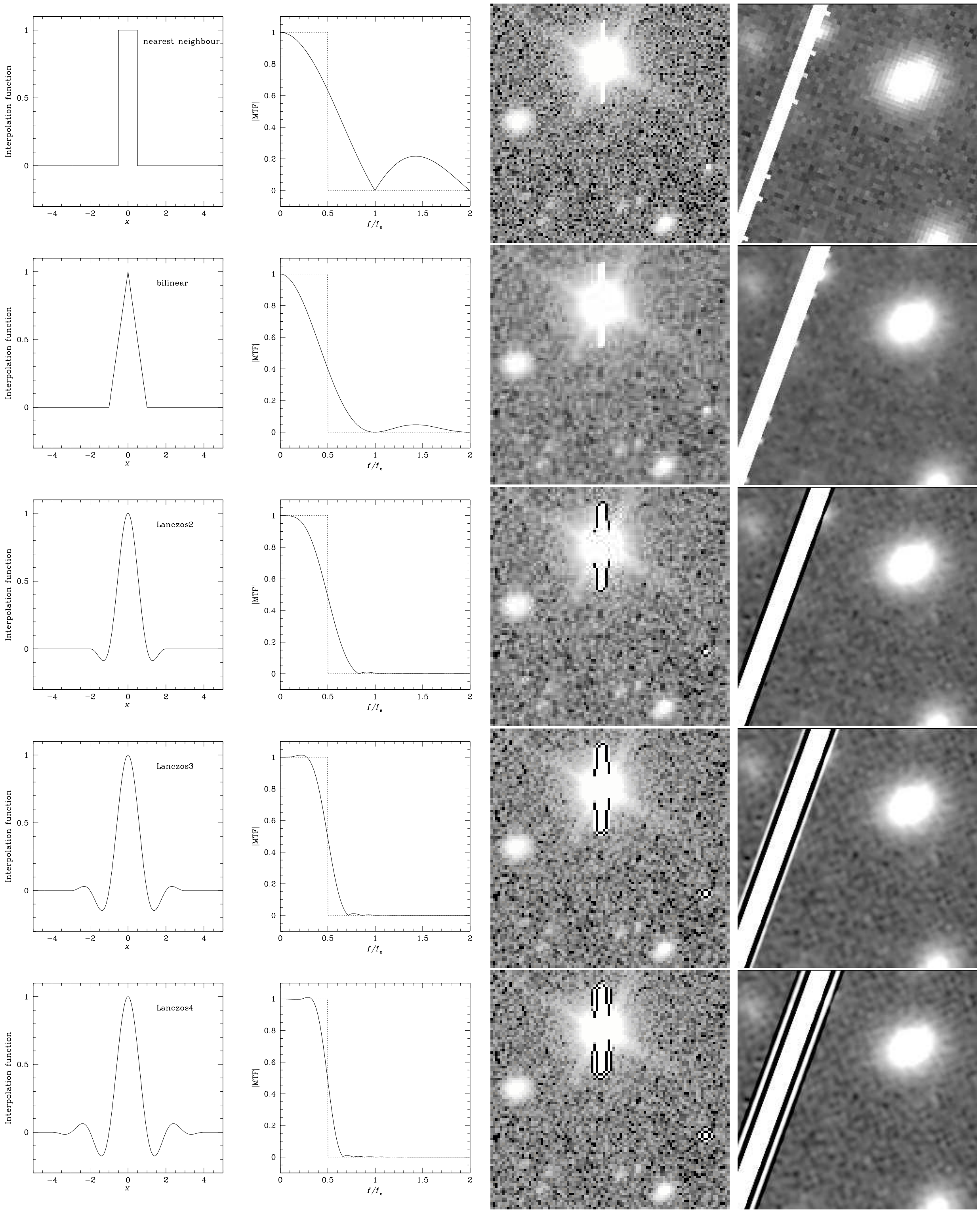


Figure 5: Comparison between resampling methods. *From top to bottom*: nearest-neighbour, bilinear, Lanczos2, Lanczos3, Lanczos4. *From left to right*: Interpolation function profile, Modulation Transfer Function, result from shifting an image by half a pixel in both direction, and result for a 5× zoom + rotation by 20 degrees.

Nearest-neighbour interpolation provides a good conservation of the noise spectrum at scales close to unity; unfortunately, it generates a terrible aliasing when zooming in, and can distort a lot object shapes at places. Its usage should therefore be restricted to images such as flag- or weight-maps.

Bilinear interpolation is fast and doesn't generate negative artifacts. However, it creates a lot of smoothing by correlating the values of neighbour pixels. On images with white noise, this may lead to obvious "moiré" effects (Fig. 6). Nevertheless, bilinear interpolation can be useful for processing undersampled data.

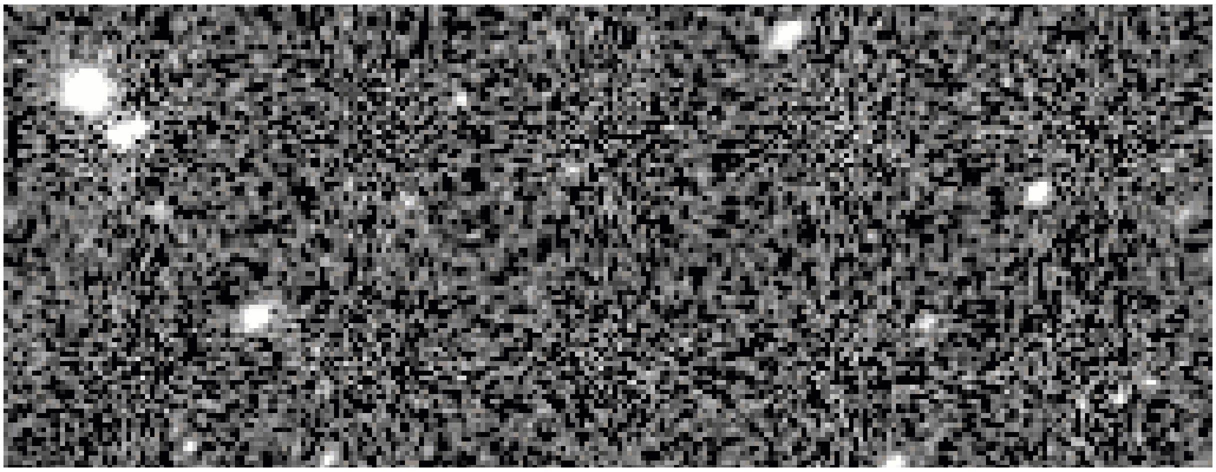


Figure 6: Example of moiré pattern on the background noise, generated by bilinearly resampling an image containing white noise at a pixel scale slightly different from 1.

In general, Lanczos3 resampling represents the best compromise. As can be seen in Fig. 7, it brings a substantial benefit over bilinear interpolation in preserving the signal, while creating relatively modest artifacts around image discontinuities.

5.4.2 Oversampling

Unfortunately the procedure described above generates aliasing when "zooming out" sufficiently an image by resampling it at a lower resolution. Moreover, the resampled background white noise stays constant instead of being proportional to the zooming factor (see Fig. 9). This is because the algorithm essentially *decimates* the data instead of binning them within the output pixel footprints: a similar effect applies in the panning windows of astronomical visualization tools, for instance. This problem can be approximately solved by dilating appropriately the interpolation kernel, or by pre-filtering the input image (like textures in 3D hardware). A more exact and more efficient solution is to *oversample* the output pixel grid (Fig. 9), in order to obtain a density of samples per unit area (or hypervolume) at least equal to that of the input image.

Oversampling is controlled by the `OVERSAMPLING` configuration parameter. If `OVERSAMPLING` is set to 1, no oversampling is applied. An `OVERSAMPLING` of 2 oversamples the data by $2^{n_{\rm dim}}$ per pixel, and so on. Oversampling can be different in each dimension: `OVERSAMPLING` 2,3 will oversample each pixel in a 2×3 grid for instance. An `OVERSAMPLING` of 0 (the default) lets SWarp select automatically the most appropriate oversampling factor in each dimension, by comparing pixel scales at the reference point. Although it works fairly well in many cases, situations where the pixel scale varies a lot over the image — like in all-sky projections — are not yet properly

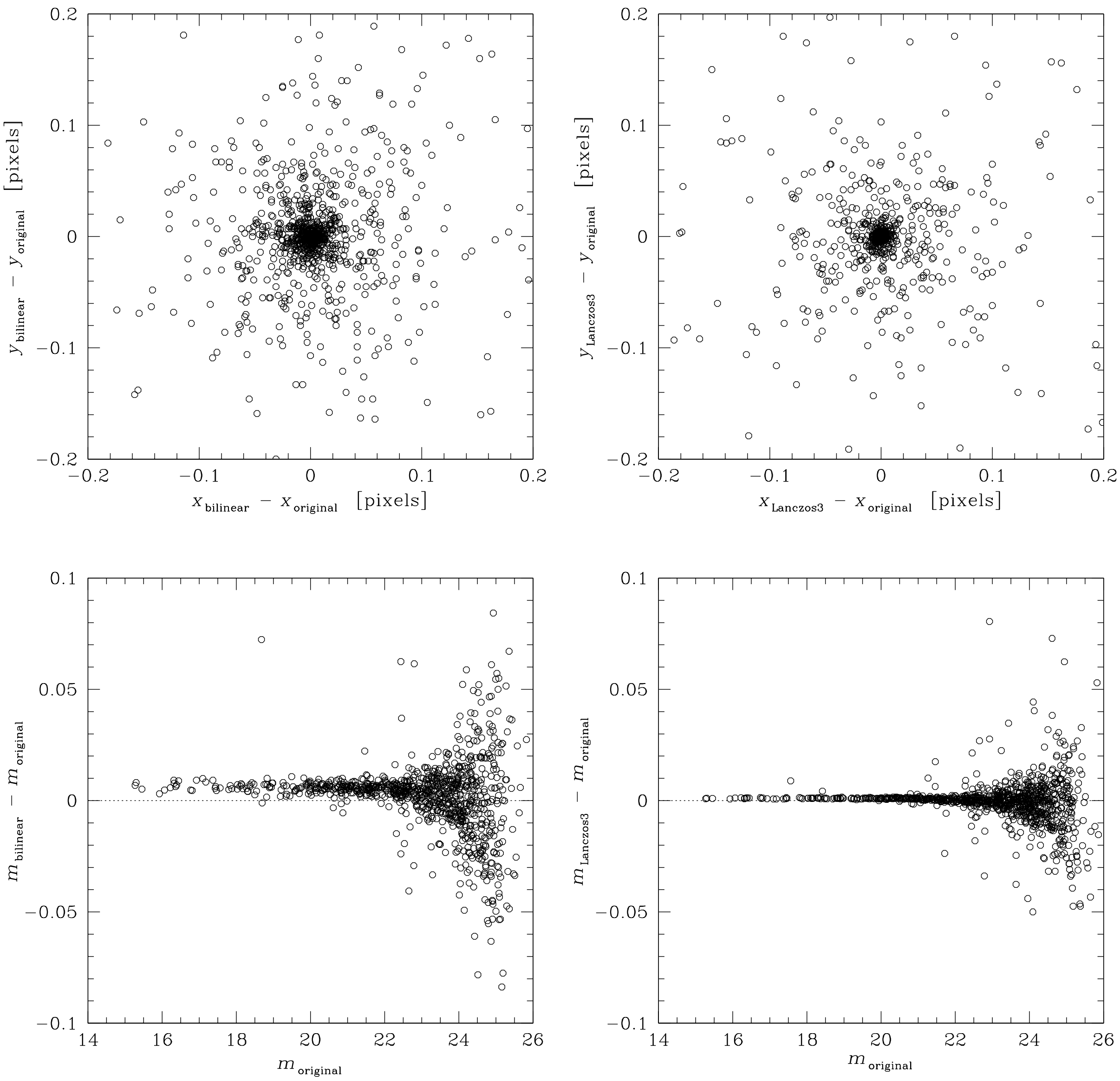


Figure 7: Effects of the resampling on position (*top*) and flux (*bottom*) measurements. *Left*: bilinear interpolation; *right*: Lanczos3 interpolation. In both cases, a simulated deep sky image with 0.7" seeing, containing stars and white background noise, was rotated by 20 degrees and then rotated back to match the original image. Fluxes were measured in a fixed 2" aperture. The dispersions seen here reflect the differences between *measurements* on the original and resampled images. These dispersions are much smaller than what one would observe by comparing the measurements on the resampled images with the theoretical (noise-free) positions or fluxes of the simulation. Note however the significant magnitude offset and flux dispersion in the bilinear case, consequences of the stronger smoothing induced by bilinear interpolation.

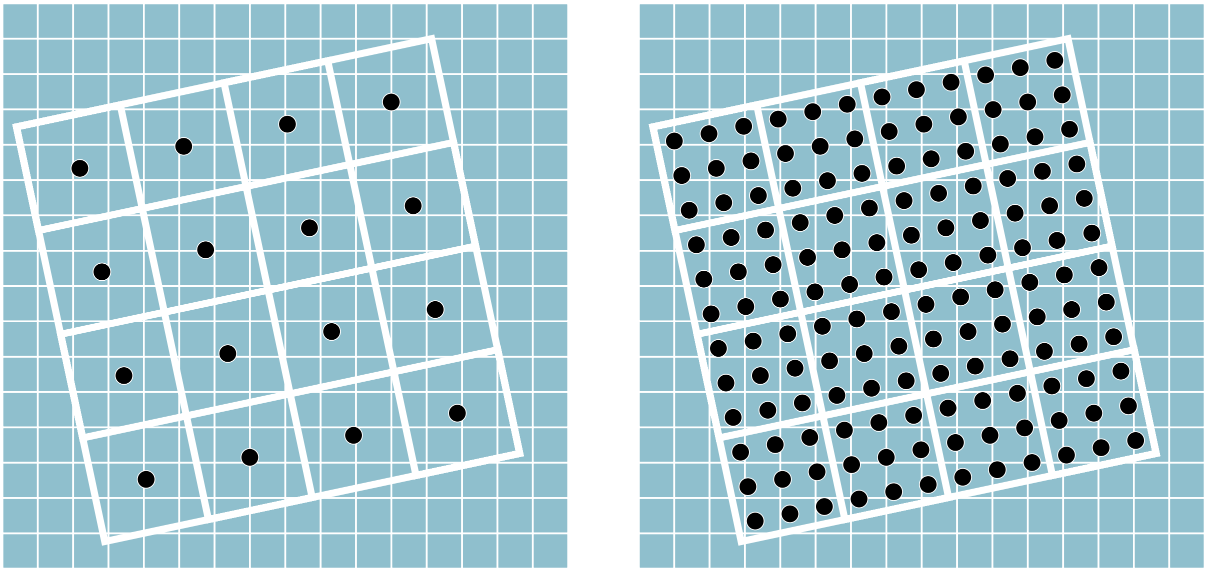


Figure 8: Oversampling in SWarp. The input grid is shown as small grey squares, whereas the output grid (resampled image) is represented by the large tilted ones. *Left*: Without oversampling, only one interpolation (dark spot) of the input image is done at the center of each output pixel. *Right*: with oversampling, several interpolated samples are obtained on a regular subgrid, and then binned in each output pixel. Here a 3×3 oversampling is sufficient.

handled, and manual setting should then be prefered.

Note that oversampling considerably slows down the processing; `OVERSAMPLING` values should therefore be selected with some caution!

5.4.3 Noise stability issues

So far we have ignored the influence of noise variations in the resampling process. In theory, the interpolation schemes described above apply only if the noise is stationary (in the wide sense) over the extent of the interpolation kernel. Artifacts aside, this can be considered as true for the background noise, since the weight-maps are reasonably stable at the interpolation scale. However, the photon noise associated with the sources themselves may vary strongly over the scale of the PSF FWHM. As most astronomical images are barely oversampled, the hypothesis of noise stationarity breaks down on bright point sources for which intrinsic photon noises dominates[4] In the most severe cases, resampled noise peaks may generate distorsions in the resampled profiles.

Low-background-noise simulations were conducted in order to evaluate the amplitude of these distorsions on correctly-sampled data (PSF FWHM = 3 pixels). The effect is small, although not totally negligible on sources with intermediate intensity. On profile-fitting measurements for instance, photometry can be affected at the level of a few millimag *rms*. The degradation of astrometric precision was found not to exceed a few millipixels *rms*. On typical background-noise limited images the effects are even smaller.

[4]It is possible to stabilize the noise variance using a non-linear dynamic scale transform (Anscombe 1948, see also Stark *et al.* 1998). The transformed signal is still bandpass-limited but unfortunately, resampling and transforming it back biases significantly the data.

Figure 9: The effect of oversampling in SWarp. A deep, real image with a 0.2" pixel scale and 0.8" seeing FWHM is resampled at 1" resolution. *Left*: no oversampling. *Right*: with 5×5 oversampling. Note the lower noise level and higher depth in the right image.

5.4.4 Weight-maps

The processing of the weight-maps (see §5.8) follows that of the data images, except that one is dealing now with variances instead of fluxes. The resampled weight at position $\mathbf{x}$ may be written as

$$\tilde{w}(\mathbf{x}) = \frac{1}{\sum_i \frac{k_i^2(\mathbf{x})}{w_i}} \tag{3}$$

Therefore when an input weight within the range of the interpolation function is zero, the interpolated weight is also zero. The general consequence is that the borders of interpolated images are trimmed by half the range of the interpolation function. Similarly, small "holes" in a provided weight-map are dilated by the interpolation function footprint. For example, once interpolated with a Lanczos4 kernel, a single, isolated zero-weight pixel will yield a clump of about 64 pixels in the resampled image! This is another illustration of the disadvantages in using large interpolation kernels.

5.5 Background subtraction

The flux at each pixel is a function of the sum of a "background" signal and light coming from the objects of interest. At most wavelengths, the strongest contribution to the background is of instrumental/atmospheric/ecliptic origin, and is therefore prone to changes between exposures. If not subtracted, the compositing of all these exposures will often produce an ugly patchwork created by all the different individual backgrounds. A solution to this problem is to apply background-subtraction prior to resampling and co-adding the data. Large-scale gradients of instrumental origin are commonly found on astronomical images, hence subtracting a constant from each frame will generally yield poor results, as shown in Fig. 10). It is therefore necessary to subtract a smooth *background map* which contains the low-spatial-frequency noise components of the data, including any offset. Subtracting a background-map may alter or destroy the signal of scientific interest; thus some caution is needed in choosing parameters for this procedure.

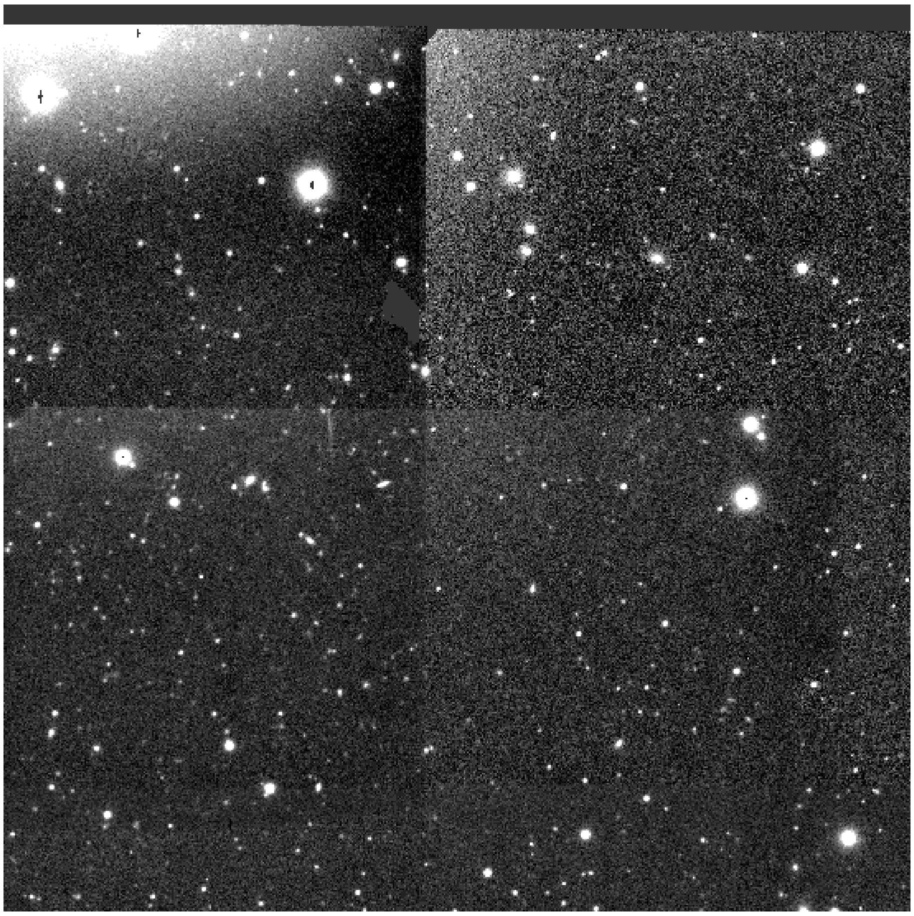


Figure 10: Example of residual gradients in a co-addition after a constant has been subtracted from input images.

Background subtraction is activated by setting the `SUBTRACT_BACK` configuration parameter to `Y` (which is the default). Setting it to `N` will disable subtraction, but background *estimation* will still take place (it is needed by other SWarp tasks), and the processing time will stay approximately the same.

Background estimation uses SExtractor's algorithm and is controllable with the same keywords[5]. The following is largely copied from SExtractor documentation (Bertin 1999).

To construct the background map, SWarp makes a first pass through the pixel data, computing an estimator for the local background in each mesh of a grid that covers the whole frame. The background estimator is a combination of $\kappa.\sigma$ clipping and mode estimation, similar to the one employed in Stetson's DAOPHOT program (see e.g. Da Costa 1992). Briefly, the local background histogram is clipped iteratively until convergence at $\pm 3\sigma$ around its median; if σ is changed by less than 20% during that process, we consider that the field is not crowded and we simply take the mean of the clipped histogram as a value for the background; otherwise we estimate the mode with:

$$\mathrm{Mode} = 2.5 \times \mathrm{Median} - 1.5 \times \mathrm{Mean} \tag{4}$$

This expression is different from the usual approximation

$$\mathrm{Mode} = 3 \times \mathrm{Median} - 2 \times \mathrm{Mean} \tag{5}$$

[5]In SWarp All background configuration keywords accept a list of values, one value for each input frame.

(e.g. Kendall and Stuart 1977), but was found to be more accurate with our clipped distributions, from the simulations we made. Fig. 11 shows that the expression of the mode above is considerably less affected[6] by crowding than a simple clipped mean — like the one used in FOCAS (Jarvis and Tyson 1981) or by Infante (1987) — but is ≈ 30% noisier. For this reason we revert to the mean in non-crowded fields.

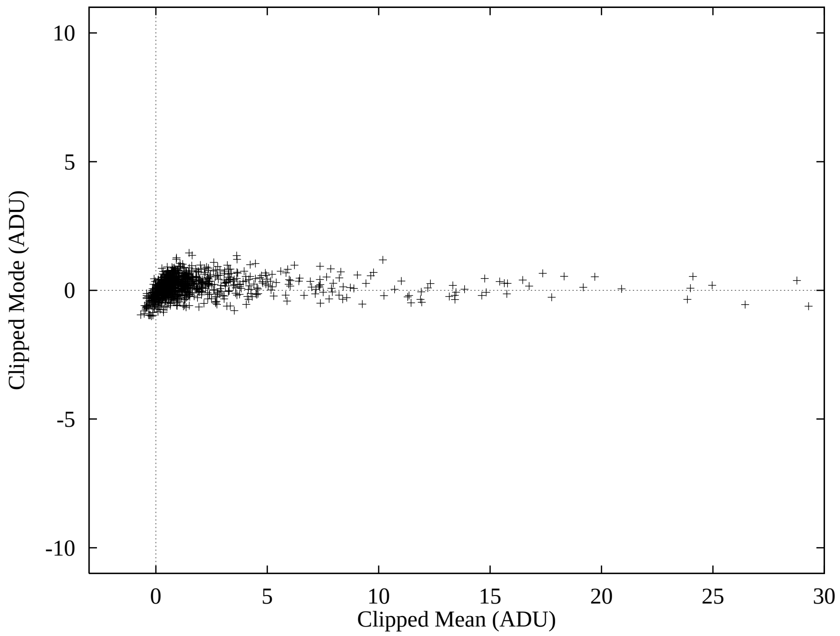


Figure 11: Simulations of 32×32 pixels background meshes polluted by random Gaussian profiles. The true background lies at 0 ADU. While being slightly noisier, the clipped "Mode" gives a more robust estimate than a clipped Mean in crowded regions.

The choice of the mesh size (in pixels), `BACK_SIZE`, is critical. If it is too small, the background estimation is affected by the presence of objects and random noise. But more important is the fact that part of the flux of extended objects can be absorbed in the background map. The effect may be almost unnoticeable on individual input images (where the signal-to-noise ratio is low), and have measurable photometric consequences on the deep, coadded image. It is therefore advised to use large `BACK_SIZE`s in SWarp. Of course if the mesh size is too large, it will not be able to reproduce all the variations of the background; a good compromise has to be found by the user. Typically, for reasonably sampled images, a size of 128 (the default) to 512 pixels should work well.

The final background map is a (natural) bicubic-spline interpolation between the meshes of the grid. Before interpolating, a median filter can be applied to suppress possible local overestimations due to bright stars or artifacts. `BACK_FILTERSIZE` sets the size (in background meshes) of the median filter. "1" means no filtering applied to the background grid. Usually a size of 3 meshes (the default) is sufficient, but it may be necessary to use larger dimensions, especially to compensate, in part, for small background mesh sizes, or in the case of large artifacts in the images. Median filtering also helps reducing possible ringing effects of the bicubic-spline around bright features. In some specific cases it might be desirable to median-filter only background meshes whose original values exceed some threshold above the filtered-value. This differential threshold is set by the `BACK_FILTERTHRESH` parameter, in ADUs. The default is 0.

[6] Obviously in some very unfavorable cases (like small meshes falling on bright stars), it leads to totally inaccurate results.

By default the computed background-map is automatically subtracted from the input image. But there are some situations where it is more appropriate to subtract a *constant* from the image (e.g., images where the background noise distribution is strongly skewed). The `BACK_TYPE` configuration parameter (set by default to "AUTO") can be switched to `MANUAL` to allow for the value specified by the `BACK_DEFAULT` parameter to be subtracted from the input image. The default value is 0.

As said before, the background estimation procedure is used not only for background subtraction, but also for other tasks in SWarp, such as weight calibration. Thus even if `SUBTRACT_BACK` is set to `N` or `BACK_TYPE` is in `MANUAL` mode, "reasonable" values for other background parameters must be given to ensure proper working of the software.

Note that the present version of background subtraction doesn't work on non-2D images.

5.6 Scaling the flux

How are fluxes modified by image warping? Let us assume F is the integrated flux (in units of e^- for simplification) of a source of finite extent $\mathcal{S}$ that would be recorded on a perfect detector array. In the continuous limit, we define

$$F \equiv \int_{\mathcal{S}} f(\boldsymbol{x})\, d^2x, \tag{6}$$

where $f(\boldsymbol{x})$ is the pixel value at physical position $\boldsymbol{x}$ on the detector. In real life, pixel values are affected by a variable efficiency q, yielding a measured "raw" flux

$$F_r = \int_{\mathcal{S}} q(\boldsymbol{x}) f(\boldsymbol{x})\, d^2x. \tag{7}$$

Digital images are generally divided by a "flat-field" and even a "super-flat" prior to SWarping. The assumption behind flat-fielding is that the light received from the sky or the dome, and recorded to form the flat-field, has uniform radiance (i.e. constant flux per solid angle). The flux measured on the flat-fielded image is therefore

$$F_f = \int_{\mathcal{S}} \frac{q(\boldsymbol{x}) f(\boldsymbol{x})}{f_0 q(\boldsymbol{x}) \delta\Omega(\boldsymbol{x})}\, d^2x = \int_{\mathcal{S}} \frac{f(\boldsymbol{x})}{f_0 \delta\Omega(\boldsymbol{x})}\, d^2x, \tag{8}$$

where $\delta\Omega$ is the local sky area sustained by a pixel (area $\equiv 1$ in pixel units), and f_0 the scaling factor of the flat-field, which we will set to 1 for the sake of simplicity. As can be seen, flat-fielding does not make the image "flat" in terms of sensitivity; it introduces a dependency with astrometrical distorsion. With most cameras the effect is generally small (≤ 1 millimag). The resampling operations described in §5.4 are designed to conserve surface brightness per pixel, hence the flux recorded on the warped image, with new physical coordinates $\boldsymbol{x}'$ is now

$$F_{fw} = \int_{\mathcal{S}} \frac{f(\boldsymbol{x})}{\delta\Omega(\boldsymbol{x})}\, d^2x' = \int_{\mathcal{S}} \frac{f(\boldsymbol{x})}{\delta\Omega(\boldsymbol{x})} \left|\frac{\partial x'_i}{\partial x_j}\right| d^2x = \int_{\mathcal{S}} f(\boldsymbol{x}) \left|\frac{\partial x'_i}{\partial \xi_j}\right| d^2x. \tag{9}$$

$\boldsymbol{\xi}$ represents the local (angular) sky coordinate vector. We have made use of the fact that if pixel size is small compared to the rate of change of plate scale (which is almost always true), $\delta\Omega(\boldsymbol{x})$ is equal to the Jacobian of the de-projection $\left|\frac{\partial \xi_i}{\partial x_j}\right|$. Now if an equal-area projection is selected for the output image, $\left|\frac{\partial x'_i}{\partial \xi_j}\right|$ is constant and we have the nice relation $F_{fw} \propto F$. This means that swarping properly flatfielded data using an equal-area output projection produces an image with a perfectly flat response to the incoming flux[7]. This is why equal-area projections should be prefered to other projections whenever possible.

Immediately following resampling, the intensity of each image is scaled according to the configuration parameters. The flux-scaling parameter p_i is the product of two factors: a "photometric" factor, and an "astrometric" one. Currently, the photometric part must be specified by the user (for instance in a pipeline it is generally produced by the photometric calibration process). The photometric factor can be set directly using the configuration parameter `FSCALE_DEFAULT`: there must be one value per input image, separated by a coma. It can also be read from the FITS header. The `FSCALE_KEYWORD` configuration parameter tells SWarp what FITS keyword to look for in each input image. The default is "FLXSCALE". If the `FSCALE_KEYWORD` is not found in the image header, then the `FSCALE_DEFAULT` value is taken instead. `FSCALE_DEFAULT` is defaulted to 1 for all images.

The astrometric part of the flux-scaling factor corrects for the difference in pixel size between the input and output images. The `FSCALASTRO_TYPE` configuration parameter controls the behaviour of this "astrometric" flux-scaling. In the current version, the default behaviour (`FSCALASTRO_TYPE FIXED`) is to apply a *constant* correction factor to account for possible mismatches in pixel size. Hence flux is conserved only with equal-area projections. "Astrometric" flux-scaling can also be deactivated by using the `FSCALASTRO_TYPE NONE` option. Future versions of SWarp may include variable pixel-scale correction for non-equal-area projections.

5.7 Combining resampled images

This is the last part of the processing. Now at each pixel position of the output image, SWarp has to combine data values coming from all the resampled image, each one coming with a rough estimate of its variance (from the resampled weight-map). Many combinations are therefore possible. The user can choose between the following options, as arguments to the `COMBINE_TYPE` configuration parameter:

- `AVERAGE`: The output is simply an unweighted average of all pixel values with non-zero weights:
$$F = \frac{\sum_i p_i f_i}{n_{\neq 0}}, \tag{10}$$
and the composite weight is
$$W = \frac{n_{\neq 0}^2}{\sum_i \frac{1}{w_i}}, \tag{11}$$
where the w_i are proportional to the inverse of the scaled variance: $\frac{p_i^2}{\sigma^2}$. Needless to say that this combination is not optimum in terms of S/N, unless all input images have identical Gaussian noise.

- `CHI2`: The output is the square-root of the reduced χ^2 of all pixel values with non-zero weights:
$$F = \frac{\sqrt{\sum_i w_i f_i^2}}{n_{\neq 0}}. \tag{12}$$
By construction, the composite weight (the absolute one) is
$$W \equiv 1. \tag{13}$$

[7]Warning: this flux correction can be extremely inaccurate — 10% error or even more! — for sources that are undersampled on the output image.

The result of the combination is a so-called "χ^2 image". Although it does not respond linearly to the input signals, it can be used for detecting sources. As shown by Szalay et al. (1999), the "χ^2 image" is indeed the optimum combination to achieve panchromatic detection on a set of images taken at different wavelengths, provided the data sets are background-noise limited and that the noise is uncorrelated between frames. This assumes further that the *Point Spread Function* (PSF) has been homogenized in all channels. χ^2 images are most often used in deep panchromatic surveys requiring photometric redshift analyses. The double-image mode of SExtractor allows one to detect on the χ^2 image while making the photometric measurements on each of the single-band images.

- `MEDIAN`: The output is the median of all scaled pixel values with non-zero weights:

$$F = \mathrm{median}(f_i). \tag{14}$$

Assuming Gaussian noise distribution, we obtain the following approximation to the composite weight (see e.g. Kendall & Stuart 1977):

$$W = \begin{cases} \frac{2}{\pi}\left(\frac{\sum_i \sqrt{w_i}}{n_{\neq 0}}\right)^2 (n_{\neq 0} + \frac{\pi}{2} - 1) & \text{if } n_{\neq 0} \text{ is even,} \\ \frac{2}{\pi}\left(\frac{\sum_i \sqrt{w_i}}{n_{\neq 0}}\right)^2 (n_{\neq 0} + \pi - 2) & \text{otherwise.} \end{cases} \tag{15}$$

This approximation can become inaccurate if w_j varies by large proportions (a factor of 3 or more) from frame to frame. The median is convenient for combining data polluted by unidentified glitches or noise spikes. It generally provides "safe" (robust) co-additions even with strongly non-Gaussian noise distributions. However it is suboptimal for Gaussian noise: the resulting variance is increased by $\approx 60\%$ compared to what is obtained with an average. As with all non-linear combinations, one should check that input images have approximately the same Point Spread Function if point-sources are to be co-added.

- `MIN`: The output is the minimum of all pixel values with non-zero weights:

$$F = \min(p_i f_i). \tag{16}$$

The variance of F is too "noise-distribution dependant" to allow some estimation, hence we set

$$W = \begin{cases} 1 \text{ if } n_{\neq 0} \neq 0, \\ 0 \text{ otherwise.} \end{cases} \tag{17}$$

- `MAX`: The output is the maximum of all pixel values with non-zero weights:

$$F = \min(p_i f_i). \tag{18}$$

The variance of F is too "noise-distribution dependant" to allow some estimation, hence we set

$$W = \begin{cases} 1 \text{ if } n_{\neq 0} \neq 0, \\ 0 \text{ otherwise.} \end{cases} \tag{19}$$

Maxima and minima can be useful for identifying defects or rare events in a set of data.

- `WEIGHTED`: The output is a weighted average of input values:

$$F = \frac{\sum_i w_i p_i f_i}{\sum_i w_i}. \tag{20}$$

The output weight is just the sum of input weights:

$$W = \sum_i w_i. \tag{21}$$

This combination should be the most appropriate for detecting and measuring faint sources on properly weighted, homogeneous images. Because it is a linear processing, new data can be added later if needed.

5.8 Weighted coaddition

Weight-maps provide a convenient way to store the standard flux error assigned to each pixel. For each input image $1 \leq i \leq N$ entering co-addition, one can define the following parameters in an arbitrarily small "pixel" j:

- The local, uncalibrated flux $f_{ij} = \overline{f_{ij}} + \Delta f_{ij}$, where $\overline{f_{ij}}$ is the flux contributed by the sky background, and Δf_{ij} that contributed by celestial sources,
- the local, uncalibrated variance of the flux $\sigma^2_{ij} = \overline{\sigma^2_{ij}} + \Delta\sigma^2_{ij}$, where $\overline{\sigma^2_{ij}}$ is the flux variance contributed by the background noise, and $\Delta\sigma^2_{ij}$ that contributed by the photon statistics of celestial sources,
- the local,normalized weight w_{ij},
- the electronic gain of the CCD g_i, in e$^-$/ADU (defined at $w_{ij} = 1$),
- the relative flux scaling factor p_i deduced from the photometric solution to calibrate the images:
$$p_i \Delta f_{ij} = p_l \Delta f_{lj} \quad \forall i, j, l, \tag{22}$$
- and the relative weight scaling factor k_i deduced from the comparison of background noises, so that input images are weighted with $k_i w_i$.

Δf_i, $\Delta\sigma^2_i$, w_i and g_i are related through:

$$g_i w_{ij} = \frac{\Delta f_{ij}}{\Delta\sigma^2_{ij}}. \tag{23}$$

Now, to optimally co-add calibrated images, one could weight them using

$$k_i w_{ij} = \frac{1}{p_i^2 \sigma^2_{ij}}. \tag{24}$$

However, such weight maps exhibit strong variations on small scales, in the presence of celestial objects (σ^2_{ij} increases a lot on bright pixels). The modulating effect of weighting, combined with variations of the PSF and sampling errors would lead to significant distorsions of stellar profiles. It is therefore more appropriate to weight pixels according to the intensity of the local background noise, which is far smoother:

$$k_i w_{ij} = \frac{1}{p_i^2 \overline{\sigma^2_{ij}}}. \tag{25}$$

This has also the advantage that one can use normalized flat-fields as w_{ij}'s, without prior knowledge of the CCD gain in the case where instrumental noise is negligeable. For faint objects, this weighting scheme is as efficient as that of (24), and is only suboptimum for the objects with very high surface-brightness, when the k_i's vary a lot from exposure to exposure. But as these objects are easily detected on individual exposures, the most accurate photometry is still possible by combining the N measurements.

In practice, SWarp can read several types of weight-maps, although they are all internally converted to variance for processing. The input weight-map format must be specified with the `WEIGHT_TYPE` keyword. `WEIGHT_TYPE NONE` is for no input weight-map (the default), `MAP_RMS` for indicating that the data contain the absolute standard deviation of pixel values, `MAP_VARIANCE`, for weight-map data stored as relative variances, and `MAP_WEIGHT` for relative weights. Relative variances or weights are scaled internally using local variance measurements made directly on the input images.

When producing composite fields larger than the input images, the latter must be background-subtracted prior to coaddition, to avoid generating discontinuites in the output image. Thus one can write the output coadded flux as

$$\Delta f_j = \frac{\sum_i k_i w_{ij} p_i \Delta f_{ij}}{\sum_i k_i w_{ij}} = p_l \Delta f_{lj} \quad \forall l, \tag{26}$$

and the resulting variance as

$$\Delta\sigma_j^2 = \frac{\sum_i k_i^2 w_{ij}^2 p_i^2 \Delta\sigma_{ij}^2}{\left(\sum_i k_i w_{ij}\right)^2}. \tag{27}$$

Using (23), an equivalent "local gain" G_j in the coadded image can be computed:

$$G_j w_j = \frac{\Delta f_j}{\Delta\sigma_j^2} = \frac{\sum_i k_i w_{ij} p_i \Delta f_{ij}}{\sum_i k_i^2 w_{ij}^2 p_i^2 \Delta\sigma_{ij}^2} . \sum_i k_i w_{ij}, \tag{28}$$

where w_j is the composite weight-map of the coadded image. From our definition of weights, w_j is inversely proportional to $\overline{\sigma_j^2}$ and must be 1 if all input weights are at 1; hence it is easy to show that

$$w_j = \frac{\sum_i k_i w_{ij}}{\sum_i k_i}. \tag{29}$$

Substituting (29) in (28), and using (22) and (23), one gets

$$G_j = \frac{\sum_i k_i w_{ij}}{\sum_i k_i^2 w_{ij} p_i g_i^{-1}} . \sum_i k_i. \tag{30}$$

Unfortunately, as can be seen, this "coadded gain" will vary with position in the general case. Nevertheless, some approximations can be made to simplify this expression. First of all, in most cases, g_i will be almost constant from one input image to another. Second, if exposures are taken under photometric conditions with constant sky brightness and negligible instrumental noise, one should have $k_i \propto p_i^{-1}$, removing the dependence with input weights, and therefore position in the coadded image. In that very case, the resulting gain is simply

$$G \approx g \sum_i p_i^{-1}. \tag{31}$$

Sadly, in many bands, the presence of clouds does not decrease the sky brightness as much as source brightness, and doesn't act at all like a decrease in global sensitivity or exposure time. Coadded regions of a survey that are taken under non-photometric conditions experience fluctuating "gains". But this effect is generally small: taking the quite unfavourable case of $k_i \propto p_i^{-2}$ (constant background noise) with p_i's varying by a factor of 4 between two exposures, and weights varying from 0.5 to 1, makes G_j to vary by 20% at most. This should not cause significant difficulties in any profile fitting routine. Therefore (31) still remains a very good approximation in the general case, and the "coadded gains" provided by this method are still more stable than what unweighted co-addition offers.

This does not prevent regions with lower coverage (in which N is smaller) having lower gains. To avoid large gain drops in the gaps of CCD mosaic images, it is recommended for the observations ot use large, random dithering patterns consisting of at least 4-5 exposures per covered sky area.

5.9 Examples.

In the following, examples of use of SWarp are given, together with commented configuration files.

5.9.1 Example 1

Let us assume one wants to produce a full-sky Aitoff representation in galactic coordinates of a series of observed fields stored in `*.fits` 2D files(with WCS info), for illustration purposes. The files might be dummy ones, supplemented with hand-made headers, or obtained from virtual telescopes such as *SkyView*[8], or real ones. In all cases an additional full sky map is useful to delimit the full-sky: one may for instance download a 360 degrees Aitoff projection of COBE-DIRBE data from SkyView. The syntax is

% `swarp *.fits`

A possible `default.swarp` configuration file is

```
#---------------------------------- Output -----------------------------------

IMAGEOUT_NAME   coadd.fits
WEIGHTOUT_NAME  coadd.weight.fits

#------------------------------- Input Weights -------------------------------

WEIGHT_TYPE     MAP_WEIGHT              # Not used here
WEIGHT_SUFFIX   .weight.fits
WEIGHT_IMAGE

#------------------------------- Co-addition ---------------------------------

COMBINE_TYPE    AVERAGE                 # This coaddition is for illustration
                                        # only: the weight-map will contain
                                        # a sum of field footprints

#-------------------------------- Astrometry ---------------------------------

CELESTIAL_TYPE  GALACTIC                # Coordinate system forced to galactic
PROJECTION_TYPE AIT                     # Code for Aitoff
CENTER_TYPE     MANUAL                  # Imposed to alpha = delta = 0.0
CENTER          00:00:00.0, +00:00:00.0 #
```

[8] `http://skyview.gsfc.nasa.gov`

```
PIXELSCALE_TYPE MANUAL                  # The full sky area will exceed the
                                        # fraction that contains the fields
PIXEL_SCALE     1800.0                  # in arcsec: Half a degree per pixel
                                        # at image center, on both axes
IMAGE_SIZE      800,400                 # 360/0.5 = 720, 180/0.5 = 360,
                                        # plus a margin

#------------------------------- Resampling ---------------------------------

RESAMPLING_TYPE BILINEAR                # For illustration purposes, no need
                                        # for a sophisticated interpolation
OVERSAMPLING    4                       # A small oversampling only to have
                                        # pretty, antialiased field limits
INTERPOLATE     N
GAIN_KEYWORD    GAIN
GAIN_DEFAULT    0.0

#--------------------------- Background subtraction ---------------------------

SUBTRACT_BACK   N                       # No background subtraction
BACK_TYPE       AUTO
BACK_DEFAULT    0.0
BACK_SIZE       128
BACK_FILTERSIZE 3

#------------------------- Virtual memory management -------------------------

VMEM_DIRECTORY  .
VMEM_MAX        2047
MEM_MAX         128                     # 128 MB should be enough to avoid
                                        # swapping

#------------------------------ Miscellaneous ---------------------------------

DELETE_TMPFILES Y                       # Delete temporary resampled FITS files
VERBOSE_TYPE    NORMAL
```

In this application, coverage maps can be generated using the output weight-map instead of the image itself.

5.9.2 Example 2

In this example, one has a set of CCD images taken with a standard dithering strategy, `input*.fits`, and the related set of weight-maps `input*.w.fits` However the unusual thing is that for some reason the output has to be tilted by 30 degrees with respect to the local north-south axis, and the pixels must have an aspect ratio of 16:9!

First, on starts with a fairly standard configuration file:

```
#---------------------------------- Output -----------------------------------
```

```
IMAGEOUT_NAME   coadd.fits              # Output filename
WEIGHTOUT_NAME  coadd.weight.fits       # Output weight-map filename

#------------------------------- Input Weights --------------------------------

WEIGHT_TYPE     MAP_WEIGHT              #
                                        # (all or for each weight-map)
WEIGHT_SUFFIX   .w.fits            # Suffix to use for weight-maps
WEIGHT_IMAGE                            # Weightmap filename if suffix not used
                                        # (all or for each weight-map)

#------------------------------- Co-addition ----------------------------------

COMBINE_TYPE    WEIGHTED                # weight-maps are provided

#-------------------------------- Astrometry ----------------------------------

CELESTIAL_TYPE  NATIVE                  # Standard stuff
PROJECTION_TYPE TAN                     # A tangent projection will do
CENTER_TYPE     ALL                     # We want all the data to fit in
CENTER          00:00:00.0, +00:00:00.0 # Not used in CENTER_TYPE ALL mode
PIXELSCALE_TYPE MEDIAN                  # Will be overriden by coadd.head
PIXEL_SCALE     0.0                     # Not used in MEDIAN mode
IMAGE_SIZE      0                       # Automatic sizing

#-------------------------------- Resampling ----------------------------------

RESAMPLING_TYPE LANCZOS3                # High quality resampling
OVERSAMPLING    0                       # Auto. oversampling (=1 in that case)
INTERPOLATE     N
GAIN_KEYWORD    GAIN
GAIN_DEFAULT    0.0

#--------------------------- Background subtraction ---------------------------

SUBTRACT_BACK   Y                       # Needed for co-adding dithered fields
BACK_TYPE       AUTO
BACK_DEFAULT    0.0
BACK_SIZE       128
BACK_FILTERSIZE 3

#------------------------- Virtual memory management --------------------------

VMEM_DIRECTORY  .
VMEM_MAX        2047
MEM_MAX         128                     # 128 MB should be enough to avoid
                                        # swapping

#------------------------------ Miscellaneous ---------------------------------
```

```
DELETE_TMPFILES Y                    # Delete temporary resampled FITS files
VERBOSE_TYPE    NORMAL
```

To implement the unusual output features required, one must write a `coadd.head` ASCII file that contains a custom anisotropic scaling matrix. A `coadd.head` for pixels 0.2" large, that tilts the image by 30 degrees and applies a 16:9 anamorphosis to the data writes:

```
CD1_1   =  6.4150E-5
CD1_2   =  2.0833E-5
CD2_1   = -3.7037E-5
CD2_2   =  3.6084E-5
END
```

Acknowledgments Many thanks go to Mark Calabretta for his great astrometric library, Nicolas Devillard for introducing me to memory-mapping techniques, Yannick Mellier and Mario Radovich (from TERAPIX) for extensive testing and suggestions, and Henry Joy McCracken for help with the manual and additional testing.

References

[1] Anscombe F.J., 1948, *Biometrika* **15**, 246

[2] Bertin E., 1999, *SExtractor 2.1, User's manual*, IAP

[3] Da Costa G.S., 1992, in *Astronomical CCD Observing and Reduction Techniques*, ed. Howell S.B. (ASP Conf. Series)

[4] Fruchter A., Hook R.N., 1997, *SPIE* **3164**, 120

[5] Infante L., 1987, *A&A* **183**, 177

[6] Jarvis J.J., Tyson J.A., 1981, *AJ*, **86**, 476

[7] Kendall M., Stuart K., 1977, *The Advanced Theory of Statistics*, **Vol. 1**, (Charles Griffin & Co., London)

[8] Starck J.-L., Murtagh F., Bijaoui A., 1998, *Image Processing and Data Analysis* (Cambridge University Press)

[9] Szalay A.S., Connolly A.J., Szokoly G.P., 1999, *AJ* **117**, 68

[10] Wolberg G., 1992, *Digital Image Warping* (IEEE Computer Society Press)