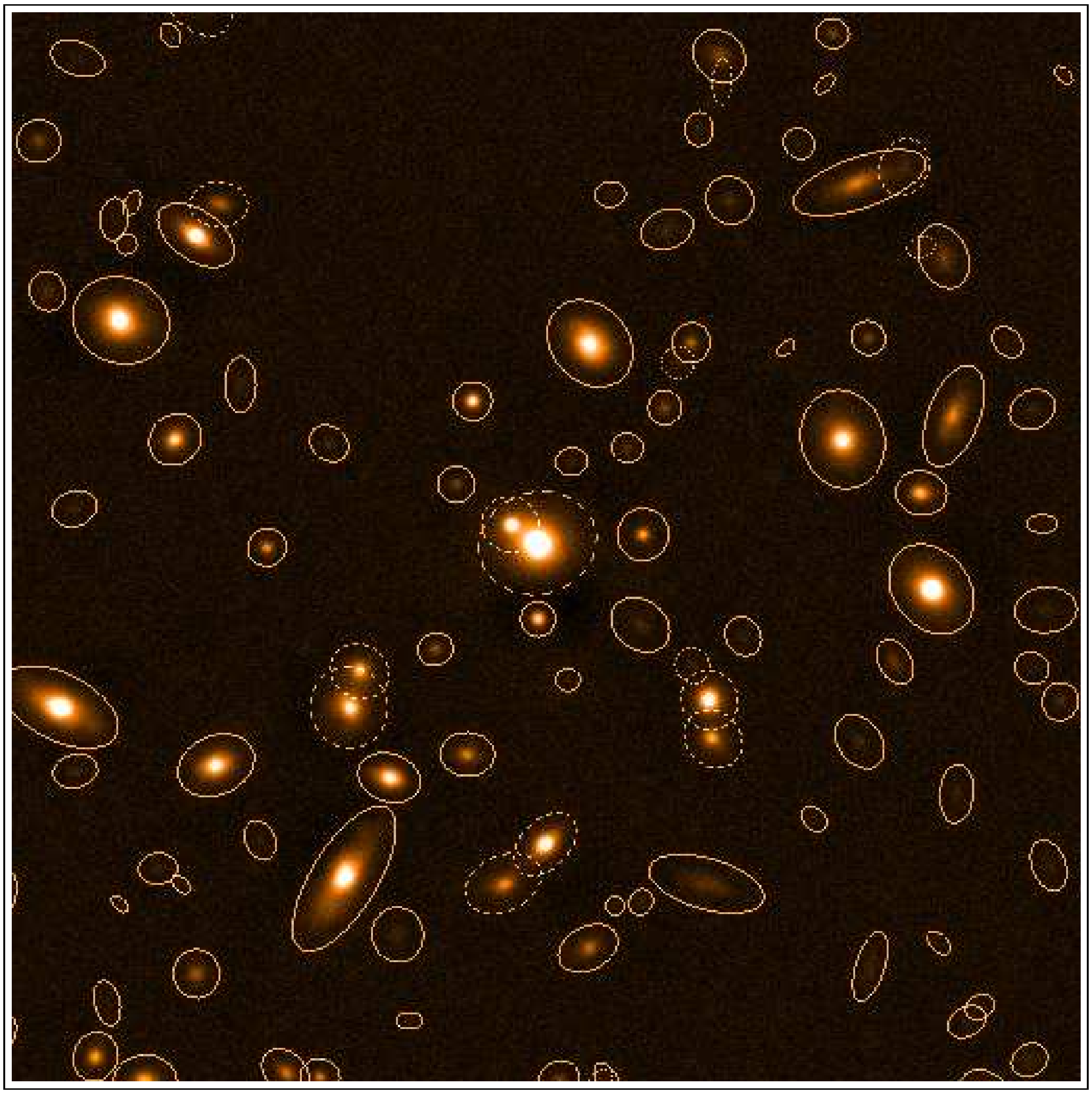
SExtractor
$v2.1.3$
User's guide

Draft

E. BERTIN

Institut d'Astrophysique
& Observatoire de Paris



Contents

1 What is SExtractor?

SExtractor (*Source-Extractor*) is a program that builds a catalogue of objects from an astronomical image. It is particularly oriented towards reduction of large scale galaxy-survey data, but it also performs well on moderately crowded star fields. Its main features are:

- Simplicity of usage and configuration.
- Speed: typically 500 kpixel/s with a Pentium2@450MHz.
- Ability to work with very large images (up to 65k $\times$ 65k pixels on 32bit machines, or 2G $\times$ 2G pixels on 64bit machines), thanks to buffered image access.
- Robust deblending of overlapping extended objects.
- Real-time filtering of images to improve detectability.
- Neural-Network-based star/galaxy classifier.
- Flexible catalogue output of desired parameters only.
- Pixel-to-pixel photometry in dual-image mode.
- Handling of weight-maps and flag-maps.
- Optimum handling of images with variable S/N.
- Special mode for photographic scans.
- Modularity of the code that enables one to implement new parameters.

The purpose of SExtractor was to find a compromise between refinement in both detection and measurements, and computational speed.

2 Installing the software

2.1 Software and hardware requirements

Since the beginning in 1993, the development of SExtractor was always made on Unix systems (successively: SUN-OS, HP/UX, SUN-Solaris, Digital Unix and GNU/Linux). Successful ports by external contributors have been reported on non-Unix OSes such as AMIGA-OS, DEC-VMS and even MS-DOS Windows95[1] and NT `;  )`. They are however not currently supported by the author, and Unix remains the recommended system for running SExtractor. The software is generally run in (ANSI) text-mode from a shell. A window system is therefore unnecessary with present versions.

On the hardware side, memory requirements obviously depend on the size of the images to be processed. But to give an idea, a typical processing of 1024 $\times$ 1024 pixel images should require no more than 8 MB of silicon memory. For very large images, (32000 $\times$ 32000 pixels or more), a minimum of 200MB is recommended. Swap-space can of course be put to contribution, although a strong performance hit is to be expected.

[1] Binaries are available on the WWW, see e.g. `http://www.tass-survey.org/tass/software/software.html#sextract`

2.2 Obtaining SExtractor

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

2.3 Installation

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

```
gzip -dc sex_2.x.x.tar.gz | tar xv
```

A new directory called `sextractor2.x.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`".

3 Using SExtractor

3.1 Syntax

SExtractor is run from the shell with the following syntax:

% `sex` *image* [`-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). Actually, *two* image filenames can be provided, separated by a comma:

% `sex` *image1*,*image2*

This syntax makes SExtractor run in the so-called "double-image mode": *image1* will be used for detection of sources, and *image2* for measurements only. *image1* and *image2* must have the same dimensions. Changing *image2* for another image will not modify the number of detected sources, neither affect their positional or basic shape parameters. But most photometric parameters, plus a few others, will use *image2* pixel values, which allows one to easily measure pixel-to-pixel colours.

3.2 The configuration file

SExtractor needs a configuration file. If no configuration file-name is specified in the command line, SExtractor tries to load a file called "`default.sex`" from the local directory.

3.2.1 Format

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, and not only for string parameters. Some parameters are assigned default values in SExtractor and can therefore be omitted from the configuration file; they are listed in §3.2.2.

3.2.2 Configuration parameter list

Here is a complete list of all the *configuration* parameters known to SExtractor. Many of them are just used with default values. Please refer to §?? for a detailed description of their meaning.

Parameter	default	type	Description
`ANALYSIS_THRESH`	—	*floats* ($n \leq 2$)	Threshold (in surface brightness) at which `CLASS_STAR` and FWHM_ operate. 1 argument: relative to Background RMS. 2 arguments: mu (mag.arcsec^{-2}), Zero-point (mag).
`ASSOC_DATA`	`2,3,4`	*integers* ($n \leq 32$)	Nos of the columns in the `ASSOC` file that will be copied to the catalog output.
`ASSOC_NAME`	`sky.list`	*string*	Name of the `ASSOC` ASCII file.
`ASSOC_PARAMS`	`2,3,4`	*integers* ($2 \leq n \leq 3$)	Nos of the columns in the `ASSOC` file that will be used as coordinates and weight for cross-matching.
`ASSOC_RADIUS`	`2.0`	*float*	Search radius (in pixels) for `ASSOC`.
`ASSOC_TYPE`	`MAG_SUM`	*keyword*	Method for cross-matching in `ASSOC`:
		`FIRST`	– keep values corresponding to the first match found,
		`NEAREST`	– values corresponding to the nearest match found,
		`MEAN`	– weighted-average values,
		`MAG_MEAN`	– exponentialy weighted-average values,
		`SUM`	– sum values,
		`MAG_SUM`	– exponentialy sum values,
		`MIN`	– keep values corresponding to the match with minimum weight,
		`MAX`	– keep values corresponding to the match with maximum weight.
`ASSOCSELEC_TYPE`	`MATCHED`	*keyword*	What sources are printed in the output catalog in case of `ASSOC`:
		`ALL`	– all detections,
		`MATCHED`	– only matched detections,
		`-MATCHED`	– only detections that were not matched.
`BACK_FILTERSIZE`	—	*integers* ($n \leq 2$)	Size, or Width,Height (in background meshes) of the background-filtering mask.

BACK_SIZE	—	*integer* ($n \leq 2$)	Size, or Width,Height (in pixels) of a background mesh.
BACK_TYPE	INTERNAL	*keywords* ($n \leq 2$)	What background is subtracted from the images:
		INTERNAL	– the internal interpolated background-map,
		MANUAL	– a user-supplied constant value provided in BACK_VALUE.
BACK_VALUE	0.0,0.0	*floats* ($n \leq 2$)	in BACK_TYPE MANUAL mode, the constant value to be subtracted from the images.
BACKPHOTO_THICK	24	*integer*	Thickness (in pixels) of the background LOCAL annulus.
BACKPHOTO_TYPE	GLOBAL	*keyword*	Background used to compute magnitudes:
		GLOBAL	– taken directly from the background map,
		LOCAL	– recomputed in a "rectangular annulus" around the object.
CATALOG_NAME	—	*string*	Name of the output catalogue. If the name "STDOUT" is given and CATALOG_TYPE is set to ASCII, ASCII_HEAD, or ASCII_SKYCAT, the catalogue will be piped to the standard output (*stdout*)
CATALOG_TYPE	—	*keyword*	Format of output catalog:
		ASCII	– ASCII table; the simplest, but space and time consuming,
		ASCII_HEAD	– as ASCII, preceded by a header containing information about the content,
		ASCII_SKYCAT	– SkyCat ASCII format (WCS coordinates required),
		FITS_1.0	– FITS format as in SExtractor 1,
		FITS_LDAC	– FITS "LDAC" format (the original image header is copied).
CHECKIMAGE_NAME	check.fits	*strings* ($n \leq 16$)	File name for each "check-image".
CHECKIMAGE_TYPE	NONE	*keywords* ($n \leq 16$)	Type of information to put in the "check-images":
		NONE	– no check-image,
		IDENTICAL	– identical to input image (useful for converting formats),
		BACKGROUND	– full-resolution interpolated background map,
		BACKGROUND_RMS	– full-resolution interpolated background noise map,
		MINIBACKGROUND	– low-resolution background map,
		MINIBACK_RMS	– low-resolution background noise map,
		-BACKGROUND	– background-subtracted image,
		FILTERED	– background-subtracted filtered image (requires FILTER = Y),

		OBJECTS	– detected objects,
		-OBJECTS	– background-subtracted image with detected objects blanked,
		APERTURES	– MAG_APER and MAG_AUTO integration limits,
		SEGMENTATION	– display patches corresponding to pixels attributed to each object.
CLEAN	—	*boolean*	If true, a "cleaning" of the catalogue is done before being written to disk.
CLEAN_PARAM	—	*float*	Efficiency of "cleaning".
DEBLEND_MINCONT	—	*float*	Minimum contrast parameter for deblending.
DEBLEND_NTHRESH	—	*integer*	Number of deblending sub-thresholds.
DETECT_MINAREA	—	*integer*	Minimum number of pixels above threshold triggering detection.
DETECT_THRESH	—	*floats* ($n \leq 2$)	Detection threshold. 1 argument: (ADUs or relative to Background RMS, see THRESH_TYPE). 2 arguments: μ (mag.arcsec^{-2}), Zero-point (mag).
DETECT_TYPE	CCD	*keyword*	Type of device that produced the image:
		CCD	– linear detector like CCDs or NICMOS,
		PHOTO	– photographic scan.
FILTER	—	*boolean*	If true, filtering is applied to the data before extraction.
FILTER_NAME	—	*string*	Name of the file containing the filter definition.
FILTER_THRESH		*floats* ($n \leq 2$)	Lower and higher thresholds (in background standard deviations) for a pixel to be considered in filtering (used for retina-filtering only).
FITS_UNSIGNED	N	*boolean*	Force 16-bit FITS input data to be interpreted as unsigned integers.
FLAG_IMAGE	flag.fits	*strings* ($n \leq 4$)	File name(s) of the "flag-image(s)".
FLAG_TYPE	OR	*keyword*	Combination method for flags on the same object:
		OR	– arithmetical OR,
		AND	– arithmetical AND,
		MIN	– minimum of all flag values,
		MAX	– maximum of all flag values,
		MOST	– most common flag value.
GAIN		*float*	"Gain" (conversion factor in e^-/ADU) used for error estimates of CCD magnitudes .
INTERP_MAXXLAG	16	*integers* ($n \leq 2$)	Maximum x gap (in pixels) allowed in interpolating the input image(s).
INTERP_MAXYLAG	16	*integers* ($n \leq 2$)	Maximum y gap (in pixels) allowed in interpolating the input image(s).

INTERP_TYPE	ALL	*keywords* ($n \leq 2$)	Interpolation method from the variance-map(s) (or weight-map(s)):
		NONE	– no interpolation,
		VAR_ONLY	– interpolate only the variance-map (detection threshold),
		ALL	– interpolate both the variance-map and the image itself.
MAG_GAMMA		*float*	γ of the emulsion (takes effect in PHOTO mode only).
MAG_ZEROPOINT		*float*	Zero-point offset to be applied to magnitudes.
MASK_TYPE	CORRECT	*keyword*	Method of "masking" of neighbours for photometry:
		NONE	– no masking,
		BLANK	– put detected pixels belonging to neighbours to zero,
		CORRECT	– replace by values of pixels symetric with respect to the source center.
MEMORY_BUFSIZE	—	*integer*	Number of scan-lines in the image-buffer. Multiply by 4 the frame width to get equivalent memory space in bytes.
MEMORY_OBJSTACK	—	*integer*	Maximum number of objects that the object-stack can contain. Multiply by 300 to get equivalent memory space in bytes.
MEMORY_PIXSTACK	—	*integer*	Maximum number of pixels that the pixel-stack can contain. Multiply by 16 to 32 to get equivalent memory space in bytes.
PARAMETERS_NAME	—	*string*	The name of the file containing the list of parameters that will be computed and put in the catalogue for each object.
PHOT_APERTURES	—	*floats* ($n \leq 32$)	Aperture diameters in pixels (used by MAG_APER).
PHOT_AUTOPARAMS	—	*floats* ($n = 2$)	MAG_AUTO controls: scaling parameter k of the 1st order moment, and minimum R_{min} (in units of A and B).
PHOT_AUTOAPERS	0.0,0.0	*floats* ($n = 2$)	MAG_AUTO minimum (circular) aperture diameters: estimation disk, and measurement disk.
PHOT_FLUXFRAC	0.5	*floats* ($n \leq 32$)	Fraction of FLUX_AUTO defining each element of the FLUX_RADIUS vector.
PIXEL_SCALE	—	*float*	Pixel size in arcsec (for surface brightness parameters, FWHM and star/galaxy separation only).
SATUR_LEVEL	—	*float*	Pixel value above which it is considered saturated.
SEEING_FWHM	—	*float*	FWHM of stellar images in arcsec (only for star/galaxy separation).

STARNNW_NAME	—	*string*	Name of the file containing the neural-network weights for star/galaxy separation.
THRESH_TYPE	RELATIVE	*keywords* ($n \leq 2$)	Meaning of the DETECT_THRESH and ANALYSIS_THRESH parameters :
		RELATIVE	– scaling factor to the background RMS,
		ABSOLUTE	– absolute level (in ADUs or in surface brightness).
VERBOSE_TYPE	NORMAL	keyword	How much SExtractor comments its operations:
		QUIET	– run silently,
		NORMAL	– display warnings and limited info concerning the work in progress,
		EXTRA_WARNINGS	– like NORMAL, plus a few more warnings if necessary,
		FULL	– display a more complete information and the principal parameters of all the objects extracted.
WEIGHT_GAIN	Y	*boolean*	If true, weight maps are considered as gain maps.
WEIGHT_IMAGE	weight.fits	*strings* ($n \leq 2$)	File name of the detection and measurement “weight-image”, respectively.
WEIGHT_TYPE	NONE	*keywords* ($n \leq 2$)	Weighting scheme (for single image, or detection and measurement images):
		NONE	– no weighting,
		BACKGROUND	– variance-map derived from the image itself,
		MAP_RMS	– variance-map derived from an external RMS-map,
		MAP_VAR	– external variance-map,
		MAP_WEIGHT	– variance-map derived from an external weight-map,

3.3 The catalog parameter file

In addition to the configuration file detailed above, SExtractor needs a file containing the list of parameters that will be listed in the output catalog for every detection. This allows the software to compute only catalog parameters that are needed. The name of this catalog-parameter file is traditionally suffixed with `.param`, and must be specified using the `PARAMETERS_NAME` config parameter.

3.3.1 Format

The format of the catalog parameter list is ASCII, and there must be *only one keyword per line*. Presently two kinds of keywords are recognized by SExtractor: scalars and vectors. Scalars, like `X_IMAGE`, yield single numbers in the output catalog. Vectors, like `MAG_APER(4)` or

[1] Optional parameter

`VIGNET(15,15)`, yield arrays of numbers. The order in which the parameters will be listed in the catalogue are the same as that of the keywords in the parameter list. Comments are allowed, they must begin with a "#". Here is a descriptive list of available parameter keywords.

3.4 Example of configuration

4 Overview of the software

The complete analysis of an image is done in two passes through the data. During the first pass, a model of the sky background is built, and a couple of global statistics are estimated. During the second pass, the image is background-subtracted, filtered and thresholded "on-the-fly". Detections are then deblended, pruned ("CLEANed"), photometered, classified and finally written to the output catalog. Let's now enter a little more into the details of each of these operations[2].

5 Handling of image data

SExtractor accepts images stored in FITS[3] format (Wells et al. 1981, see also `http://fits.gsfc.nasa.gov`). Currently, only "Basic FITS" images are recognized. For images with `NAXIS` > 2, only the first data-plane is loaded. Extensions are ignored. If WCS[4] information (Greisen & Calabretta 1995, `http://www.cv.nrao.edu/fits/documents/wcs/wcs.all.ps`) is available in the header, it is automatically used by SExtractor to compute astrometric parameters. Other astrometric descriptions like AST (*Starlink* format) or the solution coefficients of the DSS [5] plates are not recognized by the software.

In SExtractor, as in all similar programs, FITS axis "1" is traditionaly refered as the `X` axis, and FITS axis "2" as the `Y` axis.

6 Detection and segmentation

In SExtractor, the detection of sources is part of a process called *segmentation* in the image-processing vocabulary. Segmentation normally consists of identifying and separating image regions which have different properties (brightness, colour, texture...) or are delineated by edges. In the astronomical context, the segmentation process consists of separating objects from the sky background. This is however a somewhat imprecise definition, as astronomical sources have, on the images — and even often physically —, no clear boundaries, and may overlap. We shall therefore use the following working definition of an object in SExtractor: a group of pixels selected through some detection process and for which the flux contribution of an astronomical source is believed to be dominant over that of other objects. Note that this means that a simple x, y position vector alone cannot be handled by SExtractor as a detection: most measurement routines require some rough shape information about the objects.

[2]In the text, uppercase keywords in typewriter font refer to parameters from the configuration file or from the parameter file

[3]*Flexible Image Transport System*

[4]*World Coordinate System*

[5]*Digital Sky Survey*

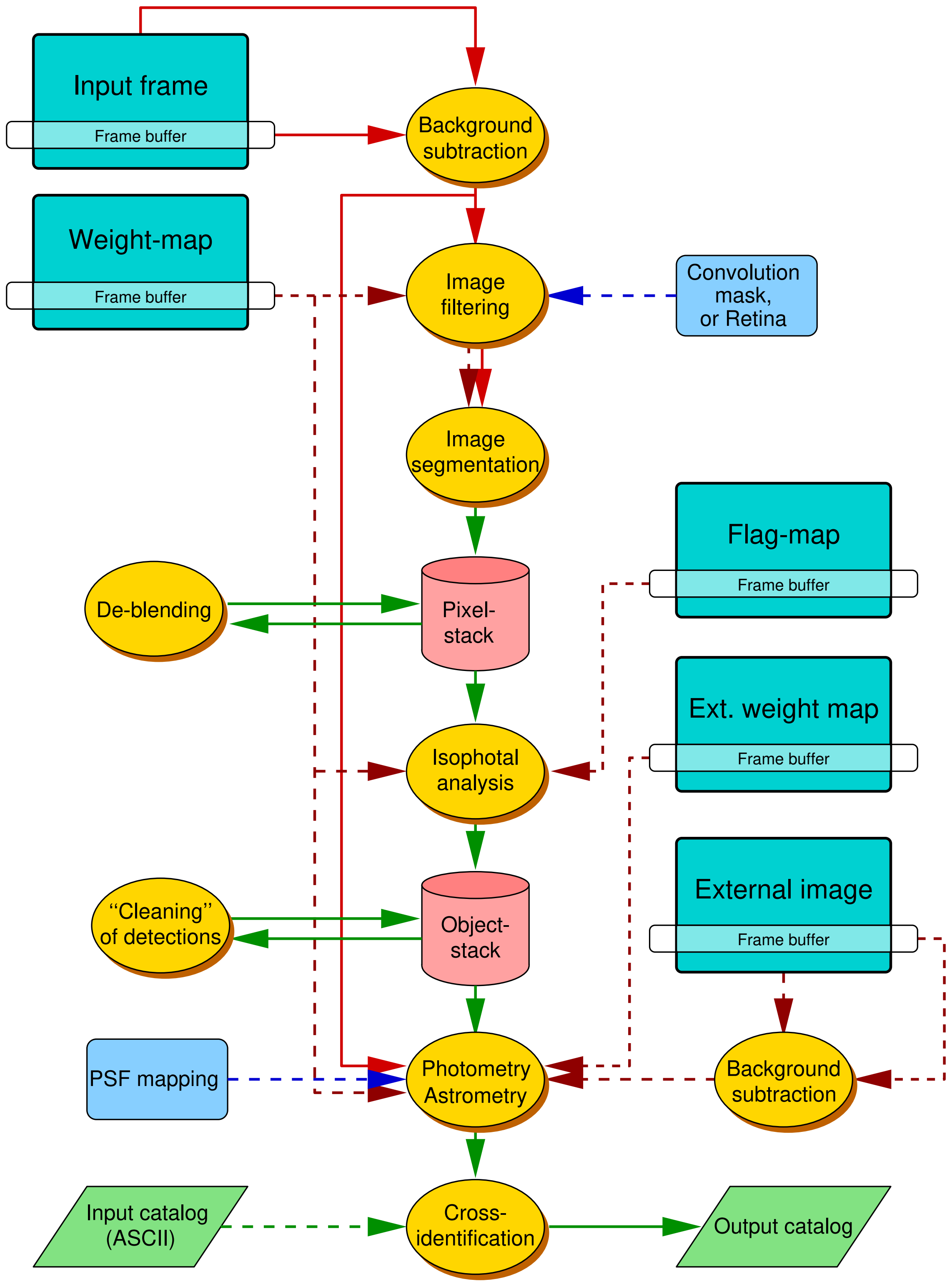


Figure 1: Layout of the main SExtractor procedures. Dashed arrows represent optional inputs.

Segmentation in SExtractor is achieved through a very simple thresholding process: a group of connected pixels that exceed some threshold above the background is identified as a detection. But things are a little bit more complicated in practice. First, on most astronomical images, the background is not constant over the frame, and its determination can be ambiguous in crowded regions. Second, the software has to operate on noisy data, and some filtering adapted to the characteristics of the image has to be applied prior to detection, to reduce the contamination by noise peaks. Third, many sources that overlap on the image are unlikely to be detected separately with a single detection threshold, and require a de-blending procedure, which is actually multi-thresholding in SExtractor. Each of these points will now be described in greater detail below. It is worth mentioning here that these 3 difficulties could, to a large extent, be bypassed using a wavelet decomposition (e.g. Bijaoui et al. 1998). Although such an algorithm might be implemented in a future version of SExtractor, current constraints in processing speed, available memory (processing of gigantic images) make the "pedestrian approach" still more interesting in the case of large scale surveys.

6.1 Background estimation

The value measured at each pixel is a function of the sum of a "background" signal and light coming from the objects of interest. To be able to detect the faintest of these objects and also to measure accurately their fluxes, one needs to have an accurate estimate of the background level in any place of the image, a "background map". Strictly speaking, there should be one background map per object, that is, what would the image look like if that object was absent. But, at least for detection, we may start by assuming that most discrete sources do not overlap too severely, which is generally the case for high galactic latitude fields.

To construct the background map, SExtractor 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:

$$\mathrm{Mode} = 2.5 \times \mathrm{Median} - 1.5 \times \mathrm{Mean} \tag{1}$$

This expression is different from the usual approximation

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

(e.g. Kendall and Stuart 1977), but was found to be more accurate with our clipped distributions, from the simulations we made. Fig. 2 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 $\approx 30\%$ noisier. For this reason we revert to the mean in non-crowded fields.

Once the grid is set up, a median filter can be applied to suppress possible local overestimations due to bright stars. The resulting background map is then simply a (natural) bicubic-spline interpolation between the meshes of the grid. In parallel with the making of the background map, an "RMS-background-map", that is, a map of the background noise in the image is produced. It will be used if the `WEIGHT_TYPE` parameter is set different from `NONE` (see §7.1).

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

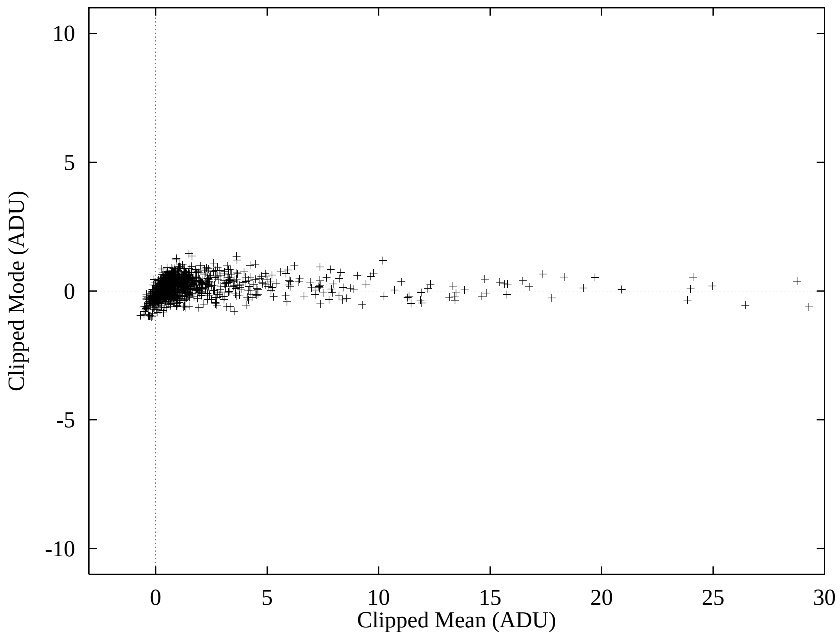


Figure 2: Simulations of 32×32 pixels background meshes polluted by random Gaussian profiles. The true background lies at 0 ADU. While being slightly noisier, the clipped "Mode" gives a more robust estimate than a clipped Mean in crowded regions.

6.1.1 Configuration parameters and tuning

. The choice of the mesh size (`BACK_SIZE`) is very important. If it is too small, the background estimation is affected by the presence of objects and random noise. Most importantly, part of the flux of the most extended objects can be absorbed in the background map. If the mesh size is too large, it cannot reproduce the small scale variations of the background. Therefore a good compromise has to be found by the user. Typically, for reasonably sampled images, a width[7] of 32 to 256 pixels works well. The user has some control over the background map by specifying the size of the median filter (`BACK_FILTERSIZE`). A width and height of 1 means that no filtering will be applied to the background grid. Usually a size of 3×3 is enough, 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 artefacts 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. It is important to note that all `BACK_` configuration parameters also affect the background-RMS map.

It is possible to subtract a user-supplied value, which can be zero if no background-subtraction is wanted.

6.1.2 CPU cost

. The background estimation operation can take a considerable time on the largest images, e.g. 10 minutes for a 32000×32000 frame on a Pentium2@450MHz.

[7]SExtractor offers the possibility of rectangular background meshes; but it is advised to use square ones, except in some very special cases (rapidly varying background in one direction for example).

6.2 Filtering

6.2.1 Convolution

Detectability is generally limited at the faintest flux levels by a background noise. The power-spectrum of the noise and that of the superimposed signal can be significantly different. Some gain in the ability to detect sources may therefore be obtained simply through appropriate linear filtering of the data, prior to segmentation. In low density fields, an optimal convolution kernel h ("matched filter") can be found that maximizes detectability. An estimator of detectability is for instance the signal-to-noise ratio at source position $(x_0, y_0) \equiv (0,0)$:

$$\left(\frac{\mathrm{S}}{\mathrm{N}}\right)^2 \equiv \frac{((s*h)(x_0,y_0))^2}{\overline{(n*h)^2}}, \tag{3}$$

where s is the signal to be detected, n the noise, and '$*$' the convolution operator. Moving to Fourier space, we get:

$$\left(\frac{\mathrm{S}}{\mathrm{N}}\right)^2 = \frac{(\int \mathcal{S}\mathcal{H}\,d\omega)^2}{\int |\mathcal{N}|^2|\mathcal{H}|^2\,d\omega}, \tag{4}$$

where $\mathcal{S}$ and $\mathcal{H}$ are the Fourier-transforms of s and h, respectively, and $|\mathcal{N}|^2$ is the power-spectrum of the noise. Remarking, using Schwartz inequality, that

$$\left|\int \mathcal{S}\mathcal{H}\,d\omega\right|^2 \leq \int \frac{|\mathcal{S}|^2}{|\mathcal{N}|^2} d\omega \int |\mathcal{N}|^2|\mathcal{H}|^2 d\omega\,, \tag{5}$$

we see that

$$\left(\frac{\mathrm{S}}{\mathrm{N}}\right)^2 \leq \int \frac{|\mathcal{S}|^2}{|\mathcal{N}|^2} d\omega\,. \tag{6}$$

Equality (maximum S/N) in (5) and (6) is achieved for

$$\frac{\mathcal{S}}{|\mathcal{N}|} \propto |\mathcal{N}|\mathcal{H}^*\,,\ \text{that is} \tag{7}$$

$$\mathcal{H} \propto \frac{\mathcal{S}^*}{|\mathcal{N}|^2}. \tag{8}$$

In the case of white noise (a valid approximation for many astronomical images, especially CCD ones), $|\mathcal{N}|^2 = cste$; the optimal convolution kernel for detecting stars is then the PSF flipped over the x and y directions. It may also be described as the cross-correlation with the template of the sources to be detected (for more details see, e.g. Bijaoui & Dantel 1970, or Das 1991).

There are of course a few problems with this method. First of all, many sources of unquestionable interest, like galaxies, appear in a variety of shapes and scales on astronomical images. A perfectly optimized detection routine should ultimately apply all relevant convolution kernels one after the other in order to make a complete catalog. Approximations to this approach are the (isotropic) wavelet analysis mentioned earlier, or the more empirical ImCat algorithm (Kaiser et al. 1995), for both of which sources to detect are assumed to be reasonably round. The impact on memory usage and processing speed of such refinements is currently judged too severe to be applied in SExtractor. Simple filtering does a good job in general: the topological constraints added by the segmentation process make the detection somewhat tolerant towards larger objects. Extended, very Low-Surface-Brightness (LSB) features found in astronomical images are often artifacts (flat-fielding errors, optical "ghosts" or halos). However, it is true that some of them can be genuine objects, like LSB galaxies, or distant galaxy clusters burried in the background noise. For detecting those with software like SExtractor, a specific processing is needed (see for instance Dalcanton et al. 1997 and references therein). The simplest way to achieve the detection of extended LSB objects in SExtractor is to work on `MINIBACK` check-images (see §**??**).

A second problem may occur because of overlaps with other objects. Convolving with a low-pass filter (the PSF has no negative side-lobes) diminishes the contrast between objects, and makes segmentation less effective in isolating individual sources. This can to some extent be recovered by deblending (see §6.4). In severely crowded fields however, confusion noise becomes the limiting factor for detection, and it is then advisable not to filter at all, or to use a bandpass-filter (compensated filter).

Finally, the PSF appears sometimes to be variable across the field. The convolution mask should ideally follow these changes in order to allow for optimal detection everywhere in the image. However, considering approximately-Gaussian PSF cores and convolution kernels, detectability is a rather slow function of their FWHMs[8]: a mismatch as large as 50% between the kernel FWHM and that of the PSF will lead to no more than a 10% loss in peak S/N (Irwin 1985). Considering that PSF variations are generally much smaller than this, filtering in SExtractor is limited to constant kernels.

6.2.2 Non-linear filtering

There are many situations in which convolution is of little help: filtering of (strongly) non-Gaussian noise, extraction of specific image patterns,... In those cases, one would like to extend the concept of a convolution kernel to that of a more general stationnary filter, able for instance to mimick boolean-like operations on pixels. What one wants like is thus a mapping from $\mathbf{R}^n$ to $\mathbf{R}$ around each pixel. But the more general the filter, the more difficult it is to design "by-hand" for each case, specifying how input pixel #i should be taken into account with respect to input pixel #j to form the output, etc.. The solution to this is machine-learning. Given a training set containing input and output pixels, a machine-learning software will adapt its internal parameters in order to minimize a "cost function" (generally a χ^2 error) and converge toward the desired mapping-function. These parameters can then for example be reloaded by a "read-only" routine to provide the actual filtering.

SExtractor implements this kind of "read-only" functionnality in the form of the so-called "retina-filtering". The **EyE**[9] software (Bertin 1997) performs neural-network-learning on input and output images to produce "retina-files". These files contain weights that describe the behaviour of the neural network. The neural network can thus be seen as an "artificial retina" that takes its stimuli from a small rectangular array of pixels and produces a response according to prior learning (for more details, see the **EyE** documentation). Typical applications of the retina are the identification of glitches.

6.2.3 What is filtered, and what isn't

Although filtering is a benefit for detection, it distorts profiles and correlates the noise; it is therefore nefast for most measurement tasks. Because of this, filtering is applied "on the fly" to the image, and *directly* affects only the detection process and the isophotal parameters described in §9.1. Other catalog parameters are indirectly affected — through the exact position of the barycenter and typical object extent —, but the effect is considerably less. Obviously, in double-image mode, filtering is only applied to the *detection* image.

[8]Full-Width at Half-Maximum

[9]*Enhance Your Extraction*

6.2.4 Image boundaries and bad pixels

"Virtual" pixels that lie outside image boundaries are arbitrarily set to zero. This makes sense since filtering occurs on a background-subtracted image. When weighting is applied (§??), bad pixels (pixels with weight < `WEIGHT_THRESH`) are interpolated by default (§7.5) and should therefore not cause much trouble. It is recommended not to turn-off interpolation of bad pixels when filtering is on.

6.2.5 Configuration parameters.

Filtering is triggered when the `FILTER` keyword is set to `Y`. If active, a file with name specified by `FILTER_NAME` is searched for and loaded. Filtering with large retinas can be extremely time consuming. In many cases, one is only interested in filtering pixels whose values stand out from the background noise. The `FILTER_THRESH keyword` can be given to specify the range of pixel values within which retina-filtering will be applied, in units of background noise standard deviation. If one value is given, it is interpreted as a lower threshold. For instance:

```
FILTER_THRESH 3.0
```

will allow filtering for pixel values exceeding $+3\sigma$ above the local background, whereas

```
FILTER_THRESH -10.0,3.0
```

will only allow filtering for pixel values between -10σ and $+3\sigma$. `FILTER_THRESH` has no effect on convolution.

The result of the filtering process can be verified through a `FILTERED` check-image: see §??.

6.2.6 CPU cost.

The SExtractor filtering routine is particularly optimized for small kernels. It thus provides a convenient way of filtering large image data. On a P2@450MHz, a convolution by a 5×5 kernel will typically contribute ≈ 1 second to the processing time of a 2048×2048 image. The numbers for non-linear (retina) filtering depend on the complexity of the neural network, but can be a hundred times larger.

6.2.7 Filter file formats.

As described above, two kinds of filter files are recognized by SExtractor: convolution files (traditionaly suffixed with "`.conv`"), and "retina" files ("`.ret`" extensions[10]).

Retina files are written exclusively by the **EyE** software, as FITS binary-tables.

Convolution files are in ASCII format. The following example shows the content of the `gauss_2.0_5x5.conv` file which can be found in the `config/` sub-directory of the SExtractor distribution:

```
CONV NORM
# 5x5 convolution mask of a gaussian PSF with FWHM = 2.0 pixels.
0.006319 0.040599 0.075183 0.040599 0.006319
0.040599 0.260856 0.483068 0.260856 0.040599
0.075183 0.483068 0.894573 0.483068 0.075183
```

[10] In SExtractor, file name extensions are just conventions; they are not used by the software to distinguish between different file formats.

```
0.040599 0.260856 0.483068 0.260856 0.040599
0.006319 0.040599 0.075183 0.040599 0.006319
```

The `CONV` keyword appearing at the beginning of the first line tells SExtractor that the file contains the description of a convolution mask (kernel). It can be followed by `NORM` if the mask is to be normalized to 1 before being applied, or `NONORM` otherwise[11]. The following lines should contain an equal number of kernel coefficients, separated by <space> of <TAB> characters. Coefficients in the example above are read from left to right and top to bottom, corresponding to increasing `NAXIS1` (x) and `NAXIS2` (y) in the image. Formatting is free, and number representations like `-0.14`, `-0.1400`, `-1.4e-1` or `-1.4E-01` are equivalent. The width of the kernel is set by the number of values per line, and its height is given by the number of lines. Lines beginning with "`#`" are treated as comments.

6.3 Thresholding

Thresholding is applied to the background-subtracted, filtered image to isolate connected groups of pixels. Each group defines the approximate position and shape of a basic SExtractor detection that will be processed further in the pipeline. Groups are made of pixels whose values exceed the local threshold and which touch each other at their sides or angles ("8-connectivity").

6.3.1 Configuration parameters.

Thresholding is mostly controlled through the `DETECT_THRESH` and `DETECT_MINAREA` keywords.

`DETECT_THRESH` sets the threshold value. If one single value is given, it is interpreted as a threshold in units of the background's standard deviation. For example:

```
DETECT_THRESH 1.5
```

will set the detection threshold at 1.5σ above the local background. It is important to note that em the standard deviation quoted here is that of the un`FILTER`ed image, at the pixel scale. Hence, on images with white Gaussian background noise for instance, a `DETECT_THRESH` of 3.0 will be close to optimum if low-pass `FILTER`ing is turned off, but sub-optimum (too high) if it is on. On the contrary, if the background noise of the image is intrinsically correlated from pixel-to-pixel, a `DETECT_THRESH` of 3.0 (with no `FILTER`ing) wil be too low and will result in a poor reliability of the extracted catalog.

Two numbers can be given as arguments to `DETECT_THRESH`, in which case the first one is interpreted as an absolute threshold in units of "magnitudes per square-arcsecond", and the second as a zero-point in the same units.

```
DETECT_THRESH 27.2,30.0
```

will for example set the threshold at $10^{-0.4(27.2-30)} = 13.18$ ADUs above the local background.

`DETECT_MINAREA` sets the minimum number of pixels a group should have to trigger a detection. Obviously this parameter can be used just like `DETECT_THRESH` to detect only bright and "big" sources, or to increase detection reliability. It is however more tricky to manipulate at low detection thresholds because of the complex interplay of object topology, noise correlations (including those induced by filtering), and sampling. In most cases it is therefore recommended to keep `DETECT_MINAREA` at a small value, typically 1 to 5 pixels, and let `DETECT_THRESH` and the filter define SExtractor's sensitivity.

[11] If the sum of the kernel coefficients happens to be exactly zero, the kernel is normalized to variance unity.

6.4 Deblending

Each time an object extraction is completed, the connected set of pixels passes through a sort of filter that tries to split it into eventual overlapping components. This case appears more frequently when the field is crowded or when the detection threshold is set very low. The deblending method adopted in SExtractor, is based on *multi-thresholding*, and works on any kind of object; but it is unable to deblend components that are so close that no saddle is present in their profile. However, as no assumption has to be made on the shape of the objects, it is perfectly suited for galaxies as well as for high galactic latitude stellar fields.

Typical problematic cases for deblending include patchy, extended **Sc** galaxies (which have to be considered as single entities), and close or interacting pairs of optically faint galaxies (which have to be considered as separate objects). Basically, the multi-thresholding algorithm employs a multiple isophotal analysis technique similar to those in use at the APM and the COSMOS machines (Beard, McGillivray and Thanish 1991); in a first time, each extracted set of connected pixels is re-thresholded at N levels linearly or exponentially spaced between its primary extraction threshold and its peak value. This gives us a sort of 2-dimensional "model" of the light distribution within the object(s), which is stored in the form of a tree structure (fig. 3). Then the algorithm goes downwards, from the tips of branches to the trunk, and decides at each junction whether it shall extract two (or more) objects or continue its way down. To meet the conditions described earlier, the following simple decision criteria are adopted: at any junction threshold t_i, any branch will be considered as a separate component if

(1) the integrated pixel intensity (above t_i) of the branch is greater than a certain fraction δ_c of the total intensity of the composite object.

(2) condition (1) is verified for at least one more branch at the same level i.

Note that ideally, condition (1) is both flux- and scale-invariant. However for faint, poorly resolved objects, the efficiency of the deblending is limited mostly by seeing and sampling. From the analysis of both small and extended galaxy images, a compromise value for the contrast parameter $\delta_c \sim 0.005$ proved to be optimum. This should normally exclude to separate objects with a difference in magnitude greater than ≈ 6.

The outlying pixels with flux lower than the separation thresholds have to be reallocated to the proper components of the merger. To do so, we have opted for a *statistical* approach: at each faint pixel we compute the contribution which is expected from each sub-object using a bivariate Gaussian fit to its profile, and turn it into a probability for that pixel to belong to the sub-object. For instance, a faint pixel lying halfway between two close bright stars having the same magnitude will be appended to one of these with equal probabilities. One big advantage of this technique is that the morphology of any object is completely defined simply through its list of pixels.

To test the effects of deblending on photometry and astrometry measurements, we made several simulations of photographic images of double stars with different separations and magnitudes under typical observational conditions (fig. 4). It is obvious that multiple isophotal techniques fail when there is no saddle point present in profiles (i.e. for distance between stars $< 2\sigma$ in the case of Gaussian images). We measured a magnitude error ≤ 0.2 mag and a shift of the centroid (≤ 0.4 pixels) for the fainter star in the very worst cases, but no other systematic effects were noticeable.

The user can control the multi-thresholding operation through 3 parameters. The first one is the number of deblending thresholds (`DEBLEND_NTHRESH`). A good value is 32. Higher values

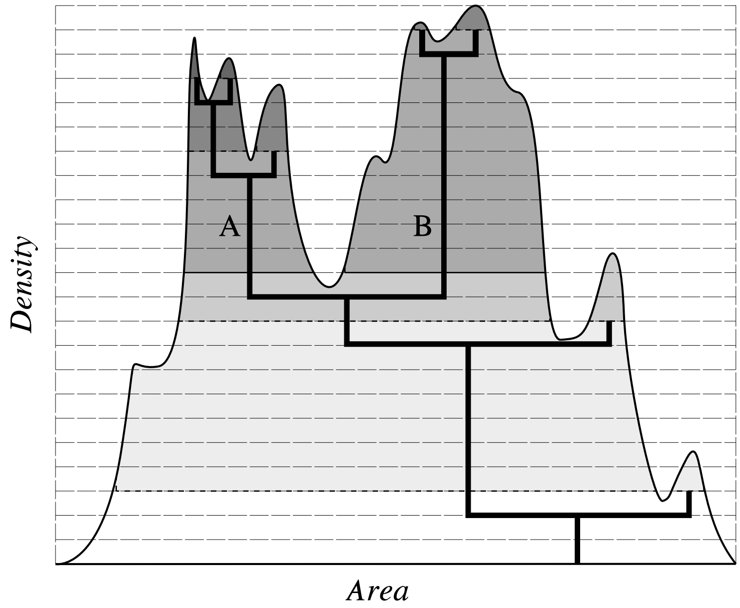


Figure 3: A schematic diagram of the method used to deblend a composite object. The area profile of the object (smooth curve) can be described in a tree-structured way (thick lines). The decision to regard or not a branch as a distinct object is determined according to its relative integrated intensity (tinted area). In that case above, the original object shall split into two components A and B. Remaining pixels are assigned to their most credible "progenitors" afterwards.

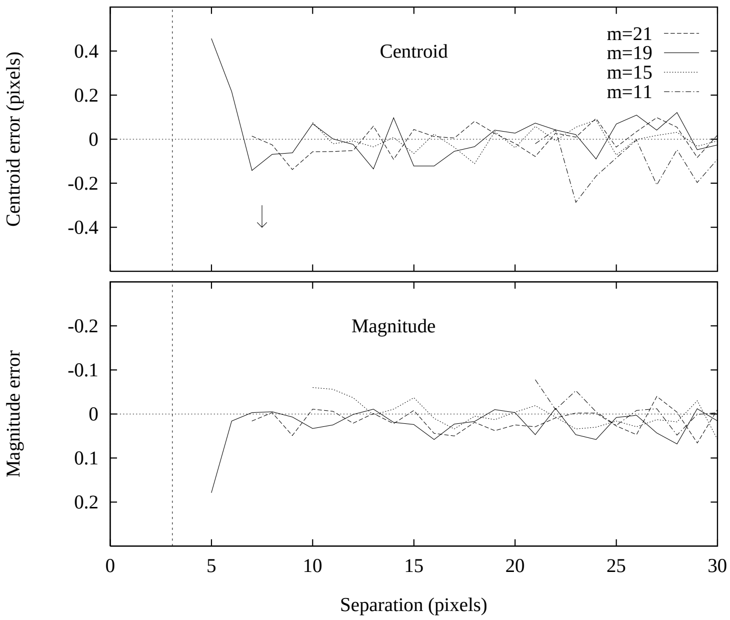


Figure 4: Centroid and corrected isophotal magnitude errors for a simulated 19^{th} magnitude star blended with a $11, 15, 19$ and 21^{th} mag. companion as a function of distance (expressed in pixels). Lines stop at the left when the objects are too close to be deblended. The dashed vertical line is the theoretical limit for unsaturated stars with equal magnitudes. In the centroid plot, the arrow indicates the direction of the neighbour. The simulation assumes a 1 hour exposure with the CERGA telescope on a IIIaJ plate and Moffat profiles with a seeing FWHM of 3 pixels (2 ").

are generally useless, except perhaps for images having an unusually high dynamic range. In case of memory problems, decreasing the number of thresholds to say, 8 or even less may be a solution. But then of course a degradation of the deblending performances may occur. The second parameter is the contrast parameter (`DEBLEND_MINCONT`). As described above, values from 0.001 to 0.01 give best results. Putting `DEBLEND_MINCONT` to 0 means that even the faintest local peaks in the profile will be considered as separate objects. Putting it to 1 means that no deblending will be authorized. The last parameter concerns the kind of scale used for the thresholds. If the image comes from photographic material, then a linear scale has to be used (`DETECTION_TYPE = PHOTO`). Otherwise, for an image obtained with a linear device like a CCD, an exponential scale is more appropriate (`DETECTION_TYPE = CCD`).

7 Weighting

The noise level in astronomical images is often fairly constant, that is, constant values for the gain, the background noise and the detection thresholds can be used over the whole frame. Unfortunately in some cases, like strongly vignetted or composited images, this approximation is no longer good enough. This leads to detecting clusters of detected noise peaks in the noisiest parts of the image, or missing obvious objects in the most sensitive ones. SExtractor is able to handle images with variable noise. It does it through *weight maps*, which are frames having the same size as the images where objects are detected or measured, and which describe the noise intensity at each pixel. These maps are internally stored in units of *absolute variance* (in ADU^2). We employ the generic term "weight map" because these maps can also be interpreted as quality index maps: infinite variance ($\geq 10^{30}$ by definition in SExtractor) means that the related pixel in the science frame is totally unreliable and should be ignored. The variance format was adopted as it linearizes most of the operations done over weight maps (see below).

This means that the noise covariances between pixels are ignored. Although raw CCD images have essentially white noise, this is not the case for co-added images, for which resampling may induce a strong correlation between neighbouring pixels. In theory, all non-zero covariances within the geometrical limits of the analysed patterns should be taken into account to derive thresholds or error estimates. Fortunately, the correlation length of the noise is often smaller than the patterns to be detected or measured, and constant over the image. In that case one can apply a simple "fudge factor" to the estimated variance to account for correlations on small scales. This proves to be a good approximation in general, although it certainly leads to underestimations for the smallest patterns.

7.1 Weight-map formats

SExtractor accepts in input, and converts to its internal variance format, several types of weight-maps. This is controlled through the `WEIGHT_TYPE` configuration keyword. These weight-maps can either be read from a FITS file, whose name is specified by the `WEIGHT_IMAGE` keyword, or computed internally. Valid `WEIGHT_TYPE`s are:

- `NONE`: No weighting is applied. The related `WEIGHT_IMAGE` and `WEIGHT_THRESH` (see below) parameters are ignored.
- `BACKGROUND`: the science image itself is used to compute internally a variance map (the related `WEIGHT_IMAGE` parameter is ignored). Robust (3σ-clipped) variance estimates are

first computed within the same background meshes as those described in §??[12]. The resulting low-resolution variance map is then bicubic-spline-interpolated on the fly to produce the actual full-size variance map. A check-image with CHECKIMAGE_TYPE MINIBACK_RMS can be requested to examine the low-resolution variance map.

- MAP_RMS: the FITS image specified by the WEIGHT_IMAGE file name must contain a weight-map in units of absolute standard deviations (in ADUs per pixel).
- MAP_VAR: the FITS image specified by the WEIGHT_IMAGE file name must contain a weight-map in units of relative variance. A robust scaling to the appropriate absolute level is then performed by comparing this variance map to an internal, low-resolution, absolute variance map built from the science image itself.
- MAP_WEIGHT: the FITS image specified by the WEIGHT_IMAGE file name must contain a weight-map in units of relative weights. The data are converted to variance units (by definition variance $\propto 1/\text{weight}$), and scaled as for MAP_VAR. MAP_WEIGHT is the most commonly used type of weight-map: a flat-field, for example, is generally a good approximation to a perfect weight-map.

7.2 Weight threshold

It may happen, that some weights are too low (or variances too high) to be of any interest: it is then more appropriate to discard such pixels than to include them in unweighted measurements such as FLUX_APER. To allow discarding these very bad pixels, a threshold can be set with the WEIGHT_THRESH parameter. The unit in which this threshold should be expressed is that of input data: ADUs for BACKGROUND and MAP_RMS maps, uncalibrated ADUs^2 for MAP_VAR, and uncalibrated weight-values for MAP_WEIGHT maps. Depending on the weight-map type, the threshold will set a lower or a higher limit for "bad pixel" values: higher for weights, and lower for variances and standard deviations. The default value is 0 for weights, and 10^{30} for variance and standard deviation maps.

7.3 Effect of weighting

Weight-maps modify the working of SExtractor in the following respects:

1. Bad pixels are discarded from the background statistics. If more than 50% of the pixels in a background mesh are bad, the local background value and its standard deviation are replaced by interpolation of the nearest valid meshes.
2. The detection threshold t above the local sky background is adjusted for each pixel i with variance σ_i^2: $t_i = \texttt{DETECT\_THRESH} \times \sqrt{\sigma_i^2}$, where DETECT_THRESH is expressed in units of standard deviations of the background noise. Pixels with variance above the threshold set with the WEIGHT_THRESH parameter are therefore simply not detected. This may result in splitting objects crossed by a group of bad pixels. Interpolation (see §7.5) should be used to avoid this problem. If convolution filtering is applied for detection, the variance map is convolved too. This yields optimum scaling of the detection threshold in the case where noise is uncorrelated from pixel to pixel. Non-linear filtering operations (like those offered by artificial retinae) are not affected.

[12] The mesh-filtering procedures act on the variance map, too.

3. The CLEANing process (§??) takes into account the exact individual thresholds assigned to each pixel for deciding about the fate of faint detections.

4. Error estimates like FLUXISO_ERR, ERRA_IMAGE, ... make use of individual variances too. Local background-noise standard deviation is simply set to $\sqrt{\sigma_i^2}$. In addition, if the WEIGHT_GAIN parameter is set to Y — which is the default —, it is assumed that the local pixel gain (i.e., the conversion factor from photo-electrons to ADUs) is inversely proportional to σ_i^2, its median value over the image being set by the GAIN configuration parameter. In other words, it is then supposed that the changes in noise intensities seen over the images are due to gain changes. This is the most common case: correction for vignetting, or coverage depth. When this is not the case, for instance when changes are purely dominated by those of the read-out noise, WEIGHT_GAIN shall be set to N.

5. Finally, pixels with weights beyond WEIGHT_THRESH are treated just like pixels discarded by the MASKing process (§??).

7.4 Combining weight maps

All the weighting options listed in §7.1 can be applied separately to detection and measurement images (§??), — even if some combinations may not always make sense. For instance, the following set of configuration lines:

```
WEIGHT_IMAGE rms.fits,weight.fits
WEIGHT_TYPE MAP_RMS,MAP_WEIGHT
```

will load the FITS file rms.fits and use it as an RMS map for adjusting the detection threshold and CLEANing, while the weight.fits weight map will only be used for scaling the error estimates on measurements. This can be done in single- as well as in dual-image mode (§??). WEIGHT_IMAGEs can be ignored for BACKGROUND WEIGHT_TYPEs. It is of course possible to use weight-maps for detection or for measurement only. The following configuration:

```
WEIGHT_IMAGE weight.fits
WEIGHT_TYPE NONE,MAP_WEIGHT
```

will apply weighting only for measurements; detection and CLEANing operations will remain unaffected.

7.5 Interpolation

TBW

8 Flags

A set of both *internal* and *external* flags is accessible for each object. Internal flags are produced by the various detection and measurement processes within SExtractor; they tell for instance if an object is saturated or has been truncated at the edge of the image. External flags come from "flag-maps": these are images with the same size as the one where objects are detected, where integer numbers can be used to flag some pixels (for instance, "bad" or noisy pixels). Different combinations of flags can be applied within the isophotal area that defines each object, to produce a unique value that will be written to the catalog.

8.1 Internal flags

The internal flags are *always* computed. They are accessible through the `FLAGS` catalog parameter, which is a short integer. `FLAGS` contains, coded in decimal, all the extraction flags as a sum of powers of 2:

`1`	The object has neighbours, bright and close enough to significantly bias the `MAG_AUTO` photometry[13], or bad pixels (more than 10% of the integrated area affected),
`2`	The object was originally blended with another one,
`4`	At least one pixel of the object is saturated (or very close to),
`8`	The object is truncated (too close to an image boundary),
`16`	Object's aperture data are incomplete or corrupted,
`32`	Object's isophotal data are incomplete or corrupted[14],
`64`	A memory overflow occurred during deblending,
`128`	A memory overflow occurred during extraction.

For example, an object close to an image border may have `FLAGS` = 16, and perhaps `FLAGS` = 8+16+32 = 56.

8.2 External flags

SExtractor understands that it must process external flags when `IMAFLAGS_ISO` or `NIMAFLAGS_ISO` are present in the catalog parameter file. It then looks for a FITS image specified by the `FLAG_IMAGE` keyword in the configuration file. The FITS image must contain the flag-map, in the form of a 2-dimensional array of 8, 16 or 32 bits integers. It must have the same size as the image used for detection. Such flag-maps can be created using for example the **WeightWatcher** software (Bertin 1997).

The flag-map values for pixels that coincide with the isophotal area of a given detected object are then combined, and stored in the catalog as the long integer `IMAFLAGS_ISO`. 5 kinds of combination can be selected using the `FLAG_TYPE` configuration keyword:

- `OR`: the result is an arithmetic (bit-to-bit) **OR** of flag-map pixels.
- `AND`: the result is an arithmetic (bit-to-bit) **AND** of non-zero flag-map pixels.
- `MIN`: the result is the minimum of the (signed) flag-map pixels.
- `MAX`: the result is the maximum of the (signed) flag-map pixels.
- `MOST`: the result is the most frequent non-zero flag-map pixel-value.

The `NIMAFLAGS_ISO` catalog parameter contains a number of relevant flag-map pixels: the number of non-zero flag-map pixels in the case of an `OR` or `AND` `FLAG_TYPE`, or the number of pixels with value `IMAFLAGS_ISO` if the `FLAG_TYPE` is `MIN`,`MAX` or `MOST`.

9 Measurements

Once sources have been detected and deblended, they enter the measurement phase. There are in SExtractor two categories of measurements. Measurements from the first category are made

[13]This flag can be activated only when `MAG_AUTO` magnitudes are requested.

[14]This flag is inherited from SExtractor V1.0, and has been kept for compatibility reasons. With SExtractor V2.0+, having this flag activated doesn't have any consequence for the extracted parameters.

on the isophotal object profiles. Only pixels above the detection threshold are considered. Many of these isophotal measurements (like X_IMAGE, Y_IMAGE, etc.) are necessary for the internal operations of SExtractor and are therefore executed even if they are not requested. Measurements from the second category have access to all pixels of the image. These measurements are generally more sophisticated and are done at a later stage of the processing (after CLEANing and MASKing).

9.1 Positional parameters derived from the isophotal profile

The following parameters are derived from the spatial distribution $\mathcal{S}$ of pixels detected above the extraction threshold. *The pixel values I_i are taken from the (filtered) detection image.*

Note that, unless otherwise noted, all parameter names given below are only prefixes. They must be followed by "_IMAGE" if the results shall be expressed in pixel units (see §..), or "_WORLD" for World Coordinate System (WCS) units (see §9.2). Example: THETA → THETA_IMAGE. In all cases parameters are first computed in the image coordinate system, and then converted to WCS if requested.

9.1.1 Limits: XMIN, YMIN, XMAX, YMAX

These coordinates define two corners of a rectangle which encloses the detected object:

$$\begin{aligned} \texttt{XMIN} &= \min_{i \in \mathcal{S}} x_i, &(9)\\ \texttt{YMIN} &= \min_{i \in \mathcal{S}} y_i, &(10)\\ \texttt{XMAX} &= \max_{i \in \mathcal{S}} x_i, &(11)\\ \texttt{YMAX} &= \max_{i \in \mathcal{S}} y_i, &(12) \end{aligned}$$

where x_i and y_i are respectively the x-coordinate and y-coordinate of pixel i.

9.1.2 Barycenter: X, Y

Barycenter coordinates generally define the position of the "center" of a source, although this definition can be inadequate or inaccurate if its spatial profile shows a strong skewness or very large wings. X and Y are simply computed as the first order moments of the profile:

$$\texttt{X} = \overline{x} = \frac{\sum\limits_{i \in \mathcal{S}} I_i x_i}{\sum\limits_{i \in \mathcal{S}} I_i}, \tag{13}$$

$$\texttt{Y} = \overline{y} = \frac{\sum\limits_{i \in \mathcal{S}} I_i y_i}{\sum\limits_{i \in \mathcal{S}} I_i}. \tag{14}$$

Actually, x_i and y_i are summed relative to XMIN and YMIN in order to reduce roundoff errors in the summing.

9.1.3 Position of the peak: XPEAK, YPEAK

It is sometimes useful to have the position XPEAK,YPEAK of the pixel with maximum intensity in a detected object, for instance when working with likelihood maps, or when searching for artifacts. For better robustness, PEAK coordinates are computed on *filtered* profiles if available. On symetrical profiles, PEAK positions and barycenters coincide within a fraction of pixel (XPEAK and YPEAK coordinates are quantized by steps of 1 pixel, thus XPEAK_IMAGE and YPEAK_IMAGE are integers). This is no longer true for skewed profiles, therefore a simple comparison between PEAK and barycenter coordinates can be used to identify asymetrical objects on well-sampled images.

9.1.4 2nd order moments: X2, Y2, XY

(Centered) second-order moments are convenient for measuring the spatial spread of a source profile. In SExtractor they are computed with:

$$\begin{aligned}
\mathtt{X2} &= \overline{x^2} = \frac{\sum\limits_{i\in\mathcal{S}} I_i x_i^2}{\sum\limits_{i\in\mathcal{S}} I_i} - \overline{x}^2, \qquad (15)\\
\mathtt{Y2} &= \overline{y^2} = \frac{\sum\limits_{i\in\mathcal{S}} I_i y_i^2}{\sum\limits_{i\in\mathcal{S}} I_i} - \overline{y}^2, \qquad (16)\\
\mathtt{XY} &= \overline{xy} = \frac{\sum\limits_{i\in\mathcal{S}} I_i x_i y_i}{\sum\limits_{i\in\mathcal{S}} I_i} - \overline{x}\,\overline{y}, \qquad (17)
\end{aligned}$$

These expressions are more subject to roundoff errors than if the 1st-order moments were subtracted before summing, but allow both 1st and 2nd order moments to be computed in one pass. Roundoff errors are however kept to a negligible value by measuring all positions relative here again to XMIN and YMIN.

9.1.5 Basic shape parameters: A, B, THETA

These parameters are intended to describe the detected object as an elliptical shape. A and B are its semi-major and semi-minor axis lengths, respectively. More precisely, they represent the maximum and minimum spatial *rms* of the object profile along any direction. THETA is the position-angle between the A axis and the NAXIS1 image axis. It is counted counter-clockwise. Here is how they are computed:

2nd-order moments can easily be expressed in a referential rotated from the x, y image coordinate system by an angle $+\theta$:

$$\begin{array}{rcllll}
\overline{x_\theta^2} &=& \cos^2\theta\,\overline{x^2} & +\sin^2\theta\,\overline{y^2} & -2\cos\theta\sin\theta\,\overline{xy}, & \\
\overline{y_\theta^2} &=& \sin^2\theta\,\overline{x^2} & +\cos^2\theta\,\overline{y^2} & +2\cos\theta\sin\theta\,\overline{xy}, & (18)\\
\overline{xy_\theta} &=& \cos\theta\sin\theta\,\overline{x^2} & -\cos\theta\sin\theta\,\overline{y^2} & +(\cos^2\theta-\sin^2\theta)\,\overline{xy}. &
\end{array}$$

One can find interesting angles θ_0 for which the variance is minimized (or maximized) along x_θ:

$$\left.\frac{\partial\overline{x_\theta^2}}{\partial\theta}\right|_{\theta_0} = 0, \qquad (19)$$

which leads to

$$2\cos\theta\sin\theta_0\,(\overline{y^2}-\overline{x^2})+2(\cos^2\theta_0-\sin^2\theta_0)\,\overline{xy}=0. \tag{20}$$

If $\overline{y^2}\neq\overline{x^2}$, this implies:

$$\tan 2\theta_0 = 2\frac{\overline{xy}}{\overline{x^2}-\overline{y^2}}, \tag{21}$$

a result which can also be obtained by requiring the covariance $\overline{xy_{\theta_0}}$ to be null. Over the domain $[-\pi/2,+\pi/2[$, two different angles — with opposite signs — satisfy (21). By definition, THETA is the position angle for which $\overline{x^2_\theta}$ is *max*imized. THETA is therefore the solution to (21) that has the same sign as the covariance $\overline{xy}$. A and B can now simply be expressed as:

$$\begin{aligned}\mathtt{A}^2 &= \overline{x^2}_{\mathtt{THETA}}, \quad \text{and} &(22)\\ \mathtt{B}^2 &= \overline{y^2}_{\mathtt{THETA}}. &(23)\end{aligned}$$

A and B can be computed directly from the 2nd-order moments, using the following equations derived from (18) after some tedious arithmetics:

$$\begin{aligned}\mathtt{A}^2 &= \frac{\overline{x^2}+\overline{y^2}}{2}+\sqrt{\left(\frac{\overline{x^2}-\overline{y^2}}{2}\right)^2+\overline{xy}^2}, &(24)\\ \mathtt{B}^2 &= \frac{\overline{x^2}+\overline{y^2}}{2}-\sqrt{\left(\frac{\overline{x^2}-\overline{y^2}}{2}\right)^2+\overline{xy}^2}. &(25)\end{aligned}$$

Note that A and B are exactly halves the a and b parameters computed by the COSMOS image analyser (Stobie 1980,1986). Actually, a and b are defined by Stobie as the semi-major and semi-minor axes of an elliptical shape with constant surface brightness, which would have the same 2nd-order moments as the analysed object.

9.1.6 Ellipse parameters: CXX, CYY, CXY

A, B and THETA are not very convenient to use when, for instance, one wants to know if a particular SExtractor detection extends over some position. For this kind of application, three other ellipse parameters are provided; CXX, CYY and CXY. They do nothing more than describing the same ellipse, but in a different way: the elliptical shape associated to a detection is now parameterized as

$$\mathtt{CXX}(x-\overline{x})^2+\mathtt{CYY}(y-\overline{y})^2+\mathtt{CXY}(x-\overline{x})(y-\overline{y})=R^2, \tag{26}$$

where R is a parameter which scales the ellipse, in units of A (or B). Generally, the isophotal limit of a detected object is well represented by $R\approx 3$ (Fig. 5). Ellipse parameters can be derived from the 2nd order moments:

$$\begin{aligned}\mathtt{CXX} &= \frac{\cos^2\mathtt{THETA}}{\mathtt{A}^2}+\frac{\sin^2\mathtt{THETA}}{\mathtt{B}^2} = \frac{\overline{y^2}}{\sqrt{\left(\frac{\overline{x^2}-\overline{y^2}}{2}\right)^2+\overline{xy}^2}} &(27)\\ \mathtt{CYY} &= \frac{\sin^2\mathtt{THETA}}{\mathtt{A}^2}+\frac{\cos^2\mathtt{THETA}}{\mathtt{B}^2} = \frac{\overline{x^2}}{\sqrt{\left(\frac{\overline{x^2}-\overline{y^2}}{2}\right)^2+\overline{xy}^2}} &(28)\\ \mathtt{CXY} &= 2\cos\mathtt{THETA}\sin\mathtt{THETA}\left(\frac{1}{\mathtt{A}^2}-\frac{1}{\mathtt{B}^2}\right) = -2\frac{\overline{xy}}{\sqrt{\left(\frac{\overline{x^2}-\overline{y^2}}{2}\right)^2+\overline{xy}^2}} &(29)\end{aligned}$$

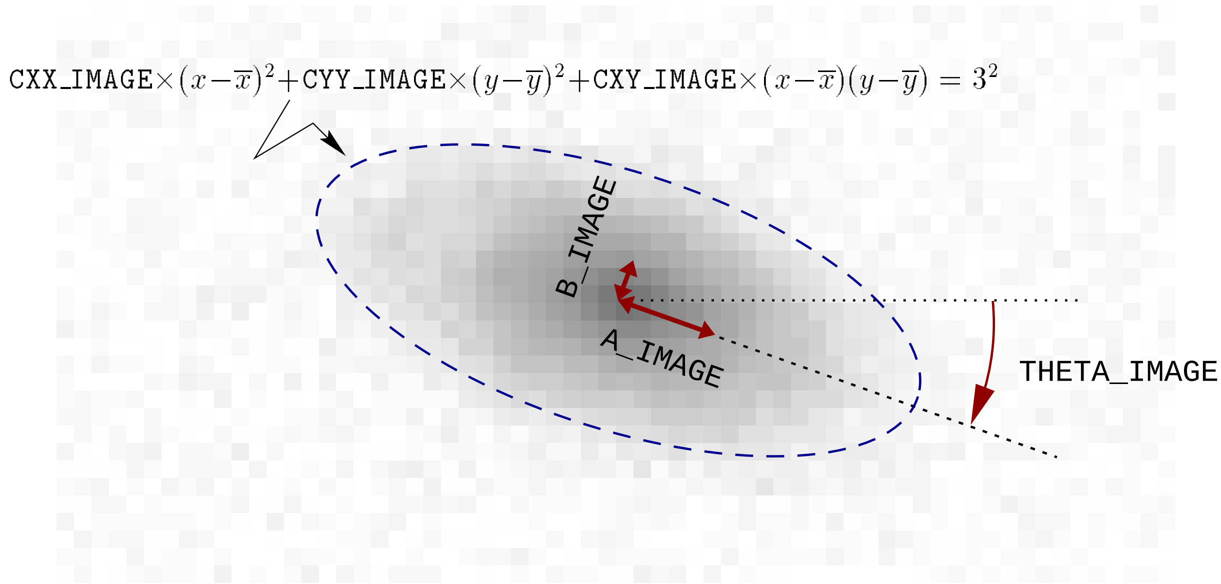


Figure 5: The meaning of basic shape parameters. THETA is negative here.

9.1.7 By-products of shape parameters: ELONGATION, ELLIPTICITY

15

These parameters are directly derived from A and B:

$$\begin{aligned} \texttt{ELONGATION} &= \frac{\texttt{A}}{\texttt{B}} \quad \text{and} && (30)\\ \texttt{ELLIPTICITY} &= 1 - \frac{\texttt{B}}{\texttt{A}}. && (31) \end{aligned}$$

9.1.8 Position errors: ERRX2, ERRY2, ERRXY, ERRA, ERRB, ERRTHETA, ERRCXX, ERRCYY, ERRCXY

Uncertainties on the position of the barycenter can be estimated using photon statistics. Of course, this kind of estimate has to be considered as a lower-value of the real error since it does not include, for instance, the contribution of detection biases or the contamination by neighbours. As SExtractor does not currently take into account possible correlations between pixels, the variances simply write:

$$\texttt{ERRX2} = \mathrm{var}(\overline{x}) = \frac{\sum\limits_{i\in\mathcal{S}} \sigma_i^2 (x_i - \overline{x})^2}{\left(\sum\limits_{i\in\mathcal{S}} I_i\right)^2}, \qquad (32)$$

$$\texttt{ERRY2} = \mathrm{var}(\overline{y}) = \frac{\sum\limits_{i\in\mathcal{S}} \sigma_i^2 (y_i - \overline{y})^2}{\left(\sum\limits_{i\in\mathcal{S}} I_i\right)^2}, \qquad (33)$$

[15] Such parameters are dimensionless and therefore do not accept any _IMAGE or _WORLD suffix

$$\texttt{ERRXY} \quad = \mathrm{cov}(\overline{x},\overline{y}) = \frac{\sum_{i\in\mathcal{S}} \sigma_i^2 (x_i - \overline{x})(y_i - \overline{y})}{\left(\sum_{i\in\mathcal{S}} I_i\right)^2}. \tag{34}$$

σ_i is the flux uncertainty estimated for pixel i:

$$\sigma_i^2 = {\sigma_B}_i^2 + \frac{I_i}{g_i}, \tag{35}$$

where σ_{Bi} is the local background noise and g_i the local gain — conversion factor — for pixel i (see §**??** for more details). Major axis `ERRA`, minor axis `ERRB`, and position angle `ERRTHETA` of the 1σ position error ellipse are computed from the covariance matrix exactly like in 9.1.5 for shape parameters:

$$\texttt{ERRA}^2 = \frac{\mathrm{var}(\overline{x}) + \mathrm{var}(\overline{y})}{2} + \sqrt{\left(\frac{\mathrm{var}(\overline{x}) - \mathrm{var}(\overline{y})}{2}\right)^2 + \mathrm{cov}^2(\overline{x},\overline{y})}, \tag{36}$$

$$\texttt{ERRB}^2 = \frac{\mathrm{var}(\overline{x}) + \mathrm{var}(\overline{y})}{2} - \sqrt{\left(\frac{\mathrm{var}(\overline{x}) - \mathrm{var}(\overline{y})}{2}\right)^2 + \mathrm{cov}^2(\overline{x},\overline{y})}, \tag{37}$$

$$\tan(2 \times \texttt{ERRTHETA}) = 2\frac{\mathrm{cov}(\overline{x},\overline{y})}{\mathrm{var}(\overline{x}) - \mathrm{var}(\overline{y})}. \tag{38}$$

And the ellipse parameters are:

$$\texttt{ERRCXX} = \frac{\cos^2 \texttt{ERRTHETA}}{\texttt{ERRA}^2} + \frac{\sin^2 \texttt{ERRTHETA}}{\texttt{ERRB}^2} = \frac{\mathrm{var}(\overline{y})}{\sqrt{\left(\frac{\mathrm{var}(\overline{x}) - \mathrm{var}(\overline{y})}{2}\right)^2 + \mathrm{cov}^2(\overline{x},\overline{y})}}, \tag{39}$$

$$\texttt{ERRCYY} = \frac{\sin^2 \texttt{ERRTHETA}}{\texttt{ERRA}^2} + \frac{\cos^2 \texttt{ERRTHETA}}{\texttt{ERRB}^2} = \frac{\mathrm{var}(\overline{x})}{\sqrt{\left(\frac{\mathrm{var}(\overline{x}) - \mathrm{var}(\overline{y})}{2}\right)^2 + \mathrm{cov}^2(\overline{x},\overline{y})}}, \tag{40}$$

$$\texttt{ERRCXY} = 2 \cos \texttt{ERRTHETA} \sin \texttt{ERRTHETA} \left(\frac{1}{\texttt{ERRA}^2} - \frac{1}{\texttt{ERRB}^2}\right) \tag{41}$$

$$= -2\frac{\mathrm{cov}(\overline{x},\overline{y})}{\sqrt{\left(\frac{\mathrm{var}(\overline{x}) - \mathrm{var}(\overline{y})}{2}\right)^2 + \mathrm{cov}^2(\overline{x},\overline{y})}}. \tag{42}$$

9.1.9 Handling of "infinitely thin" detections

Apart from the mathematical singularities that can be found in some of the above equations describing shape parameters (and which SExtractor handles, of course), some detections with very specific shapes may yield quite unphysical parameters, namely null values for `B`, `ERRB`, or even `A` and `ERRA`. Such detections include single-pixel objects and horizontal, vertical or diagonal lines which are 1-pixel wide. They will generally originate from glitches; but very undersampled and/or low S/N genuine sources may also produce such shapes. How to handle them?

For basic shape parameters, the following convention was adopted: if the light distribution of the object falls on one single pixel, or lies on a sufficiently thin line of pixels, which we translate mathematically by

$$\overline{x^2}\,\overline{y^2} - \overline{xy}^2 < \rho^2, \tag{43}$$

then $\overline{x^2}$ and $\overline{y^2}$ are incremented by ρ. ρ is arbitrarily set to $1/12$: this is the variance of a 1-dimensional top-hat distribution with unit width. Therefore $1/\sqrt{12}$ represents the typical minor-axis values assigned (in pixels units) to undersampled sources in SExtractor.

Positional errors are more difficult to handle, as objects with very high signal-to-noise can yield extremely small position uncertainties, just like singular profiles do. Therefore SExtractor first checks that (43) is true. If this is the case, a new test is conducted:

$$\mathrm{var}(\overline{x})\,\mathrm{var}(\overline{y}) - \mathrm{covar}^2(\overline{x},\overline{y}) < \rho_e^2, \qquad (44)$$

where ρ_e is arbitrarily set to $\left(\sum_{i\in\mathcal{S}}\sigma_i^2\right) / \left(\sum_{i\in\mathcal{S}} I_i\right)^2$. If (44) is true, then $\overline{x^2}$ and $\overline{y^2}$ are incremented by ρ_e.

9.2 Astrometry and `WORLD` coordinates

All SExtractor measurements related to positions, distances and areas in the image, like those described above can also be expressed in `WORLD` coordinates in the output catalog. These parameters simply have the `_WORLD` suffix instead of the `_IMAGE` appended to them. The conversion from `IMAGE` to `WORLD` coordinates is presently performed by using information found in the FITS header of the *measurement* image, even if the parameter is originally computed from the *detection* image (like the basic shape parameters for instance).

To understand how this is done in practice, let's have a general look at the way the mapping from `IMAGE` to `WORLD` coordinates is currently described in a FITS image header. First, a linear transformation (involving most of the time only scaling and possibly rotation, and more rarely shear) allows one to convert integer pixel positions (1,2,...) for each axis to some "projected" coordinate system. This is where you might want to stop if your `WORLD` system is just some kind of simple focal-plane coordinate-system (in meters for instance), or for a calibrated wavelength axis (spectrum). Now, the FITS WCS (World Coordinate System) convention allows you to apply to these "projected coordinates" a non-linear transformation, which is in fact a de-projection back to "local" spherical (celestial) coordinates. Many types of projections are allowed by the WCS convention, but the traditional tangential (gnomonic) projection is the most commonly used. The last step of the transformation is to convert these local coordinates, still relative to a projection reference point, to an absolute position in celestial longitude and latitude, for instance right-ascension and declination. For this one needs to know the reference frame of the coordinate system, which often requires some information about the equinox or the observation date. At this level, all transformations are matters of spherical trigonometry.

9.2.1 Angular coordinates

We will not describe here the transformations $(\alpha, \delta) = f(x, y)$ themselves. SExtractor de-projections rely on the WCSlib 2.4 written by Mark Calabretta, and all the details concerning those can be found in Greisen & Calabretta (1995). In addition to the `_WORLD` parameters, 3 purely angular "world" coordinates are available in SExtractor, expressed in decimal degrees:

1. `_SKY` coordinates: strictly identical to `_WORLD` coordinates, except that the units are explicitely degrees. They correspond to sky coordinates in the "native" system without any precession correction, conversion, etc.

2. `_J2000` coordinates: precession corrections are applied in the FK5 system to convert to J2000 coordinates if necessary.

3. `_B1950` coordinates: precession corrections are computed in the FK5 system and transformation to B1950 is applied.

Transformation to J2000 or B1950 is done without taking into account proper motions, which are obviously unknown for the detected objects. In both cases, epoch 2000.0 is assumed.

Here is a list of catalog parameters currently supporting angular coordinates: **TBW**

9.2.2 Use of the FITS keywords for astrometry

TBW

9.3 Cross-identification within SExtractor

SExtractor allows one to perform on-line cross-identification of each detection with an `ASCII` list. Although the cross-identification algorithm is not very sophisticated — it works in pixel-coordinates only —, it is particularly convenient for assessing SExtractor performances, on image simulations from instance. Configuration parameters related to cross-identification are prefixed with `ASSOC`.

9.3.1 The `ASSOC` list

The `ASSOC` process is initiated by requesting in the parameter file at least one of the `ASSOC` catalog parameters: `VECTOR_ASSOC` and `NUMBER_ASSOC`. Then SExtractor looks for an `ASCII` file (let's call it the `ASSOC` list) whose file name has to be specified by the `ASSOC_NAME` configuration keyword. The `ASSOC` list must contain columns of numbers separated by spaces or tabs. Each line describes a source that will enter the cross-identification process. Lines with zero characters, or beginning with "`#`" (for comments) are ignored. This means you may use any `ASCII` catalog generated by a previous SExtractor run as an `ASSOC` list.

To perform the cross-identification, SExtractor needs to know what are the columns that contain the x and y coordinates[16] in the `ASSOC` list. These shall be specified using the `ASSOC_PARAMS` configuration parameter. The syntax is: "`ASSOC_PARAMS` c_x,c_y[,c_Z]", where c_x and c_y are the positions of the columns containing the x and y coordinates (the first column has position 1). c_Z (optional) specifies an extra column containing some "Z" parameter that may be used for controlling or weighting the `ASSOC` process. Z will typically be a flux estimate. c_Z is required if `ASSOC_TYPE` is `MIN`, `MAX`, `MEAN` or `MAG_MEAN` (see below).

9.3.2 Controlling the `ASSOC` process

Two configuration parameters control the `ASSOC` process. The first one, `ASSOC_RADIUS`, accepts a decimal number which represents the maximum distance (in pixels) one should have between the barycenter of the current SExtractor detection and an `ASSOC`-list member to consider a match. This number must of course account for positional uncertainties in both catalogs. In most cases, a value of a few pixels will do just fine. The second configuration parameter, `ASSOC_TYPE`, accepts a keyword as argument and selects the kind of identification procedure one wants to operate:

- `FIRST`: this is the simplest way of performing a cross-identification. It does not require the c_Z column in `ASSOC_PARAMS`. The first geometrical match encountered while scanning

[16] The x and y coordinates must comply with the FITS (and SExtractor) convention: by definition, the center of the first pixel in the image array has pixel-coordinates (1.0,1.0).

the ASSOC list is retained as the actual match. This can used for catalogs with low spatial density.

- NEAREST: this option does not require the c_Z column in ASSOC_PARAMS. The match is performed with the ASSOC-list member the closest (in position) to the current detection, provided that it lies within the ASSOC_RADIUS.
- SUM: all parameters issued from ASSOC-list members which geometrically match the current detection are summed. c_Z is not required.
- MAG_SUM: all parameters c_i issued from ASSOC-list members which geometrically match the current detection are combined using the following law: $-2.5\log(\sum_i 10^{-0.4c_i}$. This option allows one to sum flux contributions from magnitude data. c_Z is not required.
- MIN: among all geometrical matches, retains the ASSOC-list member which has the smallest Z parameter.
- MAX: among all geometrical matches, retains the ASSOC-list member which has the largest Z parameter.
- MEAN: all parameters issued from ASSOC-list members which geometrically match the current detection are weighted-averaged, using the Z parameter as the weight.
- MAG_MEAN: all parameters issued from ASSOC-list members which geometrically match the current detection are weighted-averaged, using $10^{-0.4Z}$ as the weight. This option is useful for weighting catalog sources with magnitudes.

9.3.3 Output from ASSOC

Now that we have described the cross-identification process, let's see how informations coming from the matching with the ASSOC list are propagated to the output SExtractor catalog.

The output of ASSOC data in SExtractor catalog is done through the VECTOR_ASSOC() catalog parameter. VECTOR_ASSOC() is a vector, each element of which refers to a column from the input ASSOC list. VECTOR_ASSOC() contains either ASSOC-list member data from the best match (if ASSOC_TYPE is FIRST, NEAREST, MIN or MAX), or a combination of ASSOC-list member data (if ASSOC_TYPE is MEAN, MAG_MEAN, SUM or MAG_SUM). If no match has been found, it just contains zeros. The NUMBER_ASSOC contains the number of ASSOC-list members that geometrically match the current SExtractor detection, and obviously, if different from zero, indicates that VECTOR_ASSOC() has a meaningful content.

The ASSOC_DATA configuration parameter is used to tell SExtractor to which column refers each element of VECTOR_ASSOC(). The syntax of ASSOC_DATA is similar to that of ASSOC_PARAMS: "ASSOC_DATA c_1,c_2,c_3,..." where the c_i are the column positions in the ASSOC list. The special case "ASSOC_DATA 0" tells SExtractor to propagate all columns from the ASSOC file to the output catalog.

There are situations where it might be desirable to keep in the output SExtractor catalog only those detections that were matched with some ASSOC-list member. Such a feature is controlled by the ASSOCSELEC_TYPE configuration parameter, which accepts one of the three following keywords:

- ALL: keep all SExtractor detections, regardless of matching. This is the default.

- `MATCHED`: keep only SExtractor detections that were matched with at least one `ASSOC`-list member.
- `-MATCHED`: keep only SExtractor detections that were not matched with any `ASSOC`-list member.

Acknowledgments

References

[1] Beard S.M., McGillivray H.T., Thanisch P.F., 1990, *MNRAS* **247**, 311

[2] Bertin E., E.y.E 1.1, User's manual, 1997, Leiden

[3] Bertin E., WeightWatcher 1.2, User's manual, 1997, ESO

[4] Bijaoui A., Dantel M., 1991, *A&A* **6**, 51

[5] Bijaoui A., Slezak E., Vandame B., 1998, in *Astrophysics and Algorithms: a DIMACS Workshop on Massive Astronomical Data Sets*

[6] Dalcanton J.J., Spergel D.N., Gunn J.E., Schmidt M., Schneider D.P., 1997, *AJ*, **114**, 635

[7] Das P.K., 1991, *Optical Signal Processing*, (Springer-Verlag)

[8] Greisen E.W., Calabretta M., 1995, A.D.A.S.S. 4, 233 ()

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

[10] Irwin M.J., 1985, *MNRAS* **214**, 575

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

[12] Kaiser N., Squires G., Broadhurst T., 1995, *ApJ*, **449**, 460

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

[14] Kron R.G., 1980, *ApJS* **43**, 305

[15] Lutz R.K., 1979, *The Computer Journal* **23**, 262

[16] Moffat A.F.J., 1969,

[17] Wells D.C., Greisen E.W., Harten R.H., 1981, *A&AS* **44**, 363

A Appendices

A.1 FAQ (Frequently Asked Questions)

Fairly often, I am asked by users about the reason for some limitations or choices in the way things are done in SExtractor. In this section, I try to justify them.

Q: SExtractor supports WCS. So why isn't it possible to have the `ASSOC` cross-identification working in α, δ (or any other world-coordinates)?

A: The `ASSOC` list which is used for cross-identification can be very long (100,000 objects or more). Performing an exhaustive cross-id in real-time can therefore be extremely slow, unless the `ASSOC` coordinates are sorted in some way beforehand. In pixel coordinates, such a sorting is simple and very efficient, as SExtractor works line-by-line; but it would be much more difficult in the general WCS context. This is why this hasn't been implemented, considering it as beyond the scope of SExtractor.

Q: Why isn't the detection threshold expressed in units of the background noise standard deviation in the `FILTER`*ed* image ?

A: There are two reasons for this. First, it makes the threshold independent of the choice of a `FILTER`, which is a good thing. Second, having σ measured on the `FILTER`ed image may have given un-informed users the wrong impression that increasing filtering systematically improves the detectability of any source, whereas it depends on scale.