.

K. M. Jones

October 17, 2021

Contents

1 Introduction

Overview

There are 3 main steps to making a catalog with these programs. Each of these processes, including the input required and the output produced, is detailed in **Section 2**:

- **Pre-processing:** Pre-process the native science images using:

python **maskitextra.py** *file_sci.fits file_wht.fits*

- **Reprojecting:** Reproject your blue-band image to the pixel space of the red-band image using:

  python **reproj_kmj.py** *file_blue_sci.fits file_red_sci.fits outputfilename*

- **Convolving:** Convolve by running through the Convolutin_py3_v2.ipynb Cells 1-6 and then 7-9 and selecting the best convolution image

- **Catalog Creation** Create a dual-image mode catalog by running through the make_a_catalog.ipynb Cells 1-8 and turning on the write-to-file in Cell 8.

In addition, if you want to update or add to the metatable, or make plots of the depths which are reached in each band, you will need to run through a distinct process:

- **Pre-Processing** Use Source Extractor to produce object subtracted and segmentation images associated with the noise-normalized version of the science image and mask the output object subtracted image

- **Empty Aperture Simulation** run the Empty Aperture Simulation, and pipe its output to a universal text file via:

python **runEAA.py** *file_objsub_newnorm3_maskd.fits shortfilename_filter_osnn3m* pixelscale

- **Plotting and Fitting** Plot and fit values by running through the Working_Depth.ipynb Cells 1-4

Provided jupyter notebooks, python programs/scripts, data tables, and parameter files

- Working_Depth.ipynb
- make_a_catalog.ipynb
- Convolutin_py3_v2.ipynb
- colormag_plots.ipnb
- **depthcalc.py**
- **iterate_convolution.py**
- **getweight.py**
- **quickmask.py**
- **maskitextra.py**
- **runEAA.py**
- **reproj_kmj.py**
- metatable_v1.txt
- default_dualimage.sex
- default_objsub_norm.sex
- def_phot.param
- *gogreen_photometry.fits*
- *Images: _sci.fits and _wht.fits for multiple bands*

Requirements to get started

- **Source Extractor (2.19.5+)**. Install this in the source_extractor folder. You will also require the default_objsub_norm.sex and default_dualimage.sex, as well as def_phot.param. For SE to run successfully, you'll need ATLAS v3.6+ and FFTw v3.0+
- Registered _sci.fits and _wht.fits images
- **sewpy**, a python-usable interface for calling on Source Extractor. this can be finicky to set up.
- Most of the jupyter notebooks require **Python 3.6** and **astropy**

2 Making a Catalog from HST Archival Images

If you want to make a catalog where you compare the fluxes of the same sources in different bands, there are a couple of steps you need to do before you can effectively use Source Extractor in dual-image mode to make the catalog. The images need to be a) registered, b) reprojected and c) convolved. More specifically, you want to account for any way in which the images would be distorted or different from each other due to systematics from the instrument.

There are programs/notebooks for both b) and c) that are detailed below but it is important to note that no registration is done. The original data worked with (drizzled HST mosaics of GOGREEN clusters from G. Brammer database) were already registered. If you are working with a different data set you may need to add a registration stage before you do the below

2.1 Pre-Processing

You will eventually need a mask file that details where data exists in your image files and where it does not. To get this, run your native science and weight images through the **maskitextra.py** program. You will need to do this for every band you intend to use in the creation of the catalog–at minimum, we recommend doing this for a "blue" and "red" band, which fall on either side of the 4000 Åbreak for the cluster's redshift. For the 5 GoGreen clusters, the recommended "blue" and "red" bands are included in the metatable.

Example

General form:
python **maskitextra.py** *file_sci.fits file_wht.fits*

Parameters:

- *file_sci.fits*: full name and path of the relevant native science file
- *file_wht.fits*: full name and path of the weight file associated with *file_sci.fits*

Example:
python maskitextra.py /home/k689j329/HSTdata/j020548m5829/j020548m5829-f606w_drc_sci.fits /home/k689j329/HSTdata/j020548m5829/j020548m5829-f606w_drc_wht.fits

Example Output:

- j020548m5829-f606w_drc_sci_mask.fits
- j020548m5829-f606w_drc_sci_newnorm3.fits

2.2 Reprojectin'

Reprojection is memory intensive so you will need to run this on a machine that can be tied up for approximately 30 minutes. Simply run your native blue and red band images through reproj_kmj.py, and also give it the name you want the output file to take

Example

General form:
python reproj_kmj.py *file_blue_sci.fits file_red_sci.fits outputfilename.fits*

Parameters:

- *file_blue_sci.fits*: full name and path of the native science file for the band blueward of the 4000 Å break. This is the file that will be changed, reprojected to the pixel space of the other file.
- *file_red_sci.fits*: full name and path of the native science file for the band redward of the 4000 Å break. This file will not be changed/reprojected.
- *outputfilename.fits*: name you want your output file to take

Example:
python **reproj_kmj.py** /home/k689j329/HSTdata/j020548m5829/j020548m5829-f606w_drc_sci.fits /home/k689j329/HSTdata/j020548m5829/j020548m5829-f140w_drz_sci.fits j0205_f606w_reprj.fits

Example Output:

- j0205_f606w_reprj.fits
- j0205_f606w_reprj_wht.fits

Note to more experienced users: this program uses Astropy's **reproject_exact** to preserve the total energy received per unit area on the sky as one reprojects from the blue image's pixel space to the red image's pixel space. The quicker **reproject_interp** is **strongly NOT recommended for matched aperture photometry**.

2.3 Convolutin'

In contrast to the previous two steps, the process of convolution is more complex. Fortunately for you, you can use the Convolutin_py3_v2.ipynb jupyter notebook to go through this process. You will need **sewpy** to function for this to run.

- First update the parameters in Cell 3
- Then run Cells 4, 5, and 6
- Inspect the graph produced. It is most likely that there is no point that matches exceptionally well with the horizontal line (and even if there is, the calculations might as well be refined to higher accuracy), so proceed to run Cells 7, 8, and 9.

- Use the results of Cell 9 to determine which of the convolved images produced is the most appropriate to use going forward.

The point of this notebook is to figure out the convolution kernel that, when used on the reprojected blue image, best reproduces the PSF seen in the native red image. At an initial guess, an appropriate FWHM to define a Gaussian2Dkernel to use for convolution could be obtained by

$$FWHM_{compromise} = \sqrt{FWHM_{red}^{2.0} - FWHM_{blue}^{2.0}} \quad (1)$$

where the FWHM are in arcseconds. Alas, this simple idea does not necessarily produce the best PSF match, so the notebook constructs several convolution kernels with a reasonable range of FWHM, convolves them with the reprojected blue image, and calculates the median stellar FWHM that results.

2.3.1 Adjusting Cell 3 Input Parameters

- *fileblue*: full name and path of the native science file for the band blueward of the 4000 Åbreak.
- *filered*: full name and path of the native science file for the band redward of the 4000 Å break.
- *filereprojd* full name and path of the reprojected science file for the band blueward of the 4000 Å break. This should be the same as the *outputfilename.fits* provided to **reproj_kmj.py** (unless of course you've renamed that file since then).
- ident1 and ident2: two distinct strings that will be added to the names of the output files. Note that the part of the program that convolves the data will not overwrite existing files, so if you do not update these identifiers nothing will be convolved after the first time, no matter how many parameters you change. The default is 'sept_' and 'septb_', but you can use any string you desire.
- fwhm_blue_pix: a measurement, in pixels, of the typical size of the PSF of a star in the native blue image *fileblue*
- fwhm_red_pix: a measurement, in pixels, of the typical size of the PSF of a star in the native red image *filered* [1]

[1] Fwhm_blue_pix and fwhm_red_pix are used in Cell 4 to derive a range **yarray1** of possible FWHM for convolution kernels with which the *filereprojd* is convolved. Note however that FWHM_red_pix is also used in more extensive calculations. Fwhm_red_pix is fed into iterate_convolution.makecat() as the seeing, which is used by Source Extractor to distinguish between 'star-like' and 'galaxy-like' objects in the CLASS_STAR value that is returned (where the closer an object's value is to 1, the more likely it is a star). This CLASS_STAR key is in turn used to identify which objects are included in the calculation of the median FWHM *measured* from the convolved image. Since the point is to minimize the difference between this measured median FWHM and the FWHM we want to match, it can have an impact on that calculation. However, I have done a few small iterations (1/2 and 2x the FWHM I'd measured for J2106) and the program seems robust to such variations. High precision, therefore, is not a requirement; and note that the 'acceptable range' of the CLASS_STAR value can be adjusted via the **star_limit** kwarg when iterate_convolution.calcfwhm() is called in Cells 5 and 8.

2.3.2 Outputs:

Once you have run through Cells 1-9, you will have 15 new *.fits* images with the following format (where i=0-14):

filereprojd+'_conv_'+str(ident1)+str(i)+'.fits'

And 20 new *.fits* images with the following format (where j=0-19):

filereprojd+'_conv_'+str(ident2)+str(j)+'.fits'

The graphs and text[2] produced by Cell 9 are then used to identify which of these images you will use going forward, to the catalog creation in **Section 2.4**

Example:

If you used filereprojd=j0205_f606w_reprj.fits, ident1=a, and ident2=b, you would end up with

j0205_f606w_reprj_conv_a0.fits
j0205_f606w_reprj_conv_a1.fits
...
j0205_f606w_reprj_conv_a14.fits

and

j0205_f606w_reprj_conv_b0.fits
...
j0205_f606w_reprj_conv_b19.fits

Notes for more experienced users:

Note if you are using other than a drizzled image you may need a more complicated pixel scale conversion than simply taking CD1_1 from the header; this can be adjusted in Cell 4

The actual process of the convolution is done via **iterate_convolution.py's conv2Dgauss()** function. This takes in the following parameters:

- The fits file to be convolved (which should be the reprojected blue band image)
- The FWHM of the convolution kernel you wish to use (do not use the variance–the function corrects for that).
- an integer that gets fed into the produced filename (should default to zero)

As suggested by its name, this uses a 2DGaussian kernel, symmetric. However if you require a different kernel it is possible to adjust the function for your needs.

You can adjust the number of output files produced by changing the values used to define yarray1 and ya2 in Cells 4 and 7, respectively. Currently set to 15 and 20.

[2]hmm, through a bit of a glitch it's possible to end up with two kernels that are equally close to the required FWHM_red_recalcd. Looks like this will break the part that simply tells you which file to use. You can make your own assessment based on the graphs (note that given the way the arrays are constructed, the kernels associated with each file are plotted starting at the right and going to the left; also note 0 based indexing for python). Alternatively, simply have the program in Cell 6 or Cell 9 do a print(np.where(p)[0]) (or (g)[0]) and pick one to set t or k to

In Cells 5 and 8, the **for** loop makes a catalog of sources from each convolved image produced (using textbfiterate_convolution.py's makecat() function) and then calculates a median FWHM of the most stellar-like objects using **iterate_convolution.py's calcfwhm()** function. This function has a number of ****kwargs** that can be changed to select which objects are considered in the calculation:

def calcfwhm(table1,star_limit=0.8,maglow=-2.0, maghigh=-0.5,fwhmlim=4.5,plottin=False)

where maglow, maghigh, and fwhmlim are used to constrain the 'stellar sequence' in a FWHM-magnitude plot, and star_limit corresponds the associated CLASS_STAR value output by Source Extractor. This can vary dramatically depending on the properties of the input image–for example, when making the j210604m5845 catalog, CLASS_STAR peaked at about 0.85, making the more typical limit of stellar objects having CLASS_STAR >0.95 unreasonable. Just something to keep an eye on if you want to use it.

2.4 Dual-image Mode Source Extractor and Making a Catalog

Now, you've got two images with objects in the same pixel space/position, with stellar sources giving the same PSF implying that the light is distributed around the given source in the same way in both bands. You'll use these two images in 'dual-image mode' of Source Extractor. Dual-image mode identifies "sources" in one band (the most reliable band) and uses the locations of those sources in subsequent images/bands to measure photometric properties including flux and sizes.

- (a) Update default_dualimage.sex if needed
- (b) Update the values in Cells 3, 5, 7 and 8
- (c) (Probably) update variable names from Cell 5-8
- (d) Run Cells 1-7, skipping Cell 6.5
- (e) Uncomment the t.writeto(outputcatalog) line in Cell 8 and run Cell 8

2.4.1 (a) Update default_dualimage.sex if needed

This parameter file tells Source Extractor how to identify objects (sources) when it runs. The main parameters that you might adjust fall into several categories:

- Detection parameters: DETECT_MINAREA, DETECT_THRESH, FILTER_NAME
- Output control parameters: PHOT_APERTURES, PHOT_AUTOPARAMS, SEEING_FWHM
- Background subtraction parameters: My experiments indicated that a constant background subtraction was more reliable than an automatically calculated background, mostly due to large scale variation in the images. This requires BACK_TYPE = MANUAL and BACK_VALUE derived

2.4.2 (b) Update the values in Cells 3, 5, 7 and 8

- Like most of the .ipynb provided, the main definition of files to be processed by the notebook happens in Cell 3. Here you must update:

- filered: full name and path of the native science file for the band redward of the 4000 Å break. This is the un-altered image.
- fileblue: full name and path of the reprojected and convolved science file for the band blueward of the 4000 Å break. This file will be the one you selected after running through the convolution process; it should look something like: j0205_f606w_reprj_conv_b13.fits
- fileoutrr and fileoutrb: these are the names of the catalogs produced by source extractor. They will be opened and processed in the rest of the notebook.
- colnames and colnames2: These are lists of all of the values produced in the catalog files by running source extractor. They should correspond to the outputs set in the def_phot.param file, with one exception: only one of the identifying number columns needs to be retained, preferably that in colnames (set as 'SE_ID_f105w').
- redwht and bluewht: These are the full name and path to weight images associated with filered and fileblue, respectively.
- outputcatalog: This will be the name of the produced catalog. The recommended format is 'catalog_dualmode_j020548m5829.fits'

- There are values that will need to be changed in other cells as well, however. In Cells 5 and 8, the FLUX_AUTO, FLUX_ISO, and FLUX_APER columns are identified for manipulation using e.g.:

  nms=='FLUX_AUTO_f105w'
  nms=='FLUX_ISO_f105w'
  nms=='FLUX_APER_f105w'

  and similarly for f606w. Since the values within colnames and colnames2 changed in Cell 3, the values you are comparing nms to will also change. Cell 8 adds the names of the normalized weight columns, creating coltot– these must be updated as well.

- Cell 7 also needs to be updated as Cells 5 and 8 are, but in addition you must also update:

  - fileredmask
  - filered seg
  - filebluemask
  - fileblueseg

  at least until Cell 6.5 works and the needed values can be extracted from the metatable.

2.4.3 (c) (Probably) update variable names from Cell 5-8

As you can see scrolling through it, the notebook has a lot of variables specifically named for the filter band to which they are referring. You will see things like:

countsauto_f105w=data_total[:,int(np.where(nms=='FLUX_AUTO_f105w')[0])]

flambda_auto_f105w=countsauto_f105w*photflam_red
wif105w=np.array(wi_f105w)
and
fluxerr_iso_f606w=(C1_f606w*N_ISO_f105w + C2_f606w*N_ISO_f606w*N_ISO_f606w)/denomb

If you want to maintain careful accuracy, in addition to updating the parameters called on ('FLUX_AUTO_f105w' might become 'FLUX_AUTO_f160w' depending on your red and blue band images), you should correct all the variable names to reflect these changes as well:

countsauto_**f160w**=data_total[:,int(np.where(nms=='FLUX_AUTO_f160w')[0])]

Technically this should be unnecessary, but for matters of neatness and consistency it is a step that should probably be done.

2.4.4 (d) Run Cells 1-7, skipping Cell 6.5

The Cell 6.5 is designed to read in the data from the metatable, which will make Cell 7 neater. Unfortunately it's not yet functional.

2.4.5 (e) Uncomment the t.writeto(outputcatalog) line in Cell 8 and run Cell 8

For experienced users:

Error Calculations
Weights

3 Populating or Expanding the Meta-table

3.1 Pre-Processing 2.0

In order to perform the empty aperture simulation, you'll need a noise-normalized, object-subtracted version of your image. In the first stage of pre-processing you used **maskitextra.py** to produce a *file_sci_newnorm3.fits* and a *file_sci_mask.fits*.

- (a) Run Source Extractor on newnorm3.fits
- (b) Run quickmask.py on the objsubnewnorm3.fits

Example

General form:
(a) sextractor *file_sci_newnorm3.fits* -c default_objsub_norm.sex -CHECKIMAGE_NAME *file_sci_objsub_newnorm3.fits,file_sci_seg_newnorm3.fits*

(b) python **quickmask.py** *file_sci_objsub_newnorm3.fits file_sci_mask.fits*

Parameters:

- *file_sci_newnorm3.fits*: full name and path of the relevant noise-normalized science image

- default_objsub_norm.sex: a special parameter file that Source Extractor uses to identify objects (sources) that will be subtracted[3]
- *file_sci_objsub_newnorm3.fits*: the name of the output object-subtract, noise-normalized image
- *file_sci_seg_newnorm3.fits*: the name of the output segmentation file associated with the relevant noise-normalized science image.

Example:
(a) sextractor *j020548m5829-f606w_drc_sci_newnorm3.fits* -c default_objsub_norm.sex
-CHECKIMAGE_NAME *j020548m5829_f606w_objsub_newnorm3.fits,j020548m5829_f606w_seg_newnorm3.fits*

(b) python **quickmask.py** *j020548m5829_f606w_objsub_newnorm3.fits j020548m5829-f606w_drc_sci_mask.fits*
Example Output:

- j020548m5829_f606w_objsub_newnorm3.fits
- j020548m5829_f606w_seg_newnorm3.fits
- j020548m5829_f606w_objsub_newnorm3_maskd.fits

3.2 Empty Aperture Simulation

This process, while computationally heavy, is quite easy to run. The execution of python **runEAA.py** takes 10 min-1 hour, as it places 1000 empty apertures 30 times and measures their contents.

- Run the empty aperture simulation:

python **runEAA.py** *file_objsub_newnorm3_maskd.fits filename_osnn3m* pixelscale

- Make a comprehensive .txt file:

ls $PWD/shortfilename_filter_osnn3m.dat > all_shortfilename_aps.txt

Parameters:

- *file_objsub_newnorm3_maskd.fits*: The noise-normalized, object-subtracted, masked image upon which empty aperture simulations will be performed.
- *filename_osnn3m*: A unique name prefix for the output .dat and .reg files
- pixelscale: The pixel scale in arcsec/pixel for the input image. This is used to determine the aperture sizes in pixels for purposes of calculation

Example:
python **runEAA.py** j020548m5829_f606w_objsub_newnorm3_maskd.fits j0205_f606w_osnn3m.dat 0.05
...wait quite some time...
ls $PWD/j0205_f606w_osnn3m.dat >all_j0205_f606w_osnn3m_aps.txt

[3]There are a lot of adjustable parameters in here. default_objsub_norm.sex contains some of the best settings I derived through experimentation, but per cluster you might need to adjust FILTER_NAME for a filter with a gaussian fwhm closer to the FWHM of a typical star: and maybe (rarely?) DETECT_THRESHOLD, DETECT_MINAREA, PHOT_APERTURES, and SEEING_FWHM

3.3 Plotting and Fitting

The jupyter notebook Working_Depth.ipynb is used to analyze the noise characteristics of the available HST images used in the catalog. The values are input into the metatable manually, but calculated herein.

3.3.1 5σ AB Mag Depth Values and Plots

- (a) Update the input files in Cell 2
- (b) Comment, and uncomment, the relevant filenames, origs, hdus, hdrs, holdphots, apers, **for** loops, and plt.plot() parameters in Cell 2 [4]
- (c) Run Cells 1-2

3.3.2 RMS ($\bar{\sigma}$) Values

RMS values are reported in the metatable for a "blue" and "red" band, which fall on either side of the 4000 Åbreak for the cluster's redshift. For the 5 GoGreen clusters, the recommended "blue" and "red" bands are included in the metatable–they are the deepest and nearest bands that bracket the redshifted 4000 Å. You will need to identify these bands to proceed with the Working_Depth.ipynb.

- Execute (a)-(c) as above
- (d) In Cell 3, update the assignment of apers and sigs to aper_r, aper_b, sigs_r, and sigs_b to your identified blue and red bands
- (e) Similarly, in Cell 3 update fits_sci, fits_mask, fits_seg; fits_scib, fits_maskb, fits_segb; and **don't forget the pixel scale values each time you call depthcalc.rmscalc()**
- (f) Run Cell 3. $\bar{\sigma}$ will be printed for each band in the output field below.

3.3.3 Fitting a Polynomial to the Noise Dependence on Linear Size

Again, getting a fit to the dependence of the noise on linear size is quite straightforward:

- Execute (a)-(f)
- Run Cell 4

We expect noise in these images to be correlated somewhat across the image due to ..... (the way CCDs work, mostly). In particular, we expect the noise as a function of linear size to take the following shape:

$$\sigma(N) = c_0 + c_1 N + c_2 N^2 \tag{2}$$

where $N = \sqrt{A}$ is the linear size of apertures with area A. C_0 is set to zero for realism.

The correlation of noise across linear scales within an image is discussed in more detail in, for example, Labbe, Franx, Rudnick, et al (2003, AJ 125 3).

[4] each **for** loop has two options for the infile, depending on whether or not you remembered to include $PWD/ when you piped your .dat to the all_filename_aps.txt file

The parameters derived in Working_Depth.ipynb can be related to the a_i and b_i parameters in that text with the RMS value ($\bar{\sigma}$)as follows:

$$a_i = c_1/\bar{\sigma} \qquad \text{and} \qquad b_i = c_2/\bar{\sigma}$$

A Short Note for Beginners

Registration is the process by which the coordinate systems are aligned between two images of the same field.

Reprojection is a linear transformation of every image such that a given position on the sky will map to the same pixel in each one of your transformed images. Rotation, shift, and scale are applied to each image to map it onto another image with a different set of those values.

As Montage (http://montage.ipac.caltech.edu/docs/algorithms.html) says of reprojection, "The goal is to create an output image which is as close as possible to that which would have been created if the sky had been observed using an instrument with the output image's pixel pattern". This is used when you are making comparisons between multiple wavelength bands (which usually have CCDs with different pixel arrangements).

Finally, convolution is used to make sure that a point source (i.e. a particular star) is translated into the same resultant PSF in the reprojected blue image as it is in the native red image.