Draft version October 6, 2022
Preprint typeset using LaTeX style emulateapj v. 12/16/11

AUTOMATING ASTROMETRY

Lilikoi Latimer

Draft version October 6, 2022

ABSTRACT

We outline code written to make strides towards automating astrometry correction. The code is written in Python and is named astrom_final. The code involves images in FITS format: a reference image with correct astrometry and 'broken' images with astrometry in need of fixing. The code currently relies on two different methods of fixing the astrometry so as to provide a measure of redundancy in case one method fails.

1. PREAMBLE

The rest of this outline was written several years prior, when I was writing the code. I've updated both, and this outline will likely be helpful if you want to delve more into what the code is actually doing, but the writing may be a bit... stilted. The examples in Section 5 will likely be helpful, though.

2. INTRODUCTION

Often when viewing multiple images of the same object taken on different observation runs or at different times, one finds that the images don't quite line up. This can also be a problem when comparing images of the same object from different telescopes. The reason for this is fairly simple - our astrometry is only accurate out to a certain point. Depending on the science goals, this doesn't necessarily matter - for instance if all we want is to get an image and perform data analysis on it, we don't particularly care if the World Coordinate System (WCS) coordinates attached to said image are off by an arcsecond or two, as precisely where our object is in the sky is not what we care about.

However if we have to make a composite of several images - to make a three color image, or when overlaying radio image contours on a visible light image, for instance - we need the coordinates of these images to be accurate and consistent. While this is straightforward to do manually, it is quite tedious and inefficient to do by hand if we have many images to fix.

To this end, this code was written. The code currently works for one reference image and multiple 'broken' images, the latter being the images in need of astrometry correction. Some thought on the part of the user is still required - the tolerance allowed in how much the methods disagree, the rough expected maximum astrometrical correction, and which method to use when both methods agree - but as a whole the code is a significant improvement over manual correction.

Conventions as used in this paper:

- Reference image - the image that is assumed to have correct astrometry.
- Broken image - the image that has astrometry in need of correcting.
- Reference/Broken frame - the pixel coordinates of the respective image when viewed as an array.

This paper is organized as follows: In Section 2 we discuss responsibilities that the user must still take upon themselves regarding the input and output FITS files. In Section 3.1, we outline what the code does and how it works, and in Section 3.2 we go more in depth on how the code operates. In Section 4 we give several examples of the application of this code on images of galaxies, and in Section 5 we provide a summary of how to use the code and possible issues that might spring up. In the Appendix we have several figures regarding the terminal output for the examples in Section 4.

3. DATA

This code uses two folders as input: the first folder containing a single reference image, and the second folder containing the broken images. It may be helpful to compare the reference images and broken images in a different program, such as DS9, before running them through this code so as to make sure that there are indeed sources to find (though I haven't found this necessary in my usage). If the images are of two different parts of the sky, with no common sources between them, this code won't work (or rather it will but it will give meaningless and pointless results). Sometimes files from telescopes can say that a specific object (e.g. a galaxy) is in the frame of the image when it is not - I have had several Hubble Space Telescope (HST) images where that was the case.

This code writes out a file of the form "originalfilename_fixed.fits" if the two methods are in agreement, and will write out files of the form "originalfilename_fixed_mult.fits" and "originalfilename_fixed_corr.fits" if the two methods are in disagreement. This code will also display an error if there already exists a file by that name in the directory it is writing to, which (depending on the save_opt variable) could be the folder containing the broken images.

4. METHOD

4.1. *Broad Overview*

This code reads in a single reference image and any number of broken images. The aim of this code is to use two different methods, multiplication and correlation (detailed later), to correct the astrometry of the broken image. While neither method will work in every single case, the cases in which one method fails usually sees the other method succeed, giving us some measure of redundancy.

We start by reading in the reference image and a broken image and projecting the reference image into the broken image's frame.

For the multiplication method, we multiply the two arrays together and find out the location of the brightest pixel of the resulting array. We draw a box around that location of user-specified side length and find the locations of the respective maxima of the projected reference image and the broken image inside the box, take the difference between those locations, and use that difference as our astrometrical correction.

For the correlation method, we simply auto-correlate the projected reference image array to get a baseline, then cross-correlate the projected reference image array and the broken image array. We take the difference of the two and use that as our astrometrical correction.

We then check whether these corrections agree to within some user-specified tolerance level, and if so, write out the file corresponding to the user-specified default method, either correlation or multiplication. If they don't agree, we write out both methods as separate files so the user can look at them later to decide which to use.

We then iterate this over all the broken files in the specified folder.

4.2. *In Depth*

For maximum detail see commented code - this section will try to explain what's going on at a level of detail in between the broad overview and the code itself.

4.2.1. *Variables to Declare*

The code requires five user inputs:

- *tol* - the tolerance within which the astometrical corrections need to agree (in pixels). The default is 3 - this seems to delineate fairly well between both methods essentially agreeing with minor variations, and the methods majorly disagreeing.
- *sep* - roughly, the maximum separation expected between the broken and reference images (in arcseconds). Basically "what's the most we could be off by". This is what will be used to make the side lengths of the drawn box (technically twice this number).
- *default* - the default method to use when both methods agree; the two options are 'corr' and 'mult' - correlation and multiplication, respectively. Currently set to 'corr', as that seems to be the more accurate method when it is correct.
- *projection* - which projection option to use; options are 'exact' and 'interp'. Exact takes somewhat longer, though the exact time will vary depending on the user's pc. At most exact should take 30 seconds per file in the broken folder.
- *save_opt* - where to put the corrected files; the two options are 'sep' and 'mix'. Sep creates a separate results folder and fills it with the corrected images, and mix puts the corrected images in the broken folder (mixing corrected and uncorrected images). Note that the corrected images will have the updated filenames, regardless.

4.2.2. *The Multiplication Method*

As the astrometry for the broken image is assumed to be fairly close to correct (separation of at most single digit arcseconds), the sources in the broken image array and the projected reference image array tend to overlap somewhat, or even just barely. This makes simply multiplying the arrays together a good idea, because it tends to reduce the effect of noise (unless both images happen to have very loud noise in exactly the same spot) and bring out the intersection of the interesting sources, roughly.

To that end, we multiply the arrays together and find the location of the brightest pixel. If several brightest pixels are found, the code averages over them and then rounds to get a single location. From this we draw a box centered at that location with side lengths of roughly $2*sep$. For instance, if *sep* is two arcseconds, the box will be a square with side lengths of four arcseconds centered on the brightest pixel location from the multiplied array.

We then find the location of the maxima inside this box in the broken image array and projected reference image array, averaging and rounding if necessary. The difference between the locations of the two maxima is our astrometrical correction.

4.2.3. *The Correlation Method*

The projected reference image and the broken image should contain the same source, and by matching these sources we can find the astrometrical correction necessary to fix the broken image. We first auto-correlate the projected reference image array and find the location of the brightest pixel. We then cross-correlate the projected reference image array and the broken image and find the location of the brightest pixel in that array. We take the difference of those two locations and use that as our astrometrical correction.

4.2.4. *Comparing the Methods*

We check whether the difference of our two methods' corrections are within tolerance (*tol*). If they are, we write out the file specified by the *default* variable, and if they aren't we write out both of them so the user can look at them later to decide which to use. Note that the code will throw out an error and stop if there is already a file with the same name as the one we try to write to - see Section 2 for more detail.

4.2.5. *Printing Information*

When the code is done running we print out the list of broken files that had astrometrical corrections in agreement and the correction corresponding to the file that was written out (in pixels). We also print out the list of which files disagreed, and by how much they disagreed. We print out *tol*, *sep*, *default*, and *projection* for convenience. See the Appendix for examples of terminal output.

5. EXAMPLES

We demonstrate this code on images from two different galaxies: Haro 3 and Haro 9. The broken images are all from HST and the reference images are from the Sloan Digital Sky Survey (SDSS).

5.1. *Haro 3*

For Haro 3 we have three HST images to correct. There is one interesting case which shows how the correlation method can fail.

The terminal output for these images is displayed in the Appendix (Figure 3). From this we see that the third image has a large disagreement between the methods. What we have in Figure 1 is the reference image (blue), the multiplication method (red) and the correlation method (green). These pictures are all for a single broken image, the one with the disagreement. This is an example of when the correlation method completely fails. In Figure 1a we see the major difference between the multiplication method (red) and the correlation method (green). While the multiplication method clearly agrees with the source we want (Figure 1b), the correlation method has gotten sidetracked (Figure 1c). Apparently there was an additional source that was brighter than the source that we were interested in, and when we found the maximum from the cross-correlation we inadvertently picked out the wrong source.

5.2. *Haro 9*

The second example is matching the astrometry for the galaxy Haro 9: we have six HST images (broken images) and one SDSS image (reference image). Here we have several cases where the correlation method succeeds while the multiplication method fails - see terminal output in the Appendix (Figure 4).

Figure 2 shows one such case - multiplication method in red, correlation method in green, reference image in blue. We can see that the correlation method has succesfully identified the source we want to match, while the multiplication method has gotten sidetracked, most likely due to the large amount of noise in the HST image.

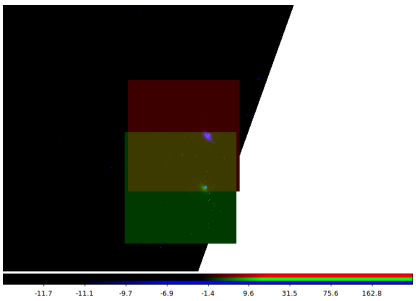
6. CONCLUSIONS AND DISCUSSION

In summary, when running this code the user should:

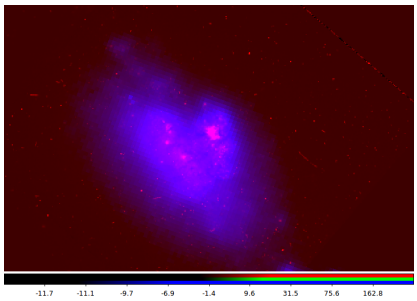
- Check that there doesn't already exist a file with the same name as the broken image with "fixed","fixed_mult", or "fixed_corr" appended to it (e.g. if the broken file is "originalfilename.fits" then there shouldn't already exist a file called "originalfilename_fixed.fits"). Generally this will only be an issue if rerunning the code on the same images.
- Check that the tolerance *tol* is the desired size (default is 3 pixels).
- Check that the rough expected maximum astrometrical separation between images is the desired size (default is 2 arcseconds).
- Check that the default method used to calculate the astrometrical correction when both methods agree is as desired (default is correlation method).

Additionally, once the code has run and output a fixed file, the user should check (on software appropriate for viewing FITS images e.g. DS9) that the fixed file is indeed fixed. Note that generally, in my experience, this code will get the astrometry close but not perfect, so checking manually and perhaps touching it up a bit may be necessary.

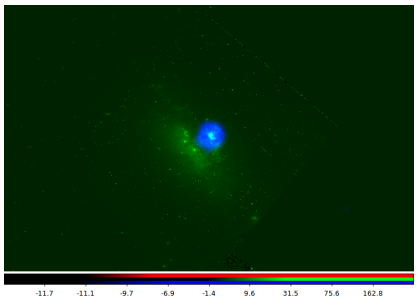
APPENDIX



(a) Zoomed-out view of the multiplication corrected image(red), the correlation corrected image (green), and the SDSS reference image(blue) in DS9.



(b) Zoomed-in image of the multiplication corrected image (red) and the SDSS reference image (blue). Note the good agreement between images - the astrometry looks correct.



(c) Zoomed-in image of the correlation corrected image (green) and the SDSS reference image (blue). Note how the correlation image's source lies directly on a very bright source in the reference image.

Figure 1. Overall and close-up views of the corrected images and the reference image for Haro 3 in DS9.

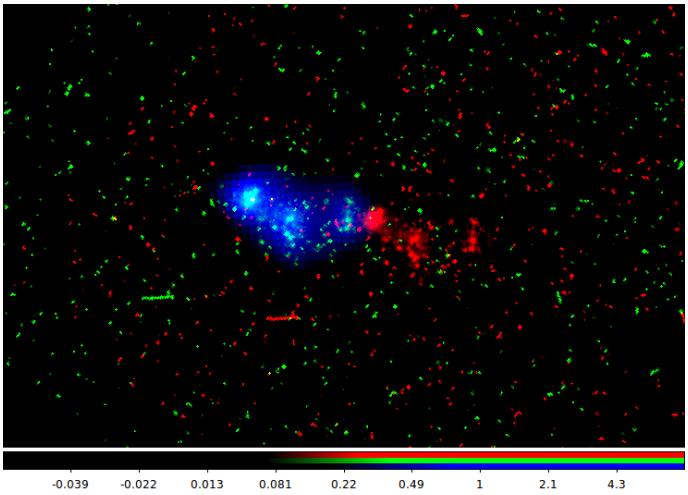


Figure 2. Multiplication corrected image (red), correlation correction image (green), and reference image (blue) for Haro 9. Displayed in DS9.

```
                                                    $ ./astrom_final.py
Reference filepath of:                                                /reference/
Broken filepath of:                                                   /broken/
Tolerance of 3 pixels
Rough separation of 2 arcseconds
Default method: corr
Projection method: exact
(1/3) Fixing u2e67r01t_drz.fits ...
      Mult. correction of [-9, -2] pixels.
      Corr. correction of [-8, -2] pixels.
      Difference is: [1 0] pixels.
(2/3) Fixing hst_05479_7r_wfpc2_f606w_pc_drz.fits ...
      Mult. correction of [-1, 1] pixels.
      Corr. correction of [1, 0] pixels.
      Difference is: [2 1] pixels.
(3/3) Fixing hst_05479_7r_wfpc2_f606w_wf_drz.fits ...
      Mult. correction of [-1, 0] pixels.
      Corr. correction of [-1008, -65] pixels.
      Difference is: [1007   65] pixels.
            Difference outside tolerance bounds!
            Saving both mult. and corr. fixed files.
------------------------
Tolerance of 3 pixels
Rough separation of 2 arcseconds
Default method: corr
Projection method: exact
Agreements: (2/3)
      u2e67r01t_drz.fits - change of [-8, -2]
      hst_05479_7r_wfpc2_f606w_pc_drz.fits - change of [1, 0]
Disagreements: (1/3)
      hst_05479_7r_wfpc2_f606w_wf_drz.fits - diff. of [1007   65]
Done.
                                                    $
```

Figure 3. Terminal output for correcting three HST images of Haro 3. Note how we have one disagreement of over 1000 pixels.

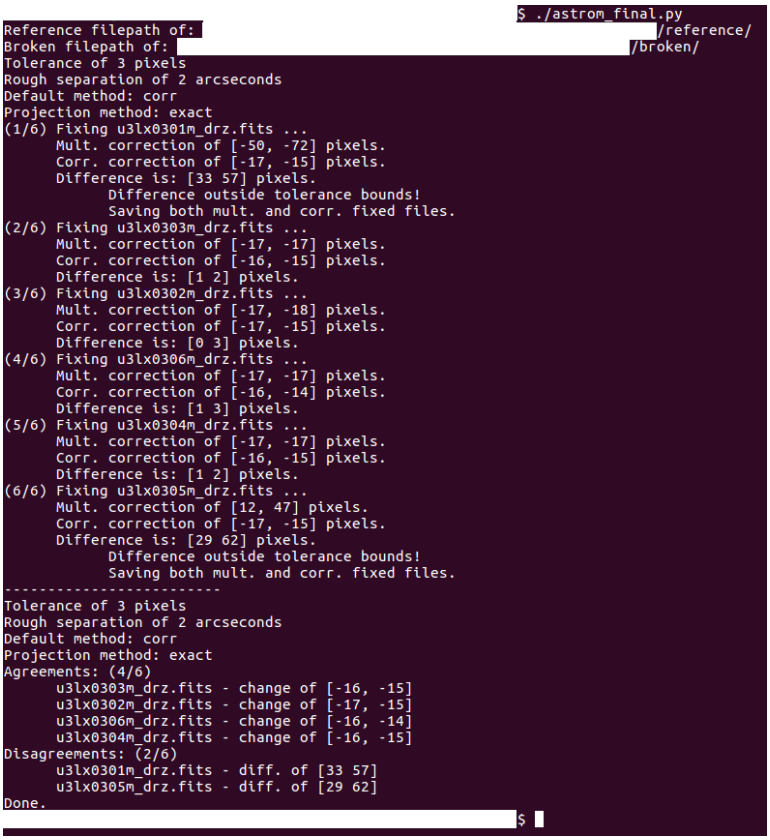


Figure 4. Terminal output for correcting six HST images of Haro 9. Note how we have two disagreements.