FITSH python wrapper instruction

Chelsea Huang, 2016/09/29, fitsh version 0.9.2, wrapper version 0.0.3

General Feature description of FITSH The FITSH package is an open-source software collection related to astronomical image and data processing developed by A. Pal. Please refer to the website fitsh.net for downloading, installing and detailed instructions regarding FITSH.

This python wrapper is an attempt to demonstrate a pipeline structure to do simple aperture photometry for simulated TESS frames based on FITSH in python. We only provide limited flexibility at the user end regarding the command line options of FITSH. Further development is required to achieve optimal photometry precision and reduction speed.

To get the wrapper:

$ cp /home/chelseahuang/chscript.tgz $yourhomedirectory

$ cd $yourhomedirectory

$ tar -xzf chscript.tgz

Key FITSH tasks used

For the simplest photometry reduction, there are three key tasks:

(1) Initial source extraction with fistar.

(2) Source cross match with external catalog using grmatch.

(3) Photometry with fiphot.

To get familiar with these commands, for example fistar, you can do either of the below:

$ fistar --long-help

or

$ python run_fistar.py --long-help

Get Around the wrapper:

The key files you interact with are Frame.py and example.cfg.

Frame.py is the executable, and example.cfg is a configuration file.

(1) test if everything works on your platform:

$ python Frame.py --dry print out a list of fitsh command the code would execute if you try to do it on an example frame. In this case, "FITS/simulated_18h00m00s+66d33m39s_ccd1_000000.fits".

$ python Frame.py --test will execute the above commands internally, this will take less than 4 seconds.

After the run, you should see the files under the FITS directory:

FITS/simulated_18h00m00s+66d33m39s_ccd1_000000.fistar (a list of stars fistar detected)

FITS/simulated_18h00m00s+66d33m39s_ccd1_000000.trans (the transformation file between the projected catalog and the detected star centroids)

FITS/simulated_18h00m00s+66d33m39s_ccd1_000000.xyls (the projection of the external catalog based on the transformation)

FITS/simulated_18h00m00s+66d33m39s_ccd1_000000.fiphot (the final photometry file)

(2) Examine the output:

$ cd FITS

$ ds9 simulated_18h00m00s+66d33m39s_ccd1_000000.fits& (this will open up a ds9 window display the simulated image.)

You can use zscale to display the image under a different scale compare to the default one so that you see most of the stars.

$ tvmark simulated_18h00m00s+66d33m39s_ccd1_000000.fistar -x 2,3 -s 3 -c red (this command will draw red circles with radius of 3 pixels at the positions of the stars fistar identify on top of the image.)

$ cat FITS/simulated_18h00m00s+66d33m39s_ccd1_000000.trans

The above commands display the content of the "trans" file, which shows summary information of the astrometry result. The few numbers to pay attention to in the summary file are: the polynomial order we used, the percentage of stars got matched between the fistar file and the catalog, and the residual reported in unit of pixel.

$ tvmark simulated_18h00m00s+66d33m39s_ccd1_000000.xyls -x 2,3 -s 2 -c green -l 1 -a green (this command will draw green circles with radius of 2 pixels at the positions of the stars projected from the catalog, the numbers displayed below the circle is the UCAC4 identification number of the star.)
or you can use

$ tvmark simulated_18h00m00s+66d33m39s_ccd1_000000.xyls -x 2,3 -s 2 -c green -l 4 -a green
to display the V magnitude of the stars instead.

At any stage, you can use:

$ tvmark -d to clean up all the things previously draw.

Look at the top few lines to understand the structure of the photometry file:

$ head simulated_18h00m00s+66d33m39s_ccd1_000000.fiphot

In the default run, column 5 is the measured magnitude of the star, you can do

$ head simulated_18h00m00s+66d33m39s_ccd1_000000.xyls and compare with the catalog magnitude in column 4.

Discussion: why are the two magnitude not exactly the same?

The same process can also be run with the following commands:

$ python Frame.py -i simulated_18h00m00s+66d33m39s_ccd1_000000.fits

(3) run it on a list of frames:

$ python Frame.py -l fits.ls

Currently the wrapper allows you to do photometry on a list of frames without parallelize the process. If you examine fits.ls, you can see the basename of 10 frames. They are all under the directory FITS. This will take about 30 seconds to run, and for each frame, four additional file will be created. You can compare the magnitude measurement of the same star manually for different frames. It is also useful to check how the residuals of the astrometry solution change over different frames.

(4) create light curves.

$ bash lcgen.sh

The above command will generate light curves for all the stars in the fiphot files under the LC directory. Currently they would only have 10 point each, you are welcome to plot a few of them. The sequence of the columns are exactly the same as the ones in the fiphot files.

Understand the configuration files

$ cp example.cfg test.cfg

and edit test.cfg. You can also copy the end files of the test run to a different place so later on we can compare the result.

There are three sections in the configure file that tunes the parameter setting of the three key tasks.

(0) Section Setup

inpath defines the directory that the fits files are stored, and the intermediate outputs are saved (might want to use different directories in the future).

(1) Section Fistar

threshold defines the detection peak threshold of fistar, in ADUs.

Change it into a different number, such as 300,000 instead, or 10,000,000, and rerun the test run with

$ python Frame.py --test -c test.cfg

use tvmark to examine the difference in how many stars were identified.

You would also expect some change in the trans file.

verbose control how many information the code print out.

(2) Section Fiphot

magtoflux defines how fiphot convert flux into magnitude. The first number represent a magnitude, and the second number represent to corresponding flux.

skyfit_sigma, **skyfit_niter** and **disjoint_radius** changes how fiphot estimates the background level.

apertures determine what kind of circular apertures to use for the photometry. Three numbers decide one aperture, the first number is the radius of the aperture, the second number is the radius of inner background annulus, and the third number is the width of the background annulus. Try to do a list of apertures, such as: '1.5:4.0:3.0,2.5:4.0:3.0,3.5:5.0:3.0,4.5:6.0:3.0' to see the change in fiphot file, and compare the magnitudes measured in different apertures for stars with different brightness.

(3) Section Grmatch

order defines the order of the polynomial transformation. Try to change this to other numbers and observe the change in the trans file.

maxdistance The maximal accepted distance between the matched points in the coordinate frame of the input coordinate list (and not in the coordinate frame of the reference coordinate list). Try to change this to other numbers, such as 0.1 or 10 and observe the change in the trans file.

unitarity A match is considered as a good match if the unitarity of the transformation is less than the unitarity U specified by the unitarity=U directive. The unitarity of a geometrical transformation measures how it differs from the closest transformation which is affine and a combination of dilation, rotation and shift. For such a transformation the unitarity is 0 and if the second-order terms in a transformation distort a such unitary transformation, the unitarity will have the same magnitude like the magnitude of this second-order effect. Try to change it to other numbers and observe the change in trans files.

ra0, dec0 The center coordinates of the camera (optical axis). If not given, it use the example field's coordinates as default. But if you are trying to solve a different field using the Frame.py routine, not change these values will break the astrometry part. See the **Reduce a field** section if you don't know these values. When reducing a field from Field.py routine, these coordinate settings will be ignored.

ra,dec This is the center coordinates of the CCD.

catfile The name of the catalog file, if not exists, the code should try to query a new catalog online to replace it instead.

method The method to query the catalog file if it does not exit. If not given, use astroquery, the other option right now is "lfov-usno". I would recommend to use astroquery for now, because it is hard wired to querry for UCAC4, which has a better centroid position compare to usno. However, in the long run, we would like to use in house program "lfov-gaia".

Reduce a field

This is only available from wrapper version 0.0.3. It enables the user to reduce a field of n ccds without given specific information of the field.

Field.py is the excutable for this purpose.

Prerequirement:

(1) Astrometry: this routine currently require the installation of astrometry.net package. Astrometry.net is used to give a first guess of the astrometry solutions of frames since currently the ccd centre RA and DEC coordinates are not in the FITS header. This dependence may be removed later on if we decided to have relevant informations in the FITS header.

You can download this from the site above, or get the package from:

$ /home/chelseahuang/software .

You may also want to install the cfitsio package, which you can download online or from my software directory.

To ensure astrometry.net run properly, refer to README for detailed instructions (I would recommand a global installation on tessellate).

I also have the index files astrometry.net use at

$ /data/chelseahuang/CAT/INDEX/ .

After install the astrometry.net software, you would need to edit your $ astrometry.cfg file under the $ etc

directory of your install directory to have the line:

$ add_path /data/chelseahuang/CAT/INDEX

(2) Data structure:

Under the base directory where you put the frames, the frames should be divided into seperate directories according to their ccds. If you put $nccds = 4$ in the configure file for this wrapper, it expect to find four directories called ccd1, ccd2, ccd3 and ccd4 inside the base directory.

The frames from the same field is expected to have the same basename, have the same observation lenth, and start from the 0th frame, while the entire file name looks similar to this one:

$basedir/ccd1/simulated_18h00m00s+66d33m39s_ccd1_000000.fits

The current file structure from simulating a TESS field using SPyFFI autometically satisfy this requirement.

You can look into my directory for an example of reducing 2 ccds from the same field.

$ ls /home/chelseahuang/chscript/FITS/ .

Excute Field.py

First try the test example to make sure everything is correctly installed, and works.

If you copied my entired directory,

$ python Field.py –test -c example.cfg

This will reduce the two frames to the final photometry step under the ccd1 and ccd2 directory in side FITS directory. You can find the details of each steps in the previous instructions on the module Frames.py

The first time you run it, it will take a bit of time to download the catalogs and run astrometry.net. This step is only needed once per field, so the overall speed is about 25s per frame (10 hours per CCD per core for a 30 day observation).

To reduce your own field:

$ cp example.cfg $yourconfigurefile

Try to edit your own configure file:

inpath defines the base directory.

basename defines how the name of the FITS files are constructed.

nccds defines the number of ccds to reduce in the field.

ncadences defines the total number of observations with a ccd. *Make sure all the CCDs have the same number of observations, and the observations start with frame 0.*

$ python Field.py –test -c $yourconfigurefile