# wcsfit

October 28, 2016

This document describes `wcsfit`, the software used to perform astrometric calibration on OU-VIS images.

The general philosophy is that the geometrical distortion due to the optics and the focal plane array (including CCD rotation and tiling) is fixed and can be modelled using a high enough dense field. Once this model is determined with suitable calibration observations it can be applied to any observation; the only additional step required is then to solve for a the rotation, scale and translation due to the inacuracy of the initial astrometry.

This modules takes as input a Euclid OU-VIS MEF file containing CCD images and an astrometric reference catalogue. The code works by extracting a catalogue from the images and matching it to the reference catalogue. The astrometric fitting is an iterative process where 1) the offset between computed and reference coordinates is corrected, 2) a transformation including scaling, rotation and offset is computed and corrected and 3) optionally the astrometric model is fitted and refined.

```
In [1]: import wcsfit
        'wcsfit version {}'.format(wcsfit.__version__)

Out[1]: 'wcsfit version 201610'
```

## 1 Requirements

The following Python modules are required:

- numpy
- scipy
- astropy (compiled with wcslib >= 5.9)
- configobj

The following TPS are required:

- casutools/imcore >= 1.0.30

(note that casutools/wcsfit is not longer a requirement)

## 2 Code repository

The code resides in the ESA svn repository. To get a copy of the current version:

```
svn co http://euclid.esac.esa.int/svn/EC/SGS/OU/VIS/ \
4-3-01-3500_AstrometricCalibration/trunk/AstrometricCalibration
```

The `wcsfit.py` file is in the directory `prototype` together with `wcsfit.conf` containing the calculated distortion model and several documentaion files.

## 3 Command line usage

The following inputs are needed:

- image.fits: a MEF containing 1 to 36 CCD images
- image.weight.fits: a MEF containing an Euclid weight map, same number of extensions as the input image.
- refcat.fits: reference star catalogue
- wcsfit.conf: a file containing the astrometric model to use

The command line for fitting the astrometry is:

```
python wcsfit.py image.fits refcat.fits --weight image.weight.fits
--wcsconf wcsfit.conf --debug
```

The weight map is optional. Other command line switches can be found typing `python wcsfit.py --help` or reading the code documentation using `pydoc wcsfit`.

The rest of the document describes more in detail the code and the different functions available.

## 4 Prepare input image

The output of the OU-VIS simulator is a MEF file containing 144 quadrants. We first need to create a MEF file containing one CCD in each extension. During the pipeline running this is done before this module is called but here we have a convenience function to do this ourselves. For this we use `merge_quadrants` which will create another file appending `_ccd` to the file name.

We set the log level to DEBUG and start merging the quadrants.

```
In [2]: wcsfit.logger.setLevel(wcsfit.logging.DEBUG)

In [3]: wcsfit.merge_quadrants('simone.fits')

wcsfit:INFO:merging quadrants into CCD image (simone.fits)
wcsfit:INFO:output image already exists simone_ccd.fits
```

We can then start to work with the CCD MEF image file. For this doc we extract a single CCD image from the full mosaic. We open it in update mode but we can also open readonly and write the result to another file at the end.

```
In [4]: wcsfit.extract_extension('simone_ccd.fits', 'simone_ccd_2.fits', 2, clobber

In [5]: img = wcsfit.CCDImage('simone_ccd_2.fits', mode='update')
```

## 5 Initialize headers and WCS information

In order to start working with the image we have to inialize the headers, i.e. write default values for a few keywords. We move the reference point of each CCD to its center, setting CRPIX1, CRPIX2 = NAXIS1/2, NAXIS2/2 and calculating the value of CRVAL1, CRVAL2 using the original WCS information.

We also set the projection to be TPV and define an initial set of 3 degree polynomial coefficients (see below). The projection written in this step is basically a TAN projection described as TPV with no distortion (i.e. all PV zero except PV1_1 and PV2_1 which are 1).

```
In [6]: img.init_astrometry()

wcsfit:INFO:writing initial astrometric solution (ccd)
```

The package includes an astrometric model derived from a (simulated) observation of a high dense field, containing a uniform distribution of around 600 stars per CCD.

This model can be added to the headers using the command below. If we want to derive the model from scratch from a high enough dense field we can skip this command.

```
In [7]: img.add_astrometric_distortion('wcsfit.conf')

wcsfit:INFO:adding distortion coefficients
```

## 6 Object detection

Object detection is performed using `imcore` from `casutools`. In order to work, `imcore` needs to be in the `PATH` or in the `IMCORE` environmental variable.

```
In [8]: wcsfit.IMCORE='/soft/imcore'
        wcsfit.check_exe_in_path()

wcsfit:INFO:IMCORE: /soft/imcore
```

When running imcore a catalogue FITS file will be created where each extension contains the objects detected in each of the extension images.

```
In [9]: _ = img.find_stars()

wcsfit:INFO:detecting objects on images
```

TODO: Add a note on using weight maps.

## 7 Fitting

The fitting is done iteratively. Each iteration contains a coordinate offset, an affine transformation and an optional model distortion fit. These steps are iterated until convergence.

### 7.1 Coordinate offset

The first step is to compute and correct for any offset between the input catalogue coordinates and the reference catalogue using the original WCS in the header. An optimal match radius between both catalogues is computed using their nearest neighbour distribution. The CRVAL1, CRVAL2 values are updated accordingly.

### 7.2 Rotation, scale and translation

This step uses the $x, y$ coordinates of the detected objects and the standard coordinates $\xi, \eta$ of the reference stars to construct a set of equations in the form

$$\begin{aligned} \xi - \xi_0 &= cd_{11} \times (x - x_0) + cd_{12} \times (y - y_0) \\ \eta - \eta_0 &= cd_{21} \times (x - x_0) + cd_{22} \times (y - y_0) \end{aligned}$$

Where $cd_{ij}$ is the CD matrix, $x_0, y_0$ is the reference pixel and $\xi_0, \eta_0$ account for an offset in standard coordinates. In matrix notation $TX = Y$ where $T$ is the transformation matrix:

$$\begin{bmatrix} cd_{11} & cd_{12} & \xi_0 \\ cd_{21} & cd_{22} & \eta_0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x - x_0 \\ y - y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} \xi \\ \eta \\ 1 \end{bmatrix}$$

Note that $cd_{11} = -cd_{22}$ and $cd_{12} = cd_{21}$ giving an extra simplication. This set of equations can be numerically solved using least squares minimization (`scipy.optimize.least_squares`) or analitically solved. `wcsfit` incorporates both methods and give the same result. Note that `least_squares` optimizes the sum of the squares of the residuals. Other optimizations (e.g. the sum of the absolute values of the residuals) can be implemented in next versions.

### 7.3 Model distortion

wcsfit uses a precomputed distortion model which is written in the headers. This distortion model is derived from a calibration field with enough stars to provide a good fit. The model can be refined or created from scratch using the `--fitpv` flag.

$$\begin{aligned} \xi &= cd_{11} \times (x - x_0) + cd_{12} \times (y - y_0) \\ \eta &= cd_{21} \times (x - x_0) + cd_{22} \times (y - y_0) \end{aligned}$$

$$\begin{aligned} \xi' &= pv_{10} + pv_{11} \times \xi + pv_{12} \times \eta + pv_{13} \times r + pv_{14} \times \xi^2 + pv_{15} \times \xi \times \eta + pv_{16} \times \eta^2 + \dots \\ \eta' &= pv_{20} + pv_{21} \times \xi + pv_{22} \times \eta + pv_{23} \times r + pv_{24} \times \eta^2 + pv_{25} \times \eta \times \xi + pv_{26} \times \xi^2 + \dots \end{aligned}$$

where $r = \sqrt{\xi^2 + \eta^2}$. This is a set of linear equations solved by least squares.

### 7.4 Pipeline mode

In this case we have added the pre-computed distortion model to the headers using `add_astrometric_distortion()` as described above and simply run:

```
In [10]: img.wcsfit(refcat='grid_v2.fits')
```

```
wcsfit:INFO:fitting astrometry
wcsfit:INFO:First pass offset dRA: -5.67966216852, dDEC: -5.67029141652, RMS: 0.743
wcsfit:DEBUG:Iter 0 - r: 2.75714549424e-05 s: 8.10175760022e-10, tx, ty: 2.72031395
wcsfit:DEBUG:Iter 0 - RMS: 0.00190770364134 Difference: 0.741865166353
wcsfit:DEBUG:Iter 1 - r: 2.75714928009e-05 s: 8.26220143843e-10, tx, ty: 4.53246609
wcsfit:DEBUG:Iter 1 - RMS: 0.00107167442582 Difference: 0.000967534541076
wcsfit:INFO:1 CCDID: 1-5 Offset: 0.00279399388659 RMS: 0.00108224285873
```

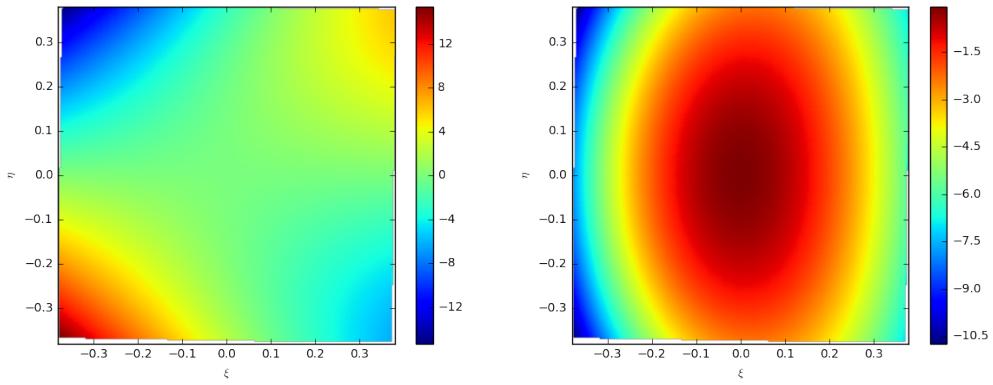
This command simply calculates the affine transformation described above and updates the headers. This is iterated until convergence. It outputs several messages about the fitting procedure.

Summary of results, printing for each CCD the number of reference stars used in the fit and the WCS rms in arcsec.

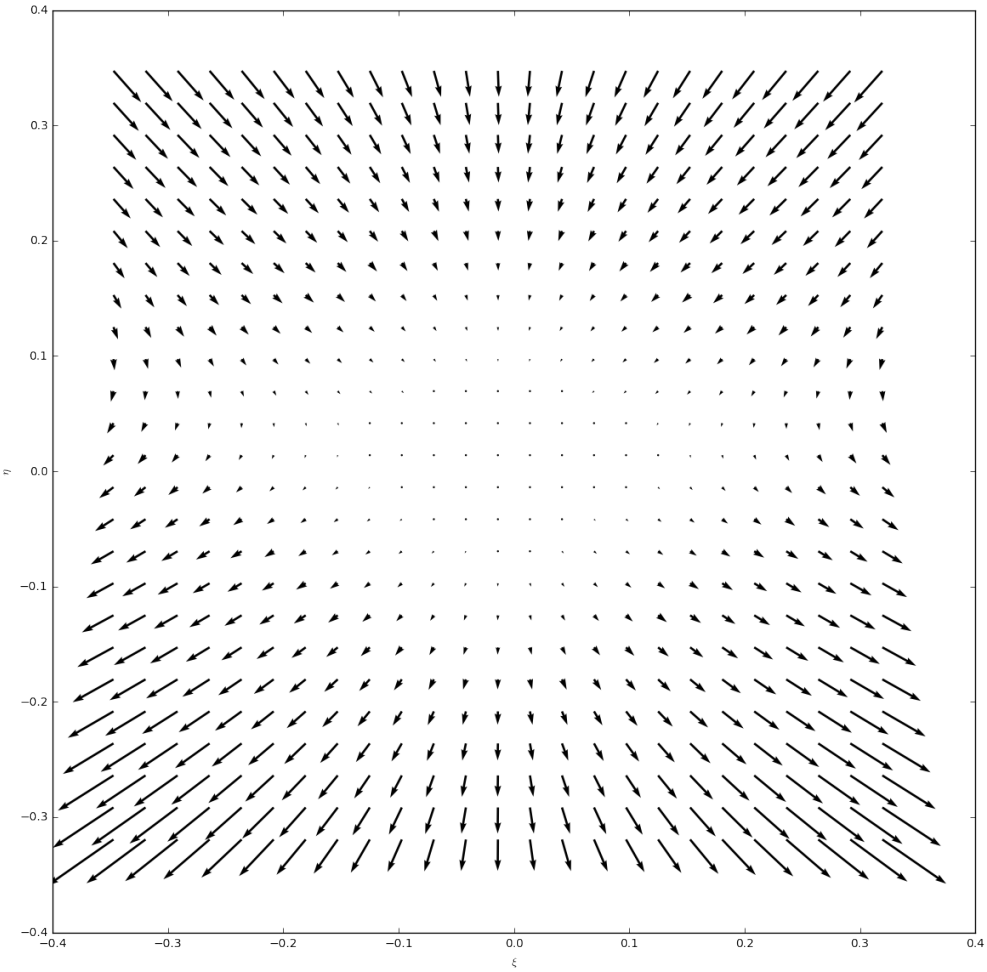
```
In [11]: wcsfit.print_summary('simone_ccd_2.fits')

wcsfit:INFO:simone_ccd_2.fits[1] CCDID: 1-5 NUMBRMS: 535 STDCRMS: 0.001082
```

## 8 Distortion plots

The following figures show the distortion in $\xi$ and $\eta$ for the full FPA.




And the quiver diagram:
These coincide very well with the plots showing the input distortion.


0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
η
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
ξ