Documentation for `WCSFit` Code

G. M. Bernstein

Dept. of Physics & Astronomy, University of Pennsylvania

`garyb@physics.upenn.edu`

13 June 2013

1. Concepts

`WCSFit` adjusts the parameters of astrometric maps to minimize the disagreements between sky positions derived from different exposures of the same objects. The list of matched detections is taken from the `WCSFoF` output file. This file contains links to the original catalog files, from which further information on each detection can be extracted.

- *Detection:* An observation of a sky object, with a position and a position uncertainty given in pixel units.
- *Match:* A group of detections that are expected to represent the same sky object and hence have the same coordiantes after mapping to the sky.
- *Field:* The sky is divided into fields. Each detection is assigned to one field; coordinate matching is done in the gnomonic projection about the center of the field. A list of fields and their centers is provided in the input file from `WCSFoF`.
- *Extension or Catalog:* An input catalog of detections. Catalogs are read from FITS binary tables (probably either generated by SExtractor or encapsulating a reference catalog). Each FITS bintable extension is considered a distinct input catalog. Each extension is assigned to one field, and must have some input world coordinate system (WCS) map into celestial coordinates. An initial WCS map is given in the input file; `WCSFit` will refine the parameters of this or some other functional form.
- *Exposure:* Each extension is assumed to come from one exposure of the sky (a reference catalog would be considered a single exposure). All extensions from a given exposure must be in the same field.
- *Instrument:* Each exposure is assumed to have been taken with a single instrument. The astrometric solution will have a component that is fixed for the instrument, followed by a remapping particular to each exposure. We would want to have a different instrument for each filter of a given camera. We may wish to assign different instruments for different runs of the same camera if we suspect the alignment changed between runs.

- *Device:* An instrument can have multiple devices (*e.g. CCDs*). Each device has its own pixel coordinate system and a distinct WCS. It is assumed that every extension (input catalog) is from a single device of a single exposure with a single instrument.

- *Reference* and *Tag* catalogs: An input catalog (extension) is a *reference* if its position information is considered known and included in the χ^2 of the global fitting, but there is no re-fitting of the WCS to be done for this exposure. A *tag* catalog contains information (such as color) that we want associated with a match, but the object positions are not used at all in the fitting process. Reference and tag extensions are assigned special instrument ID's.

- *Clipping:* The program alternates rounds of χ^2 minimization with outlier rejection. Objects more than some multiple of their uncertainty away from the mean sky position of its matched detections are discarded; we expect at least a few percent of object detections to be outliers because of cosmic rays and detector defects, proper motion (e.g. minor planets!), binaries, etc.

- *Reserved matches:* One can request that a fraction of input matches be withheld from the χ^2 calculation used to optimize the WCS. The statistics of the reserved objects are calculated at the end of `WCSFit` to allow for validation of the fit.

Each extension has a map from pixel to sky coordinates composed of two parts: first, an *instrument solution*, which gives for each device of an instrument a map from pixel coordinates to an intermediate system nominally centered on the optic axis. This is then compounded with an *exposure solution* which maps from this intermediate system to a gnomonic projection on the sky about the central coordinates for the exposure. These sky coordinates are reprojected into a common gnomonic projection about the field center for purposes of comparing sky positions of detections within a match.

There is a flexible grammar for specifying the functional form of the instrument solutions. The exposure solutions are polynomials of a chosen degree. The `WCSFit` configuration parameters allow selected parameters of the WCS maps to be fixed at input values instead of being optimized.

- *Canonical exposure:* there is often a degeneracy between the exposure solution and the instrument solutions. To break this degeneracy, each instrument must have a canonical exposure for which we define the exposure solution to be the identity map.

2. Structures

2.1. `Detection`

A `Detection` is a single measurement of an object position (defined in *Match.h*).

- `catalogNumber` identifies which input catalog this came from.
- `objectNumber` identifies the object uniquely in the input catalog.
- `xpix, ypix, sigmaPix` give the detected position and the uncertainty in each coordinate (assuming a circular error region).
- `xw,yw` are the position in the gnomonic projection of the field. These are derived quantities, recalculated on each iteration of the fit.
- `clipsqx, clipsqy` are the inverse squares of the propagation of the `sigmaPix` error through the WCS map. They are used to determine whether the `Detection` is to be clipped as an outlier.
- `wtx, wty` are the weights of this `Detection` in the χ^2 calculation. They are nominally $1/\sigma^2$ in the world coordinate system, but an exposure can have its weights further scaled by a designated quantity. Weights of zero indicate that the `Detection` does not ever contribute to χ^2 (such as for a tag).
- `itsMatch` is a pointer to the `Match` (if any) containing this `Detection`
- `map` is a pointer to the WCS function used to map this `Detection` into the field's world coordinate system.
- `isClipped` is set to `true` if this `Detection` is to be ignored in the calculation of χ^2.

2.2. `Match`

This class contains a list of detections all presumed to have the same true sky coordinates (defined in *Match.h*). An `iterator` to this list is defined as well as the `begin()` and `end()` functions needed for iteration over the list. A `Match` is constructed with a pointer to its first `Detection`. Whenever `Detection`s are added to or removed from a `Match`, the `Detection.itsMatch` member is updated. Adding a `Detection` resets `isClipped=false`.

Member functions that maintain the membership list and conditions are:

- `add(Detection* e), remove(Detection* e)` add or remove a chose detection from the `Match`.
- `erase(iterator i)` removes a `Detection` referenced by the iterator from the `Match`, optionally deleting the detection at the same time. Returns an iterator to the next object on the list after the erasure.
- `clear()` removes all `Detection`s from the `Match`, optionally deleting them.

- `size(), fitSize()` return the number of `Detections` in the `Match` and the number that will contribute to χ^2, respectively.
- `setReserved(), getReserved()` mark and report whether this `Match` should be reserved from the fitting process.

The member functions below execute the mapping and fitting functionality for this `Match`.

- `remap()` recalculates the $(\texttt{xpix}, \texttt{ypix}) \rightarrow (\texttt{xw}, \texttt{yw})$ mapping for each `Detection`, using whatever parameters are currently held by the WCS classes.
- `centroid(double& x, double& y, double& wtx, double& wty)` reports the centroid and total weight for all the unclipped members i of the match.

$$\texttt{wtx} = \sum_i \texttt{wtx}_i \tag{1}$$

$$\texttt{x} = \frac{\sum_i \texttt{xw}_i \cdot \texttt{wtx}_i}{\texttt{wtx}}. \tag{2}$$

- `chisq(int& dof, double& maxDeviateSq)` returns the χ^2 contribution of the unclipped `Detections`. It also updates the count of degrees of freedom `dof` and the most significant deviation `maxDeviateSq`. If the number of unclipped `Detections` contributing the χ^2 is ≤ 1, this routine returns zero and leaves its arguments unchanged, as there are no position deviations from the mean. The return value is

$$\chi^2 = \sum_i \left[(\texttt{xw}_i - \texttt{x})^2 \cdot \texttt{wtx} + (\texttt{yw}_i - \texttt{y})^2 \cdot \texttt{wty} \right]. \tag{3}$$

  The input value of `dof` is incremented by $2(N_{\text{fit}} - 1)$, with N_{fit} the number of `Detections` contributing to χ^2. The value of `maxDeviateSq` is set to the maximum of its input value and the largest value of the summand in (3).

- `accumulateChisq()` is the primary method used for fitting WCS models to the data. The method does the following:
  - If there are $N_{\text{fit}} \leq 1$ unclipped detections to be fit, returns 0 and does not alter its arguments. Otherwise it will return the number $2(N_{\text{fit}} - 1)$ of degrees of freedom, after updated its arguments as follows.
  - The maps to $\texttt{xw}_i, \texttt{yw}_i$ for each detection are recalculated using the current WCS parameter set $\mathbf{p}$, as well as the derivatives $\partial\texttt{xw}_i/\partial\mathbf{p}, \partial\texttt{yw}_i/\partial\mathbf{p}$.
  - The input scalar `chisq` is augmented by the χ^2 value in (3).
  - The input vector `beta` is augmented as

$$\texttt{beta}_i - = \sum_j \left[\frac{\partial(\texttt{xw}_j - \texttt{x})}{\partial p_i} (\texttt{xw}_j - \texttt{x})\, \texttt{wtx}_j + \frac{\partial(\texttt{yw}_j - \texttt{y})}{\partial p_i} \left(\texttt{yw}_j - \texttt{y}\right) \texttt{wty}_j \right] \tag{4}$$

– The input symmetric matrix `alpha` is augmented as

$$\texttt{alpha}_{ij} += \sum_k \left[\frac{\partial(\texttt{xw}_k - \texttt{x})}{\partial p_i} \frac{\partial(\texttt{xw}_k - \texttt{x})}{\partial p_j} \texttt{wtx}_k + \frac{\partial(\texttt{yw}_k - \texttt{y})}{\partial p_i} \frac{\partial(\texttt{yw}_k - \texttt{y})}{\partial p_j} \texttt{wty}_k \right] \tag{5}$$

Note that this updating of the α matrix is accomplished in a special `AlphaUpdater` class, which skips all the terms in the matrix that are unaffected by this `Match` and also allows multithreaded increments to the matrix. The quantities α and β are those needed to solve the linearized χ^2 minimization through the normal equation.

- `sigmaClip(double sigThresh)` executes one round of sigma clipping on the `Match`. After recalculating the centroid `x,y` of the unclipped `Detection`s, we find

$$\texttt{devSq} \equiv \max_i \left[(\texttt{xw}_i - \texttt{x})^2 \cdot \texttt{clipsqx} + (\texttt{yw}_i - \texttt{y})^2 \cdot \texttt{clipsqy} \right]. \tag{6}$$

If $\texttt{devSq} > \texttt{sigThresh}^2$, then the `Detection` with the maximum deviation is marked as clipped and the method returns `true`. Optionally this also triggers deletion of the clipped `Detection`.

- `clipAll()` marks all `Detection`s of the `Match` as clipped.

2.3. Instrument

An `Instrument` has a `name` and has a vector of the `Device`s that make it up. The class is a `NameIndex` that allows the devices to be indexed either by their numbers or names. The `addDevice()` call adds devices to the `Instrument`. The `mapNames` string vector holds the name of the instrument map portion of the WCS for each device—this name is a key for retrieving the map from a `PixelMapCollection` (see the *PixelMap* documentation). For each device we also keep track of the bounds of its pixel space (`domains` vector).

2.4. Exposure

The `Exposure` class describes a single exposure. It has a name. The `field` and `instrument` integers are indices saying which field and instrument were used for this exposure. The coordinate map for this exposure is accessed from the `PixelMapCollection` using the string key `mapName`. The world coordinates produced by `mapName` are the lon and lat of a sky coordinate system described by `projection`. The `reprojectionName` is the name of a map that will transform coordinates from the gnomonic projection about the pointing center of this *exposure* into the projection about the *field* center.

2.5. `Extension`

Each input catalog has an `Extension` object. Index integers specify which `exposure` and `device` this catalog comes from (remember that the exposure will be known to be have been taken with a particular instrument). The `map` member points to the map from pixel coordinates to world coordinates (in the gnomonic projection for this field). The `wcs` points to a related object that can output `SphericalCoords` given a pixel position. This is the final product we are trying to produce! We may wish to save a FITS-compatible version of this WCS when we are done: `tpvOutFile` gives the name of the file this should be written to. We need a WCS for the catalog to establish initial sky positions for the object—this is referenced by the `startWcs` pointer, and can be a totally different functional form from the WCS we are trying to solve.

2.6. `CoordAlign`

This is the class that controls the WCS parameter-fitting operation (in *Match.h, Match.cpp*). The multithreading of the fitting operation is handled by this class's code. The `CoordAlign` is constructed with (1) a `PixelMapCollection` that knows all of the coordinate transformations to be used for all the `Detection`s and regulates which parameters are being held fixed or free, and (2) a list of pointers to `Match`es whose χ will be minimized by adjusting the WCS parameters.

The class has the following methods:

- `remap()` recalculates the world coordinates of all objects in all `Match`es, using the current WCS parameters.
- `sigmaClip(double sigThresh)` executes a single round of sigma-clipping on all `Match`es, marking as outliers the most deviant detection in any `Match` if it is more than `sigThresh` clipping sigmas away from the `Match` centroid. You can choose whether this is done on only the reserved matches or the non-reserved ones. Optionally you can have an entire `Match` marked as useless if any one of its `Detection`s is found to be an outlier.
- `chisqDOF()` calculates the χ and number of degrees of freedom of the unclipped objects under the current WCS parameters. Again you choose whether you're operating on the reserved or the un-reserved matches.
- `getParams(), setParams(), nParams()` manipulate the global parameter vector that is the union of all the parameters for all the maps in the `PixelMapCollection`.
- `count()` methods let you know how many matches & detections are in one or all of the catalogs.

FITTING???

3. Inputs

The program is called as

`WCSFit` ???⟨*field specs*⟩ ⟨*exposure specs*⟩ [*parameter file*] [*parameter file*] . . .

The standard input is read as additional parameter specifications. If you have no additional parameters you will need to send an empty file to stdin.

3.1. Parameters

Parameters can be specified one per line as ⟨*parameter name*⟩[=]⟨*value*⟩ [#;] *comment.*

- `matchRadius:` Distance (in arcsec) for friends-of-friends matching algorithm. (Default = 1.)
- `useAffinities:` Force matches to be from the same affinity class (`true`).
- `outName:` Filename for the output FITS tables (*match.cat*).
- `minMatch:` Minimum number of detections for a match to be retained (2).
- `selfMatch:` If false, reject all matches that have more than one detection from the same exposure (`true`).
- `renameInstruments:` A translation table for instrument names. This is a string having the format ⟨*regex1*⟩ = ⟨*replace1*⟩, ⟨*regex2*⟩ = ⟨*replace2*⟩, ... Incoming instrument names are checked for matches with any of the regular expressions; if they match one, the name is translated into the replacement test. Examples:
  - `renameInstruments = "i.* = i, r.* = r"` will change anything starting with `i` or `r` to the single-letter names.
  - `renameInstruments "(..).* = \1"` will use just the first two letters of any instrument name. The translated instrument name will be used when looking up an instrument in the input `PixelMapCollection` files.
- `stringAttributes, intAttributes, doubleAttributes:` comma-separated lists of any columns from the input exposure table that are to be passed on to the output tables.

3.2. Files

`WCSFoF` requires the names of two input files on the command line. The first is the *field specification* file. Each line of this file describes one field:

```
<Field name> <RA> <Dec> <extent>
```

The field names should be unique and have no spaces. The RA and Dec give the ICRS coordinates of the field center. The extent gives (in degrees) the half-length of the square field that will contain all objects in this field. The program will not properly match objects outside this square (unless they are assigned to another field).

The second input is the name of a FITS file with a single binary table extension containing our *filetable*. Each row of the filetable describes one FITS file that contains one or more extensions that will serve as our input detection catalogs.

The filetable will usually have been prepared by the *parse3.py* preprocessor. The filetable must have the columns specified below. Basically each column of the filetable gives one of the attributes that we will assign to the extensions read for that row of the filetable.

If a cell in the filetable begins with `@`, it is assumed to give the keyword of an entry in the FITS catalog's header. The desired value of the attribute is to be the value of the header card with this keyword. The primary header of the FITS file is searched first, then overridden by any values found in the header of the catalog's binary table extension. An error results if this keyword is not present in the input catalog's header.

The filetable must have these columns:

- `Filename:` (string) the path to the FITS file containing the input catalog(s).
- `Extension:` (int, default = -1) the extension number (zero-indexed) in the FITS file of the binary table detection catalog. If this is a negative integer, then it is assumed that all of the extensions of the FITS file (except extension 0, the primary HDU) are valid binary table catalogs. The code will sense if the FITS file is in the "LDAC" format produced by SExtractor, in which each catalog comes paired with another bintable holding the original image's header information. Note that any time `WCSFoF` needs to look for information in the input catalog headers, it will first look in the primary header of the FITS file, then override this with any values found in the header of the binary table extension (including any LDAC headers).
- `Field:` (string, default `_NEAREST`) the name of the field to which this file's catalogs belong. The special value "`_NEAREST`" can be entered to assign this input file to the field nearest to the `RA` and `Dec` attributes.
- `Exposure:` (string) the name of the exposure to which this catalog(s) belong.
- `Instrument:` (string) the name of the instrument with which the catalog's detections were made. The special values `REFERENCE` and `TAG` indicate the corresponding status for the extension.

- `Device:` (string) the name of the device with which the catalog's detections were made. Note that for multi-extension input files this is almost certainly specified as the value to be found in a header entry for each extension.

- `RA, Dec:` (strings) the position of the center of this exposure or extension, in the traditional sexagesimal hours/degrees notation, *e.g.* "12:24:30" "-30:04:02.8". Values are not needed if `WCSFile = ICRS` (see below) and the `Field` has been explicitly specified.

- `Select:` (string) specifies the selection criterion for objects from the input catalog to be used as detections. It should be an expression that evaluates to a boolean. For example, `FLAGS == 0 && ERRAWIN_IMAGE<0.05`. Variable names must correspond to columns in the extension's object catalog. Names starting with `@` are taken to be header keywords, and the value associated with the keyword in the extension (or primary) header is used.

- `Star_select:` (string) another boolean expression that determines whether an object is considered a star or a galaxy. Detections passing the star selection are assigned to the affinity class `STELLAR`.

- `Affinity:` (string) the affinity class to which non-stellar detections from this catalog are assigned.

- `xKey, yKey:` (string) the name of the columns in the catalog binary table that give the x and y pixel coordinates of each detection.

- `idKey:` (string, default `_ROW`) the name of the column in the catalog binary table that contains a unique identifying integer for each detection in that catalog. The special value `_ROW` will take the (zero-indexed) row number of the detection in the binary table as its identifier.

- `WCSFile:` (string) This gives the path to a file that, optionally, specifies the WCS for the extension(s) in this row of the filetable. The options here are to enter:

  - The name of a serialized `PixelMapCollection` file. In this case each extension of the catalog file will seek a map in this collection with the name ⟨*exposure name*⟩/⟨*device name*⟩. An error occurs if no map with this name is in the collection.
  - The name of an FITS-header-style ASCII file (as produced by SCAMP), in which we expect to find a WCS specified with FITS conventions. If the input catalog file has multiple extensions, the WCS for each are listed sequentially in the WCS file, seperated by the *END* keyword.
  - Nothing—in which case a WCS will be expected to be specified in the FITS headers of the catalog file.
  - The special value `ICRS`. This means that the x and y coordinates in the catalog are *already* in the ICRS system and no WCS is needed. These coordinates must be in **degrees.**

- Any attribute requested via the parameters `stringAttributes, intAttributes,` or `doubleAttributes` must be present in the filetable.

4. Outputs

4.1. `Files`