Astrometry.net Documentation

Release master

Lang, Hogg, Mierle, Roweis, etal

May 16, 2016

Contents:

CHAPTER 1

Astrometry.net code README

This code is accompanied by the paper:

Lang, D., Hogg, D. W.; Mierle, K., Blanton, M., & Roweis, S., 2010, Astrometry.net: Blind astrometric calibration of arbitrary astronomical images, Astronomical Journal 137, 1782–1800. http://arxiv.org/abs/0910.2233

The original purpose of this code release was to back up the claims in the paper in the interest of scientific repeatability. Over the years, it has become more robust and usable for a wider audience, but it's still neither totally easy nor bug-free.

This release includes a snapshot of all of the components of our current research code, including routines to:

- Convert raw USNO-B and Tycho2 into FITS format for easier use
- Uniformize, deduplicate, and cut the FITSified catalogs
- Build index files from these cuts
- Solve the astrometry of images using these index files

The code includes:

- A simple but powerful HEALPIX implementation
- The QFITS library with several modifications
- libkd, a compact and high-performance kdtree library

The code requires *index* files, processed from an astrometric reference catalog such as USNO-B1 or 2MASS. We have released several of these; see *Getting Index Files*.

1.1 Installing

See *Building/installing the Astrometry.net code*.

1.2 Getting Index Files

Get pre-cooked index files from: <http://data.astrometry.net/4200>_ (these are built from the 2MASS catalog).

Or, for wide-angle images, <http://data.astrometry.net/4100>_ (these are built from the Tycho-2 catalog).

We used to have the “4000-series” files, but these suffer from a bug where parts of the sky do are not covered by the reference catalog.

Each index file is designed to solve images within a narrow range of scales. The index files designed to solve small (angular size) images are rather large files, so you probably only want to grab the index files required for the images you wish to solve. If you grab extra index files, the solver will run more slowly, but the results should be the same.

The files are named like *index-42XX.fits* or *index-42XX-YY.fits*. *XX* is the “scale”, *YY* is the “healpix” number. These are called the “4200-series” index files.

Each index file contains a large number of “skymarks” (landmarks for the sky) that allow our solver to identify your images. The skymarks contained in each index file have sizes (diameters) within a narrow range. You probably want to download index files whose quads are, say, 10% to 100% of the sizes of the images you want to solve.

For example, let’s say you have some 1-degree square images. You should grab index files that contain skymarks of size 0.1 to 1 degree, or 6 to 60 arcminutes. Referring to the table below, you should grab index files 4203 through 4209. You might find that the same number of fields solve, and faster, using just one or two of the index files in the middle of that range - in our example you might try 4205, 4206 and 4207.

For reference, we used index files 202 alone for our SDSS tests (13x9 arcmin fields); these are the same scale is the new 4202 files.

The medium-sized index files are split into 12 “healpix” tiles; each one covers 1/12th of the sky. The small-sized ones are split into 48 healpixes. See the maps here; you might not need all of them. http://trac.astrometry.net/browser/trunk/src/astrometry/util/hp.png http://trac.astrometry.net/browser/trunk/src/astrometry/util/hp2.png

Index Filename	Range of skymark diameters (arcminutes)
`index-4219.fits`	1400–2000
`index-4218.fits`	1000–1400
`index-4217.fits`	680–1000
`index-4216.fits`	480–680
`index-4215.fits`	340–480
`index-4214.fits`	240–340
`index-4213.fits`	170–240
`index-4212.fits`	120–170
`index-4211.fits`	85–120
`index-4210.fits`	60—85
`index-4209.fits`	42–60
`index-4208.fits`	30–42
`index-4207-*.fits`	22–30
`index-4206-*.fits`	16–22
`index-4205-*.fits`	11–16
`index-4204-*.fits`	8–11
`index-4203-*.fits`	5.6–8.0
`index-4202-*.fits`	4.0–5.6
`index-4201-*.fits`	2.8–4.0
`index-4200-*.fits`	2.0–2.8

Download the index files you need and then either:

- Copy the files to the `data` directory wherever you installed the Astrometry.net code (`INSTALL_DIR/data`, perhaps `/usr/local/astrometry/data`); OR
- Copy the files to the top-level (astrometry-$VERSION) source directory, and run:

```
$ make install-indexes
```

Next, you can (optionally) configure the solver by editing the file:

```
INSTALL_DIR/etc/astrometry.cfg
```

1.2.1 Big-Endian Machines

Most CPUs these days are little-endian. If you have an Intel or AMD chip, you can skip this section. The most common big-endian CPU in recent times is the PowerPC used in Macs. If you have one of these, read on.

The index files we are distributing are for little-endian machines. For big-endian machines, you must do the following:

```
cd /usr/local/astrometry/data
for f in index-*.fits; do
  fits-flip-endian -i $f -o flip-$f -e 1 -s 4 -e 3 -s 4 -e 4 -s 2 -e 5 -s 8 -e 6 -s 2 -e 8 -s 4 -e 9
  for e in 0 2 7; do
    modhead flip-$f"[$e]" ENDIAN 01:02:03:04
  done
done
```

assuming `fits-flip-endian` and `modhead` are in your path. The files `flip-index-*.fits` will contain the flipped index files.

If that worked, you can swap the flipped ones into place (while saving the originals) with:

```
cd /usr/local/astrometry/data
mkdir -p orig
for f in index-*.fits; do
  echo "backing up $f"
  mv -n $f orig/$f
  echo "moving $f into place"
  mv -n flip-$f $f
done
```

1.3 Solving

Finally, solve some fields.

(If you didn't build the plotting commands, add "–no-plots" to the command lines below.)

(These lists of index files have not been updated; usually replacing "2xx" by "42xx" should work; for some of them the exact set that will solve has changed.)

If you have any of index files 4112 to 4118 (213 to 218):

```
$ solve-field --scale-low 10 demo/apod4.jpg
```

If you have any of index files 4115 to 4119 (219):

```
$ solve-field --scale-low 45 demo/apod5.jpg
```

If you have any of index files 210 to 214:

```
$ solve-field --scale-low 1 demo/apod3.jpg
```

If you have any of index files 206 to 211:

```
$ solve-field --scale-low 1 demo/apod2.jpg
```

If you have any of index files 203 to 205:

```
$ solve-field apod1.jpg
```

If you have any of index files 200 to 203:

```
$ solve-field demo/sdss.jpg
```

Copyrights and credits for the demo images are listed in the file `demo/CREDITS` .

Note that you can also give solve-field a URL rather than a file as input:

```
$ solve-field --out apod1b http://antwrp.gsfc.nasa.gov/apod/image/0302/ngc2264_croman_c3.jpg
```

If you don't have the netpbm tools (eg jpegtopnm), do this instead:

If you have any of index files 213 to 218:

```
$ solve-field --scale-low 10 demo/apod4.xyls
```

If you have index 219:

```
$ solve-field --scale-low 30 demo/apod5.xyls
```

If you have any of index files 210 to 214:

```
$ solve-field --scale-low 1 demo/apod3.xyls
```

If you have any of index files 206 to 211:

```
$ solve-field --scale-low 1 demo/apod2.xyls
```

If you have any of index files 203 to 205:

```
$ solve-field demo/apod1.xyls
```

If you have any of index files 200 to 203:

```
$ solve-field demo/sdss.xyls
```

1.3.1 Output files

<base>-ngc.png	an annotation of the image.
<base>.wcs	a FITS WCS header for the solution.
<base>.new	a new FITS file containing the WCS header.
<base>-objs.png	a plot of the sources (stars) we extracted from the image.
<base>-indx.png	sources (red), plus stars from the index (green), plus the skymark ("quad") used to solve the image.
<base>-indx.xyls	a FITS BINTABLE with the pixel locations of stars from the index.
<base>.rdls	a FITS BINTABLE with the RA,Dec of sources we extracted from the image.
<base>.axy	a FITS BINTABLE of the sources we extracted, plus headers that describe the job (how the image is going to be solved).
<base>.solved	exists and contains (binary) 1 if the field solved.
<base>.match	a FITS BINTABLE describing the quad match that solved the image.
<base>.kmz	(optional) KMZ file for Google Sky-in-Earth. You need to have "wcs2kml" in your PATH. See http://code.google.com/p/wcs2kml/downloads/list http://code.google.com/p/google-gflags/downloads/list

1.4 Tricks and Tips

- To lower the CPU time limit before giving up:

```
$  solve-field --cpulimit 30 ...
```

  will make it give up after 30 seconds.

  (Note, however, that the "backend" configuration file (astrometry.cfg) puts a limit on the CPU time that is spent on an image; solve-field can reduce this but not increase it.)

- Scale of the image: if you provide bounds (lower and upper limits) on the size of the image you are trying to solve, solving can be much faster. In the last examples above, for example, we specified that the field is at least 30 degrees wide: this means that we don't need to search for matches in the index files that contain only tiny skymarks.

  Eg, to specify that the image is between 1 and 2 degrees wide:

```
$ solve-field --scale-units degwidth --scale-low 1 --scale-high 2 ...
```

  If you know the pixel scale instead:

```
$ solve-field --scale-units arcsecperpix \
    --scale-low 0.386 --scale-high 0.406 ...
```

  When you tell solve-field the scale of your image, it uses this to decide which index files to try to use to solve your image; each index file contains quads whose scale is within a certain range, so if these quads are too big or too small to be in your image, there is no need to look in that index file. It is also used while matching quads: a small quad in your image is not allowed to match a large quad in the index file if such a match would cause the image scale to be outside the bounds you specified. However, all these checks are done before computing a best-fit WCS solution and polynomial distortion terms, so it is possible (though rare) for the final solution to fall outside the limits you specified. This should only happen when the solution is correct, but you gave incorrect inputs, so you shouldn't be complaining! :)

- Guess the scale: solve-field can try to guess your image's scale from a number of different FITS header values. When it's right, this often speeds up solving a lot, and when it's wrong it doesn't cost much. Enable this with:

```
$ solve-field --guess-scale ...
```

- If you've got big images: you might want to downsample them before doing source extraction:

```
$ solve-field --downsample 2 ...
$ solve-field --downsample 4 ...
```

- Depth. The solver works by looking at sources in your image, starting with the brightest. It searches for all "skymarks" that can be built from the N brightest stars before considering star N+1. When using several index files, it can be much faster to search for many skymarks in one index file before switching to the next one. This flag lets you control when the solver switches between index files. It also lets you control how much effort the solver puts in before giving up - by default it looks at all the sources in your image, and usually times out before this finishes.

  Eg, to first look at sources 1-20 in all index files, then sources 21-30 in all index files, then 31-40:

```
$ solve-field --depth 20,30,40 ...
```

  or:

```
$ solve-field --depth 1-20 --depth 21-30 --depth 31-40 ...
```

Sources are numbered starting at one, and ranges are inclusive. If you don't give a lower limit, it will take 1 + the previous upper limit. To look at a single source, do:

```
$ solve-field --depth 42-42 ...
```

- Our source extractor sometimes estimates the background badly, so by default we sort the stars by brightness using a compromise between the raw and background-subtracted flux estimates. For images without much nebulosity, you might find that using the background-subtracted fluxes yields faster results. Enable this by:

```
$ solve-field --resort ...
```

- If you've got big images: you might want to downsample them before doing source extraction:

```
$ solve-field --downsample 2 ...
```

or:

```
$ solve-field --downsample 4 ...
```

- When solve-field processes FITS files, it runs them through a "sanitizer" which tries to clean up non-standards-compliant images. If your FITS files are compliant, this is a waste of time, and you can avoid doing it:

```
$ solve-field --no-fits2fits ...
```

- When solve-field processes FITS images, it looks for an existing WCS header. If one is found, it tries to verify that header before trying to solve the image blindly. You can prevent this with:

```
$ solve-field --no-verify ...
```

Note that currently solve-field only understands a small subset of valid WCS headers: essentially just the TAN projection with a CD matrix (not CROT).

- If you don't want the plots to be produced:

```
$ solve-field --no-plots ...
```

- "I know where my image is to within 1 arcminute, how can I tell solve-field to only look there?"

```
$ solve-field --ra, --dec, --radius
```

Tells it to look within "radius" degrees of the given RA,Dec position.

- To convert a list of pixel coordinates to RA,Dec coordinates:

```
$ wcs-xy2rd -w wcs-file -i xy-list -o radec-list
```

Where xy-list is a FITS BINTABLE of the pixel locations of sources; recall that FITS specifies that the center of the first pixel is pixel coordinate (1,1).

- To convert from RA,Dec to pixels:

```
$ wcs-rd2xy -w wcs-file -i radec-list -o xy-list
```

- To make cool overlay plots: see `plotxy`, `plot-constellations`.

- To change the output filenames when processing multiple input files: each of the output filename options listed below can include "%s", which will be replaced by the base output filename. (Eg, the default for –wcs is "%s.wcs"). If you really want a "%" character in your output filename, you have to put "%%".

  Outputs include:

  – –new-fits

  - –kmz
  - –solved
  - –cancel
  - –match
  - –rdls
  - –corr
  - –wcs
  - –keep-xylist
  - –pnm

  also included:

  - –solved-in
  - –verify

- Reusing files between runs:

  The first time you run solve-field, save the source extraction results:

```
$ solve-field --keep-xylist %s.xy input.fits ...
```

  On subsequent runs, instead of using the original input file, use the saved xylist instead. Also add `--continue` to overwrite any output file that already exists.

```
$ solve-field input.xy --no-fits2fits --continue ...
```

  To skip previously solved inputs (note that this assumes single-HDU inputs):

```
$ solve-field --skip-solved ...
```

1.4.1 Optimizing the code

Here are some things you can do to make the code run faster:

- we try to guess "-mtune" settings that will work for you; if we're wrong, you can set the environment variable ARCH_FLAGS before compiling:

  $ ARCH_FLAGS="-mtune=nocona" make

  You can find details in the gcc manual: http://gcc.gnu.org/onlinedocs/

  You probably want to look in the section:

  "GCC Command Options"

  -> "Hardware Models and Configurations" -> "Intel 386 and AMD x86-64 Options"

  http://gcc.gnu.org/onlinedocs/gcc-4.3.0/gcc/i386-and-x86_002d64-Options.html#i386-and-x86_002d64-Options

1.4.2 What are all these programs?

When you "make install", you'll get a bunch of programs in /usr/local/astrometry/bin. Here's a brief synopsis of what each one does. For more details, run the program without arguments (most of them give at least a brief summary of what they do).

Image-solving programs:

- solve-field: main high-level command-line user interface.
- backend: higher-level solver that reads "augmented xylists"; called by solve-field.
- augment-xylist: creates "augmented xylists" from images, which include star positions and hints and instructions for solving.
- blind: low-level command-line solver.
- image2xy: source extractor.

Plotting programs:

- plotxy: plots circles, crosses, etc over images.
- plotquad: draws polygons over images.
- plot-constellations: annotates images with constellations, bright stars, Messier/NGC objects, Henry Draper catalog stars, etc.
- plotcat: produces density plots given lists of stars.

WCS utilities:

- new-wcs: merge a WCS solution with existing FITS header cards; can be used to create a new image file containing the WCS headers.
- fits-guess-scale: try to guess the scale of an image based on FITS headers.
- wcsinfo: print simple properties of WCS headers (scale, rotation, etc)
- wcs-xy2rd, wcs-rd2xy: convert between lists of pixel (x,y) and (RA,Dec) positions.
- wcs-resample: projects one FITS image onto another image.
- wcs-grab/get-wcs: try to interpret an existing WCS header.

Miscellany:

- an-fitstopnm: converts FITS images into ugly PNM images.
- get-healpix: which healpix covers a given RA,Dec?
- hpowned: which small healpixels are inside a big healpixel?
- control-program: sample code for how you might use the Astrometry.net code in your own software.
- xylist2fits: converts a text list of x,y positions to a FITS binary table.
- rdlsinfo: print stats about a list of RA,Dec positions (rdlist).
- xylsinfo: print stats about a list of x,y positions (xylist).

FITS utilities

- tablist: list values in a FITS binary table.
- modhead: print or modify FITS header cards.
- fitscopy: general FITS image / table copier.
- tabmerge: combines rows in two FITS tables.
- fitstomatlab: prints out FITS binary tables in a silly format.
- liststruc: shows the structure of a FITS file.
- listhead: prints FITS header cards.
- imcopy: copies FITS images.
- imarith: does (very) simple arithmetic on FITS images.
- imstat: computes statistics on FITS images.
- fitsgetext: pull out individual header or data blocks from multi-HDU FITS files.
- subtable: pull out a set of columns from a many-column FITS binary table.
- tabsort: sort a FITS binary table based on values in one column.
- column-merge: create a FITS binary table that includes columns from two input tables.
- add-healpix-column: given a FITS binary table containing RA and DEC columns, compute the HEALPIX and add it as a column.
- resort-xylist: used by solve-field to sort a list of stars using a compromise between background-subtracted and non-background-subtracted flux (because our source extractor sometimes messes up the background subtraction).
- fits-flip-endian: does endian-swapping of FITS binary tables.
- fits-dedup: removes duplicate header cards.

Index-building programs

- build-index: given a FITS binary table with RA,Dec, build an index file. This is the "easy", recent way. The old way uses the rest of these programs:
  - usnobtofits, tycho2tofits, nomadtofits, 2masstofits: convert catalogs into FITS binary tables.
  - build-an-catalog: convert input catalogs into a standard FITS binary table format.
  - cut-an: grab a bright, uniform subset of stars from a catalog.
  - startree: build a star kdtree from a catalog.
  - hpquads: find a bright, uniform set of N-star features.
  - codetree: build a kdtree from N-star shape descriptors.
  - unpermute-quads, unpermute-stars: reorder index files for efficiency.
- hpsplit: splits a list of FITS tables into healpix tiles

1.4.3 Source lists ("xylists")

The solve-field program accepts either images or "xylists" (xyls), which are just FITS BINTABLE files which contain two columns (float or double (E or D) format) which list the pixel coordinates of sources (stars, etc) in the image.

To specify the column names (eg, "XIMAGE" and "YIMAGE"):

```
$ solve-field --x-column XIMAGE --y-column YIMAGE ...
```

Our solver assumes that the sources are listed in order of brightness, with the brightest sources first. If your files aren't sorted, you can specify a column by which the file should be sorted.

```
$ solve-field --sort-column FLUX ...
```

By default it sorts with the largest value first (so it works correctly if the column contains FLUX values), but you can reverse that by:

```
$ solve-field --sort-ascending --sort-column MAG ...
```

When using xylists, you should also specify the original width and height of the image, in pixels:

```
$ solve-field --width 2000 --height 1500 ...
```

Alternatively, if the FITS header contains "IMAGEW" and "IMAGEH" keys, these will be used.

The solver can deal with multi-extension xylists; indeed, this is a convenient way to solve a large number of fields at once. You can tell it which extensions it should solve by:

```
$ solve-field --fields 1-100,120,130-200
```

(Ranges of fields are inclusive, and the first FITS extension is 1, as per the FITS standard.)

Unfortunately, the plotting code isn't smart about handling multiple fields, so if you're using multi-extension xylists you probably want to turn off plotting:

```
$ solve-field --no-plots ...
```

1.4.4 Backend config

Because we also operate a web service using most of the same software, the local version of the solver is a bit more complicated than it really needs to be. The "solve-field" program takes your input files, does source extraction on them to produce an "xylist" – a FITS BINTABLE of source positions – then takes the information you supplied about your fields on the command-line and adds FITS headers encoding this information. We call this file an "augmented xylist"; we use the filename suffix ".axy". "solve-field" then calls the "backend" program, passing it your axy file. "backend" reads a config file (by default /usr/local/astrometry/etc/astrometry.cfg) that describes things like where to find index files, whether to load all the index files at once or run them one at a time, how long to spend on each field, and so on. If you want to force only a certain set of index files to load, you can copy the astrometry.cfg file to a local version and change the list of index files that are loaded, and then tell solve-field to use this config file:

```
$ solve-field --config myastrometry.cfg ...
```

1.4.5 SExtractor

http://www.astromatic.net/software/sextractor

The "Source Extractor" aka "SExtractor" program by Emmanuel Bertin can be used to do source extraction if you don't want to use our own bundled "image2xy" program.

NOTE: users have reported that SExtractor 2.4.4 (available in some Ubuntu distributions) DOES NOT WORK – it prints out correct source positions as it runs, but the "xyls" output file it produces contains all (0,0). We haven't looked into why this is or how to work around it. Later versions of SExtractor such as 2.8.6 work fine.

You can tell solve-field to use SExtractor like this:

```
$ solve-field --use-sextractor ...
```

By default we use almost all SExtractor's default settings. The exceptions are:

1. **We write a PARAMETERS_NAME file containing:** X_IMAGE Y_IMAGE MAG_AUTO
2. We write a FILTER_NAME file containing a Gaussian PSF with FWHM of 2 pixels. (See blind/augment-xylist.c "filterstr" for the exact string.)
3. We set CATALOG_TYPE FITS_1.0
4. We set CATALOG_NAME to a temp filename.

If you want to override any of the settings we use, you can use:

```
$ solve-field --use-sextractor --sextractor-config <sex.conf>
```

In order to reproduce the default behavior, you must:

```
1) Create a parameters file like the one we make, and set
   PARAMETERS_NAME to its filename

2) Set::

$ solve-field --x-column X_IMAGE --y-column Y_IMAGE \
     --sort-column MAG_AUTO --sort-ascending

3) Create a filter file like the one we make, and set FILTER_NAME to
   its filename
```

Note that you can tell solve-field where to find SExtractor with:

```
$ solve-field --use-sextractor --sextractor-path <path-to-sex-executable>
```

1.4.6 Workarounds

- No python

  There are two places we use python: handling images, and filtering FITS files.

  You can avoid the image-handling code by doing source extraction yourself; see the "No netpbm" section below.

  You can avoid filtering FITS files by using the "–no-fits2fits" option to solve-field.

- No netpbm

  We use the netpbm tools (jpegtopnm, pnmtofits, etc) to convert from all sorts of image formats to PNM and FITS.

  If you don't have these programs installed, you must do source extraction yourself and use "xylists" rather than images as the input to solve-field. See SEXTRACTOR and XYLIST sections above.

1.4.7 ERROR MESSAGES during compiling

1. `/bin/sh: line 1: /dev/null: No such file or directory`

   We've seen this happen on Macs a couple of times. Reboot and it goes away...

2. `makefile.deps:40: deps: No such file or directory`

   Not a problem. We use automatic dependency tracking: "make" keeps track of which source files depend on which other source files. These dependencies get stored in a file named "deps"; when it doesn't exist, "make" tries to rebuild it, but not before printing this message.

3.
```
os-features-test.c: In function 'main':
os-features-test.c:23: warning: implicit declaration of function 'canonicalize_file_name'
os-features-test.c:23: warning: initialization makes pointer from integer without a cast
/usr/bin/ld: Undefined symbols:
_canonicalize_file_name
collect2: ld returned 1 exit status
```

   Not a problem. We provide replacements for a couple of OS-specific functions, but we need to decide whether to use them or not. We do that by trying to build a test program and checking whether it works. This failure tells us your OS doesn't provide the canonicalize_file_name() function, so we plug in a replacement.

4.
```
configure: WARNING: cfitsio: == No acceptable f77 found in $PATH
configure: WARNING: cfitsio: == Cfitsio will be built without Fortran wrapper support
drvrfile.c: In function 'file_truncate':
drvrfile.c:360: warning: implicit declaration of function 'ftruncate'
drvrnet.c: In function 'http_open':
drvrnet.c:300: warning: implicit declaration of function 'alarm'
drvrnet.c: In function 'http_open_network':
drvrnet.c:810: warning: implicit declaration of function 'close'
drvrsmem.c: In function 'shared_cleanup':
drvrsmem.c:154: warning: implicit declaration of function 'close'
group.c: In function 'fits_get_cwd':
group.c:5439: warning: implicit declaration of function 'getcwd'
ar: creating archive libcfitsio.a
```

   Not a problem; these errors come from cfitsio and we just haven't fixed them.

1.5 License

The Astrometry.net code suite is free software licensed under the GNU GPL, version 2. See the file LICENSE for the full terms of the GNU GPL.

The index files come with their own license conditions. See the file GETTING-INDEXES for details.

1.6 Contact

You can post questions (or maybe even find the answer to your questions) at http://forum.astrometry.net . However, please also send an email to "code2 at astrometry dot net" pointing out your post to the forum – we never remember to check the forum! We would also be happy to hear via email any bug reports, comments, critiques, feature requests, and in general any reports on your experiences, good or bad.

CHAPTER 2

Change Log:

2.1 Version 0.46:

- Makefile revamp. Now possible to use system GSL rather than our shipped subset, by defining SYSTEM_GSL=yes:

```
make SYSTEM_GSL=yes
```

- Move away from *qfits*, toward *an-qfits*; reorganize *qfits-an* directory to be more like the other source directories.
- Web api: add *invert* option
- Add more Intermediate World Coords options to WCS classes

2.2 Version 0.47:

- *solve-field*: add “focalmm” as a supported image scale type.

CHAPTER 3

Building/installing the Astrometry.net code

Grab the code:

```
wget http://astrometry.net/downloads/astrometry.net-latest.tar.bz2
tar xjf astrometry.net-latest.tar.bz2
cd astrometry.net-*
```

Build it. The short version:

```
make
make py
make extra
make install  # to put it in /usr/local/astrometry
# or:
make install INSTALL_DIR=/some/other/place
```

The long version:

3.1 Prerequisites

For full functionality, you will need:

- GNU build tools (gcc/clang, make, etc.)
- cairo
- netpbm
- libpng
- libjpeg
- libz
- bzip2
- python (probably >= 2.4)
- numpy
- pyfits: http://www.stsci.edu/resources/software_hardware/pyfits (version >= 3.1)
- cfitsio: http://heasarc.gsfc.nasa.gov/fitsio/

3.1.1 Ubuntu or Debian-like systems:

```
$ sudo apt-get install libcairo2-dev libnetpbm10-dev netpbm \
                       libpng12-dev libjpeg-dev python-numpy \
                       python-pyfits python-dev zlib1g-dev \
                       libbz2-dev swig cfitsio-dev
```

3.1.2 CentOS 6.5 / Fedora / RedHat / RHEL – Detailed Instructions:

See these instructions from James Chamberlain.

3.1.3 Mac OS X using homebrew:

These instructions *Worked For Me* as of September 2012 on OSX 10.8.

First set up homebrew:

- grab XCode (from the Apps Store. Free, but you still need a credit card. Argh.)
- grab XCode Command-line utilities
- grab XQuartz
- grab Homebrew
- grab pip if you don't have it already

Get homebrew dependencies that need special instructions:

```
$ brew install --HEAD --use-gcc netpbm
```

Optionally, grab some other handy homebrew packages:

```
$ brew install cfitsio --with-examples
$ brew install md5sha1sum     # OSX doesn't come with this?!  For shame
```

Get our fork of homebrew-science and install:

```
$ brew tap homebrew/homebrew-science
$ brew install astrometry-net
```

Or:

```
$ brew install --HEAD astrometry.net
```

if you like to live dangerously (but trendily).

3.1.4 Mac OS X using Fink:

Use apt-get install as per the Debian instructions above (leaving out `zlib1g-dev` because it's already included with OSX). Note that to use Fink you will need to add something like this in your `~/.profile` or `~/.bashrc` file:

```
. /sw/bin/init.sh
export CFLAGS="-I/usr/local/include -I/sw/include"
export LDFLAGS="-L/usr/local/lib -L/sw/lib"
```

3.2 Getting/Building

If you don't have and can't get these libraries, you should still be able to compile and use the core parts of the solver, but you will miss out on some eye-candy.

Build the solving system:

```
$ make
```

If you installed the libraries listed above, build the plotting code:

```
$ make extra
```

Install it:

```
$ make install
```

You might see some error message during compilation; see the section ERROR MESSAGES below for fixes to common problems.

By default it will be installed in `/usr/local/astrometry` . You can override this by either:

- editing the top-level Makefile (look for INSTALL_DIR); or
- defining INSTALL_DIR on the command-line:

  For bash shell:

  ```
  $ export INSTALL_DIR=/path/to/astrometry
  $ make install
  ```

  or:

  ```
  $ INSTALL_DIR=/path/to/astrometry make install
  ```

  For tcsh shell:

  ```
  $ setenv INSTALL_DIR /path/to/astrometry
  $ make install
  ```

The astrometry solver is composed of several executables. You may want to add the INSTALL_DIR/bin directory to your path:

  For bash shell:

  ```
  $ export PATH="$PATH:/usr/local/astrometry/bin"
  ```

  For tcsh shell:

  ```
  $ setenv PATH "$PATH:/usr/local/astrometry/bin"
  ```

3.3 Auto-config

We use a do-it-yourself auto-config system that tries to detect what is available on your machine. It is called `os-features`, and it works by trying to compile, link, and run a number of executables to detect:

- whether the "netpbm" library is available
- whether certain GNU-specific function calls exist

You can change the flags used to compile and link "netpbm" by either:

- editing util/makefile.netpbm
- setting NETPBM_INC or NETPBM_LIB, like this:

```
    $ make NETPBM_INC="-I/tmp" NETPBM_LIB="-L/tmp -lnetpbm"
```

You can see whether netpbm was successfully detected by:

```
$ cat util/makefile.os-features
# This file is generated by util/Makefile.
HAVE_NETPBM := yes
```

You can force a re-detection either by deleting util/makefile.os-features and util/os-features-config.h, or running:

```
$ make reconfig
```

(which just deletes those files)

3.4 Overriding Things

For most of the libraries we use, there is a file called `util/makefile.*` where we try to auto-configure where the headers and libraries can be found. We use `pkg-config` when possible, but you can override things.

`*_INC` are the compile flags (eg, for the include files).

`*_LIB` is for libraries.

`*_SLIB`, when used, is for static libraries (.a files).

3.4.1 gsl:

You can either use your system's GSL (GNU scientific library) libraries, or the subset we ship. (You don't need to do anything special to use the shipped version.)

System:

```
make SYSTEM_GSL=yes
```

Or specify static lib:

```
make SYSTEM_GSL=yes GSL_INC="-I/to/gsl/include" GSL_SLIB="/to/gsl/lib/libgsl.a"
```

Or specify dynamic lib:

```
make SYSTEM_GSL=yes GSL_INC="-I/to/gsl/include" GSL_LIB="-L/to/gsl/lib -lgsl"
```

3.4.2 cfitsio:

For dynamic libs:

```
make CFITS_INC="-I/to/cfitsio/include" CFITS_LIB="-L/to/cfitsio/lib -lcfitsio"
```

Or for static lib:

```
make CFITS_INC="-I/to/cfitsio" CFITS_SLIB="/to/cfitsio/lib/libcfitsio.a"
```

3.4.3 netpbm:

```
make NETPBM_INC="-I/to/netpbm" NETPBM_LIB="-L/to/netpbm/lib -lnetpbm"
```

3.4.4 wcslib:

Ditto, with `WCSLIB_INC`, `WCSLIB_LIB`, `WCS_SLIB`

3.4.5 cairo:

`CAIRO_INC`, `CAIRO_LIB`

3.4.6 jpeg:

`JPEG_INC`, `JPEG_LIB`

3.4.7 png:

`PNG_INC`, `PNG_LIB`

3.4.8 zlib:

`ZLIB_INC`, `ZLIB_LIB`

CHAPTER 4

Building Index Files for Astrometry.net

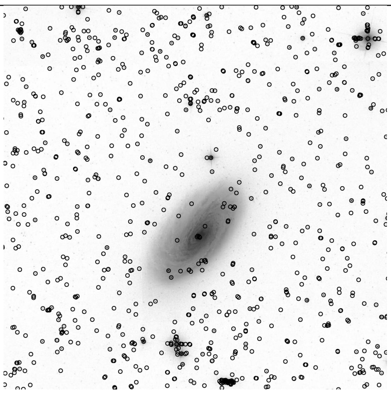
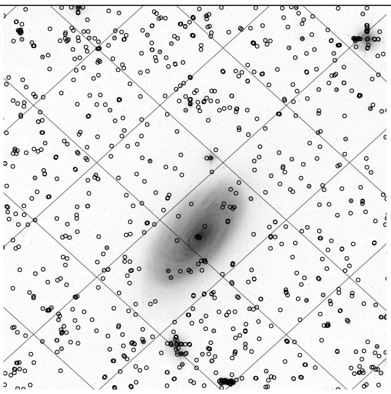
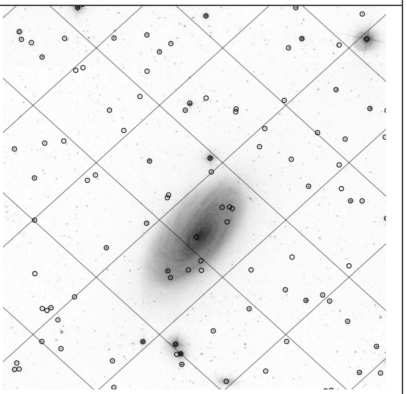
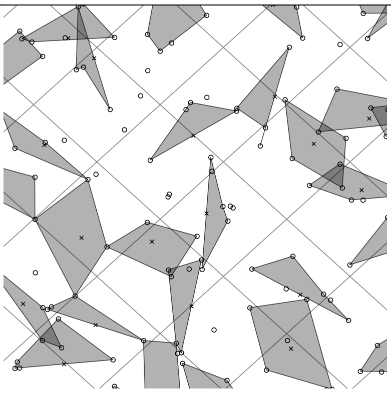
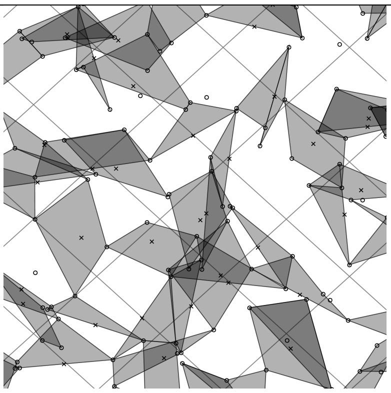
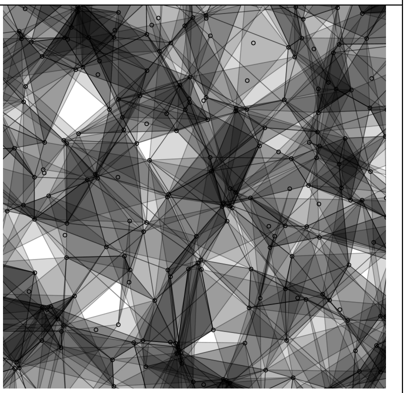
Astrometry.net searches the sky using *index files*. These contain a set of "stars" (maybe also galaxies), selected so that they are relatively bright and cover the sky uniformly. They also contain a large number of *features* or *quads* that describe the local shape of sets of (usually four) stars. Each *feature* points back to the stars it is composed of. The Astrometry.net engine works by detecting stars in your image, and then looking at sets of (usually four) stars, computing their local shape, and searching in the index files for features with similar shapes. For each similar shape that is found, it retrieves *other* stars in the area and checks whether other reference stars are aligned with other stars in your image.

While we distribute index files based on the 2MASS and Tycho-2 astrometric catalogs that should work for most purposes, some people want to use other reference catalogs for their particular purpsose. This document explains how to build custom index files from a reference catalog.

The steps are:

- *Convert your reference catalog to FITS tables*
- *Prepare your FITS tables*
- *Split the sky into pieces*
- *Building Index Files*
- *Using your shiny new index files*

Here are some pictures of the index-building process itself:

A reference catalog:	We lay down a healpix grid:	And select the brightest stars in each cell:
		
And then try to build a 4-star feature centered in each cell	And again...	And again, until the sky is densely tiled in features.
		

4.1 Convert your reference catalog to FITS tables

The Astrometry.net index files are FITS tables, and the index-building process take FITS tables as inputs.

Many astrometric reference catalogs are available in FITS format. For those that aren't, here are a few options for converting to FITS BINTABLE (binary table) format: * *text2fits.py* in the Astrometry.net package—useful for CSV (comma-separated values) and other ASCII text inputs; this is a simple parser and takes a huge amount of memory to process big files. It would be possible to make it "stream" the inputs and outputs, but I haven't done that (yet). * Custom format converters, including *2masstofits*, *nomadtofits*, *ucac3tofits*, and *usnobtofits* (all in the Astrometry.net package). * Check the Vizier service to see if your catalog is available there; sometimes you can download it as FITS binary table (in the "Preferences" box for output format). I find the Vizier search engine impossible to use; just use your favorite web search engine to query, say, "vizier ucac4". * Write your own custom converter. If I had to do this again, I would rewrite all the *Xtofits* converters above in python, probably using the struct module. But if you are converting a format that is very similar to one of the above, the fastest may be to copy-n-edit one of the existing ones. If you do this, please consider contributing your code back to the Astrometry.net codebase.

As for python FITS table handling, the *best* option is fitsio. The *most popular* option is probably pyfits. The Astrometry.net package includes a wrapper that can use either of those; util/fits.py.

The *cfitsio* package includes some tools for handling FITS tables, in particular *liststruc* (list the structure of a FITS file), *listhead* (print the headers), *fitscopy* (copy files, possible with manipulations; see extended filename syntax.).

4.2 Prepare your FITS tables

You may want to make some cuts, remove irrelevant columns, or otherwise prepare your FITS tables before feeding them into the index-building stage. At the very least, you want your FITS tables to contain *RA* and *DEC* columns, as well as a column that defines the brightness ordering of your stars: probably a *MAG*.

Any other columns you include can optionally be propagated into the index files, so that after getting an astrometric match you will also have access to this "tag-along" data. This is useful for, for example, doing an initial photometric calibration by tagging-along one or more bands of photometric data for each astrometric star.

As an example, the file 2mass-cut.py implements the cut we used to build our 2MASS-based index files. It removes any stars that are flagged in the 2MASS catalog (low quality, contaminated, etc), and writes out just the RA,Dec, and J-magnitude columns.

4.3 Split the sky into pieces

Optionally, you can split the sky into slightly overlapping pieces.

Why split the sky into pieces? First, it results in smaller files that can be easier to handle. Second, if you have an initial guess of where your image is on the sky, the Astrometry.net engine can avoid loading sky tiles that don't overlap, so it results in faster and less memory-intensive searches.

If you don't split the sky into pieces, at this point you should combine your input catalog files into a single FITS table, if you haven't done that already. You can use the *tabmerge* program for that.

Splitting the sky into pieces is done using the *hpsplit* program. It takes a number of input FITS tables and produces one output table for each *healpix* tile:

```
> hpsplit -h
This program is part of the Astrometry.net suite.
For details, visit  http://astrometry.net .
Subversion URL svn+ssh://astrometry.net/svn/trunk/src/astrometry/util/
Revision 22921, date 2013-06-02 15:07:59 -0400 (Sun, 02 Jun 2013).

Usage: hpsplit [options] <input-FITS-catalog> [...]
    -o <output-filename-pattern>  with %i printf-pattern
    [-r <ra-column-name>]: name of RA in FITS table (default RA)
    [-d <dec-column-name>]: name of DEC in FITS table (default DEC)
    [-n <healpix Nside>]: default is 1
    [-m <margin in deg>]: add a margin of this many degrees around the healpixes; default 0
    [-g]: gzip'd inputs
    [-c <name>]: copy given column name to the output files
    [-t <temp-dir>]: use the given temp dir; default is /tmp
    [-b <backref-file>]: save the filenumber->filename map in this file; enables writing backreferenc
    [-v]: +verbose
```

The number of healpix tiles is determined by the *Nside* (-n) option. `-n 1` means split the sky into 12 pieces. `-n 2` means split the sky into 48 pieces. You probably don't want to go any finer than that.

For reference, maps of the healpix tiles are here: Nside=1 healpixes; Nside=2 healpixes.

You probably want to set `-m` for the *margin* – extra overlapping area around each healpix tile. You probably want to set this about half as big as the images you are going to solve. This will mean that in the margin areas, multiple healpix tiles will contain the same stars.

If you want to "tag-along" extra information into the index files, include those columns with the `-c` option.

Example hpsplit command:

```
hpsplit -o 2mass-hp%02i.fits -n 2 -m 1 2mass/2mass-*.fits
```

Notice the `%02i` in the output filename; that's a "printf string" that says, write an integer, using 2 digits, padding with zeros. The outputs will be named 2mass-hp00.fits through 2mass-hp11.fits (for `-n 1`).

At the end of this, you will have 12 or 48 FITS tables (assuming your input catalog was all-sky; fewer if not). You will build several index file for each of these (each one covering one scale).

4.4 Building Index Files

Finally! The real deal.

build-astrometry-index has a daunting number of options, but don't panic:

```
> build-astrometry-index
You must specify input & output filenames.
This program is part of the Astrometry.net suite.
For details, visit  http://astrometry.net .
Subversion URL svn+ssh://astrometry.net/svn/trunk/src/astrometry/util/
Revision 22921, date 2013-06-02 15:07:59 -0400 (Sun, 02 Jun 2013).

Usage: build-astrometry-index
      (
         -i <input-FITS-catalog>  input: source RA,DEC, etc
    OR,
         -1 <input-index>         to share another index's stars
      )
      -o <output-index>        output filename for index
      (
         -P <scale-number>: use 'preset' values for '-N', '-l', and '-u'
               (the scale-number is the last two digits of the pre-cooked
                index filename -- eg, index-205 is  "-P 5".
                -P 0  should be good for images about 6 arcmin in size
                    and it goes in steps of sqrt(2), so:
                -P 2  should work for images about 12 arcmin across
                -P 4  should work for images about 24 arcmin across
                -P 6  should work for images about 1 degree across
                -P 8  should work for images about 2 degree across
                -P 10 should work for images about 4 degree across
                 etc... up to -P 19
  OR,
         -N <nside>            healpix Nside for quad-building
         -l <min-quad-size>    minimum quad size (arcminutes)
         -u <max-quad-size>    maximum quad size (arcminutes)
      )
      [-S <column>]: sort column (default: assume the input file is already sorted)
      [-f]: sort in descending order (eg, for FLUX); default ascending (eg, for MAG)
      [-A <column>]: specify the RA  column name in the input FITS table (default "RA")
      [-D <column>]: specify the Dec column name in the input FITS table (default "Dec")
      [-B <val>]: cut any object whose sort-column value is less than 'val'; for mags this is a brigh
      [-U]: healpix Nside for uniformization (default: same as -n)
      [-H <big healpix>]; default is all-sky
      [-s <big healpix Nside>]; default is 1
      [-m <margin>]: add a margin of <margin> healpixels; default 0
      [-n <sweeps>]    (ie, number of stars per fine healpix grid cell); default 10
      [-r <dedup-radius>]: deduplication radius in arcseconds; default no deduplication
      [-j <jitter-arcsec>]: positional error of stars in the reference catalog (in arcsec; default 1)
      [-d <dimquads>] number of stars in a "quad" (default 4).
      [-p <passes>]   number of rounds of quad-building (ie, # quads per healpix cell, default 16)
      [-R <reuse-times>] number of times a star can be used (default: 8)
      [-L <max-reuses>] make extra passes through the healpixes, increasing the "-r" reuse
                    limit each time, up to "max-reuses".
      [-E]: scan through the catalog, checking which healpixes are occupied.

      [-I <unique-id>] set the unique ID of this index

      [-M]: in-memory (don't use temp files)
      [-T]: don't delete temp files
      [-t <temp-dir>]: use this temp direcotry (default: /tmp)
      [-v]: add verbosity.
```

I will list them from most important to least (and roughly top-to-bottom).

Input file:

```
  (
     -i <input-FITS-catalog>  input: source RA,DEC, etc
OR,
     -1 <input-index>         to share another index's stars
  )
```

The `-1` version is only used in the LSST index files; everyone else should probably use `-i`. This will be the FITS file you have carefully created as detailed above.

Output filename:

```
-o <output-index>        output filename for index
```

Easy! I usually just name mine with a number, the healpix tile, and scale, but you can do anything that makes sense to you. These will be FITS tables, so the suffix .fits would be appropriate, but none of the code cares about the filenames, so do what you like.

Index scale:

```
    (
       -P <scale-number>: use 'preset' values for '-N', '-l', and '-u'
             (the scale-number is the last two digits of the pre-cooked
              index filename -- eg, index-205 is  "-P 5".
              -P 0  should be good for images about 6 arcmin in size
                  and it goes in steps of sqrt(2), so:
              -P 2  should work for images about 12 arcmin across
              -P 4  should work for images about 24 arcmin across
              -P 6  should work for images about 1 degree across
              -P 8  should work for images about 2 degree across
              -P 10 should work for images about 4 degree across
               etc... up to -P 19
OR,
       -N <nside>            healpix Nside for quad-building
       -l <min-quad-size>    minimum quad size (arcminutes)
       -u <max-quad-size>    maximum quad size (arcminutes)
    )
    ...
    [-U]: healpix Nside for uniformization (default: same as -n)
```

This determines the scale on which stars are selected uniformly on the sky, the scale at which features are selected, and the angular size of the features to create. In Astrometry.net land, we use a “preset” number of scales, each one covering a range of about square-root-of-2. Totally arbitrarily, the range 2.0-to-2.4 arcminutes is called scale zero.

You want to have features that are maybe 25% to 75% of the size of your image, so you probably want to build a range of scales. For reference, for most of the experiments in my thesis I used scale 2 (4 to 5.6 arcmin features) to recognize Sloan Digital Sky Survey images, which are 13-by-9 arcminutes. Scales 3, 4, and 1 also yielded solutions when they were included.

You will run build-astrometry-index once for each scale.

Presets in the range -5 to 19 are available. The scales for the presets are listed in the Getting Index Files documentation.

Rather than use the `-P` option it is possible to specify separately the different scales using `-N`, `-l`, `-u`, `-U`, but I wouldn't recommend it. The presets are listed in build-index-main.chealpixeshttp://trac.astrometry.net/browser/trunk/src/astrometry/blind/build-index-main.c.

Sort column:

```
[-S <column>]: sort column (default: assume the input file is already sorted)
[-f]: sort in descending order (eg, for FLUX); default ascending (eg, for MAG)
[-B <val>]: cut any object whose sort-column value is less than 'val'; for mags this is a bright lim
```

Which column in your FITS table input should we use to determine which stars are bright? (We preferentially select bright stars to include in the index files.) Typically this will be something like:

```
build-astrometry-index -S J_mag [...]
```

By default, we assume that SMALL values of the sorting column are bright – that is, it works for MAGs. If you have linear FLUX-like units, then use the `-f` flag to reverse the sorting direction.

It is also possible to *cut* objects whose sort-column value is less than a lower limit, using the `-B` flag.

Which part of the sky is this?:

```
[-H <big healpix>]; default is all-sky
[-s <big healpix Nside>]; default is 1
```

You need to tell build-astrometry-index which part of the sky it is indexing. By default, it assumes you are building an all-sky index.

If you have split your reference catalog into 12 pieces (healpix Nside = 1) using *hpsplit* as described above, then you will run *build-astrometry-index* once for each healpix tile FITS table and scale, specifying the tile number with `-H` and the Nside with `-s` (default is 1), and specifying the scale with `-P`:

```
# Healpix 0, scales 2-4
build-astrometry-index -i catalog-hp00.fits -H 0 -s 1 -P 2 -o myindex-02-00.fits [...]
build-astrometry-index -i catalog-hp00.fits -H 0 -s 1 -P 3 -o myindex-03-00.fits [...]
build-astrometry-index -i catalog-hp00.fits -H 0 -s 1 -P 4 -o myindex-04-00.fits [...]
# Healpix 1, scales 2-4
build-astrometry-index -i catalog-hp01.fits -H 1 -s 1 -P 2 -o myindex-02-01.fits [...]
build-astrometry-index -i catalog-hp01.fits -H 1 -s 1 -P 3 -o myindex-03-01.fits [...]
build-astrometry-index -i catalog-hp01.fits -H 1 -s 1 -P 4 -o myindex-04-01.fits [...]

...
# Healpix 11, scales 2-4
build-astrometry-index -i catalog-hp11.fits -H 1 -s 1 -P 2 -o myindex-02-11.fits [...]
build-astrometry-index -i catalog-hp11.fits -H 1 -s 1 -P 3 -o myindex-03-11.fits [...]
build-astrometry-index -i catalog-hp11.fits -H 1 -s 1 -P 4 -o myindex-04-11.fits [...]
```

You probably want to do that using a loop in your shell; for example, in bash:

```
for ((HP=0; HP<12; HP++)); do
  for ((SCALE=2; SCALE<=4; SCALE++)); do
    HH=$(printf %02i $HP)
    SS=$(printf %02i $SCALE)
    build-astrometry-index -i catalog-hp${HH}.fits -H $HP -s 1 -P $SCALE -o myindex-${HH}-${SS}.fits
  done
done
```

Sparse catalog?:

```
[-E]: scan through the catalog, checking which healpixes are occupied.
```

If your catalog only covers a small part of the sky, be sure to set the `-E` flag, so that `build-astrometry-index` only tries to select features in the part of the sky that your index covers.

Unique ID:

```
[-I <unique-id>] set the unique ID of this index
```

Select an identifier for your index files. I usually encode the date and scale: eg 2013-08-01, scale 2, becomes 13080102. Or I keep a running number, like the 4100-series and 4200-series files. The different healpixes at a scale do not need unique IDs.

Triangles?:

```
[-d <dimquads>] number of stars in a "quad" (default 4).
```

Normally we use four-star featurse. This allows you to build 3- or 5-star features instead. 3-star features are useful for wide-angle images. 5-star features are probably not useful for most purposes.

4.4.1 You probably don't need to set any of the options below here

RA,Dec column names:

```
[-A <column>]: specify the RA  column name in the input FITS table (default "RA")
[-D <column>]: specify the Dec column name in the input FITS table (default "Dec")
```

I would recommend naming your RA and Dec columns "RA" and "DEC", but if for some reason you don't want to do that, you need to tell `build-astrometry-index` what they're called at this point, using the `-A` and `-D` options:

```
build-astrometry-index -A Alpha_J2000 -D Delta_J2000 [...]
```

Indexing Details:

```
[-m <margin>]: add a margin of <margin> healpixels; default 0
```

Try to create features in a margin around each healpix tile. Not normally necessary: the healpix tiles can contain overlapping margins *stars*, so each one can recognize images that straddle its boundary. There's no need to also cover the margin regions with (probably duplicate) features.

```
[-n <sweeps>]    (ie, number of stars per fine healpix grid cell); default 10
```

We try to select a bright, uniform subset of stars from your reference catalog by laying down a fine healpix grid and selecting `-n` stars from each. This allows you to select fewer or more. With fewer, you risk being unable to recognize some images. With more, file sizes will be bigger.

```
[-r <dedup-radius>]: deduplication radius in arcseconds; default no deduplication
```

We can remove stars that are within a radius of exclusion of each other (eg, double stars, or problems with the reference catalog).

```
[-j <jitter-arcsec>]: positional error of stars in the reference catalog (in arcsec; default 1)
```

The index files contain a FITS header card saying what the typical astrometric error is. This is used when "verifying" a proposed match; I don't think the system is very sensitive to this value.

```
[-p <passes>]   number of rounds of quad-building (ie, # quads per healpix cell, default 16)
```

We try to build a uniform set of features by laying down a fine healpix grid and trying to build a feature in each cell. We run multiple passes of this, building a total of `-p` features in each cell.

```
[-R <reuse-times>] number of times a star can be used (default: 8)
```

By default, any star can be used in at most 8 features. This prevents us from relying too heavily on any one star.

```
[-L <max-reuses>] make extra passes through the healpixes, increasing the "-r" reuse
                  limit each time, up to "max-reuses".
```

Sometimes the `-R` option means that we "use up" all the stars in an area and can't build as many features as we would like. This option enables a second pass where we loosen up with `-R` value, trying to build extra features.

Runtime details:

```
[-M]: in-memory (don't use temp files)
[-T]: don't delete temp files
[-t <temp-dir>]: use this temp direcotry (default: /tmp)
[-v]: add verbosity.
```

The help messages are all pretty self-explanatory, no?

4.5 Using your shiny new index files

In order to use your new index files, you need to create a *backend config* file that tells the astrometry engine where to find them.

The default backend config file is in /usr/local/astrometry/etc/backend.cfg

You can either edit that file, or create a new .cfg file. Either way, you need to add lines like:

```
# In which directories should we search for indices?
add_path /home/dstn/astrometry/data

# Load any indices found in the directories listed above.
autoindex

## Or... explicitly list the indices to load.
#index index-4200-00.fits
#index index-4200-01.fits
```

It is safe to include multiple sets of index files that cover the same region of sky, mix and match, or whatever. The astrometry engine will just use whatever you give it.

If you edited the default backend.cfg file, `solve-field` and `backend` will start using your new index files right away. If you create a new index file (I often put one in the directory containing the index files themselves), you need to tell `solve-field` where it is:

```
solve-field --backend-config /path/to/backend.cfg [...]
```

That's it! Report successes, failures, frustrations, missing documentation, spelling errors, and such at the Astrometry.net google group.

CHAPTER 5

Nova.astrometry.net: API

We provide a web-services API that uses JSON to encode the parameters and results.

We have code-as-documentation in the client.py file, but will try to keep this documentation up to date as well.

5.1 JSON encoding

The first message you send to the server will be to log in using your API key. The formatting steps are:

- Form the JSON *object*:

```
{"apikey": "XXXXXX"}
```

- Form the *x-www-form-encoded* string:

```
request-json={"apikey": "XXXXXX"}
```

  which becomes:

```
request-json=%7B%22apikey%22%3A+%22XXXXXX%22%7D
```

- Send that to the "login" API URL as a POST request:

  http://nova.astrometry.net/api/login

Using the *requests* library, this looks like:

```
import requests
import json
R = requests.post('http://nova.astrometry.net/api/login', data={'request-json': json.dumps({"apikey"
print(R.text)
>> u'{"status": "success", "message": "authenticated user: ", "session": "0ps9ztf2kmplhc2gupfne2em5q
```

Note the *request-json=VALUE*: we're not sending raw JSON, we're sending the JSON-encoded string as though it were a text field called *request-json*.

All the other API calls use this same pattern. It may be more complicated on the client side, but it makes testing and the server side easier.

This form demonstrates how the request must be encoded, and what the result looks like.

5.2 Session key

After logging in with your API key, you will get back a “session key”:

```
{"status": "success", "message": "authenticated user: ", "session": "575d80cf44c0aba5491645a6818589c
```

You have to include that “session” value in all subsequent requests. The session will remain alive for some period of time, or until you log out.

5.3 Submitting a URL

API URL:

http://nova.astrometry.net/api/url_upload

Arguments:

- `session`: string, requried. Your session key, required in all requests
- `url`: string, required. The URL you want to submit to be solved
- `allow_commercial_use`: string: “d” (default), “y”, “n”: licensing terms
- `allow_modifications`: string: “d” (default), “y”, “n”, “sa” (share-alike): licensing terms
- `publicly_visible`: string: “y”, “n”
- `scale_units`: string: “degwidth” (default), “arcminwidth”, “arcsecperpix”. The units for the “scale_lower” and “scale_upper” arguments; becomes the “–scale-units” argument to “solve-field” on the server side.
- `scale_type`: string, “ul” (default) or “ev”. Set “ul” if you are going to provide “scale_lower” and “scale_upper” arguments, or “ev” if you are going to provide “scale_est” (estimate) and “scale_err” (error percentage) arguments.
- `scale_lower`: float. The lower-bound of the scale of the image.
- `scale_upper`: float. The upper-bound of the scale of the image.
- `scale_est`: float. The estimated scale of the image.
- `scale_err`: float, 0 to 100. The error (percentage) on the estimated scale of the image.
- `center_ra`: float, 0 to 360, in degrees. The position of the center of the image.
- `center_dec`: float, -90 to 90, in degrees. The position of the center of the image.
- `radius`: float, in degrees. Used with `center_ra`,‘‘center_dec‘‘ to specify that you know roughly where your image is on the sky.
- `downsample_factor`: float, >1. Downsample (bin) your image by this factor before performing source detection. This often helps with saturated images, noisy images, and large images. 2 and 4 are commonly-useful values.
- `tweak_order`: int. Polynomial degree (order) for distortion correction. Default is 2. Higher orders may produce totally bogus results (high-order polynomials are strange beasts).
- `use_sextractor`: boolean. Use the SourceExtractor program to detect stars, not our built-in program.
- `crpix_center`: boolean. Set the WCS reference position to be the center pixel in the image? By default the center is the center of the quadrangle of stars used to identify the image.

- `parity`: int, 0, 1 or 2. Default 2 means "try both". 0 means that the sign of the determinant of the WCS CD matrix is positive, 1 means negative. The short answer is, "try both and see which one works" if you are interested in using this option. It results in searching half as many matches so can be helpful speed-wise.
- `image_width`: int, only necessary if you are submitting an "x,y list" of source positions.
- `image_height`: int, ditto.
- `positional_error`: float, expected error on the positions of stars in your image. Default is 1.

Example:

And you will get back a response such as:

```
{"status": "success", "subid": 16714, "hash": "6024b45a16bfb5af7a73735cbabdf2b462c11214"}
```

The `subid` is the Submission number. The `hash` is the `sha-1` hash of the contents of the URL you specified.

5.4 Submitting a file

Submitting a file is somewhat complicated, because it has to be formatted as a *multipart/form-data* form. This is exactly the same way an HTML form with text fields and a file upload field would do it. If you are working in python, it will probably be helpful to look at the *client.py* code.

Specifically, the *multipart/form-data* data you send must have two parts:

- The first contains a text field, *request-json*, just like the rest of the API calls. The value of this field is the JSON-encoded string. It should have MIME type *text/plain*, and *Content-disposition: form-data; name="request-json"*
- The second part contains the file data, and should have MIME type *octet-stream*, with *Content-disposition: form-data; name="file"; *filename="XXX"* where XXX* is a filename of your choice.

For example, uploading a file containing the text "Hello World", the data sent in the POST would look like this:

```
--===============2521702492343980833==
Content-Type: text/plain
MIME-Version: 1.0
Content-disposition: form-data; name="request-json"

{"publicly_visible": "y", "allow_modifications": "d", "session": "XXXXXX", "allow_commercial_use": "d
--===============2521702492343980833==
Content-Type: application/octet-stream
MIME-Version: 1.0
Content-disposition: form-data; name="file"; filename="myfile.txt"

Hello World

--===============2521702492343980833==--
```

API URL:

http://nova.astrometry.net/api/upload

Arguments:

Same as URL upload, above.

5.5 Getting submission status

When you submit a URL or file, you will get back a *subid* submission identifier. You can use this to query the status of your submission as it gets queued and run. Each submission can have 0 or more "jobs" associated with it; a job corresponds to a run of the *solve-field* program on your data.

API URL:

http://nova.astrometry.net/api/submissions/SUBID

Example:

http://nova.astrometry.net/api/submissions/1019520

Arguments:

None required.

Returns (example):

```
{"processing_started": "2016-03-29 11:02:11.967627", "job_calibrations": [[1493115, 785516]],
"jobs": [1493115], "processing_finished": "2016-03-29 11:02:13.010625",
"user": 1, "user_images": [1051223]}
```

If the job has not started yet, the *jobs* array may be empty. If the *job_calibrations* array is not empty, then we solved your image.

5.6 Getting job status

API URL:

http://nova.astrometry.net/api/jobs/JOBID

Example:

http://nova.astrometry.net/api/jobs/1493115

Arguments:

- None required

Returns (example):

{"status": "success"}

5.7 Getting job results: calibration

API URL:

http://nova.astrometry.net/api/jobs/JOBID/calibration/

Example:

http://nova.astrometry.net/api/jobs/1493115/calibration/

Arguments:

- None required

Returns (example):

```
{"parity": 1.0, "orientation": 105.74942079091929,
"pixscale": 1.0906710701159739, "radius": 0.8106715896625917,
"ra": 169.96633791366915, "dec": 13.221011585315143}
```

5.8 Getting job results: tagged objects in your image

You can get either all tags (including those added by random people), or just the tags added by the web service automatically (known objects in your field).

API URL:

- http://nova.astrometry.net/api/jobs/JOBID/tags/
- http://nova.astrometry.net/api/jobs/JOBID/machine_tags/

Example:

- http://nova.astrometry.net/api/jobs/1493115/tags/
- http://nova.astrometry.net/api/jobs/1493115/machine_tags/

Arguments:

- None required

Returns (example):

```
{"tags": ["NGC 3628", "M 66", "NGC 3627", "M 65", "NGC 3623"]}
```

5.9 Getting job results: known objects in your image

API URL:

http://nova.astrometry.net/api/jobs/JOBID/objects_in_field/

Example:

http://nova.astrometry.net/api/jobs/1493115/objects_in_field/

Arguments:

- None required

Returns (example):

```
{"objects_in_field": ["NGC 3628", "M 66", "NGC 3627", "M 65", "NGC 3623"]}
```

5.10 Getting job results: known objects in your image, with coordinates

API URL:

http://nova.astrometry.net/api/jobs/JOBID/annotations/

Example:

http://nova.astrometry.net/api/jobs/1493115/annotations/

Arguments:

- None required

Returns (example, cut):

```
{"annotations": [
  {"radius": 0.0, "type": "ic", "names": ["IC 2728"],
   "pixelx": 1604.1727638846828, "pixely": 1344.045125738614},
  {"radius": 0.0, "type": "hd", "names": ["HD 98388"],
   "pixelx": 1930.2719762446786, "pixely": 625.1110603737037}
 ]}
```

Returns a list of objects in your image, including NGC/IC galaxies, Henry Draper catalog stars, etc. These should be the same list of objects annotated in our plots.

5.11 Getting job results

API URL:

http://nova.astrometry.net/api/jobs/JOBID/info/

Example:

http://nova.astrometry.net/api/jobs/1493115/info/

Arguments:

- None required

Returns (example, cut):

```
{"status": "success",
 "machine_tags": ["NGC 3628", "M 66", "NGC 3627", "M 65", "NGC 3623"],
 "calibration": {"parity": 1.0, "orientation": 105.74942079091929,
    "pixscale": 1.0906710701159739, "radius": 0.8106715896625917,
    "ra": 169.96633791366915, "dec": 13.221011585315143},
 "tags": ["NGC 3628", "M 66", "NGC 3627", "M 65", "NGC 3623"],
 "original_filename": "Leo Triplet-1.jpg",
 "objects_in_field": ["NGC 3628", "M 66", "NGC 3627", "M 65", "NGC 3623"]}
```

5.12 Getting job results: results files

Note that when using the API, you can still request regular URLs to, for example, retrieve the WCS file or overlay plots. Images submitted via the API go through exactly the same processing as images submitted through the browser interface, so you can find the status or results pages and discover the URLs of various data products that we haven't documented here.

URLs:

- http://nova.astrometry.net/wcs_file/JOBID
- http://nova.astrometry.net/new_fits_file/JOBID
- http://nova.astrometry.net/rdls_file/JOBID
- http://nova.astrometry.net/axy_file/JOBID
- http://nova.astrometry.net/corr_file/JOBID

- http://nova.astrometry.net/annotated_display/JOBID
- http://nova.astrometry.net/red_green_image_display/JOBID
- http://nova.astrometry.net/extraction_image_display/JOBID

Examples:

- http://nova.astrometry.net/wcs_file/1493115
- http://nova.astrometry.net/new_fits_file/1493115
- http://nova.astrometry.net/rdls_file/1493115
- http://nova.astrometry.net/axy_file/1493115
- http://nova.astrometry.net/corr_file/1493115
- http://nova.astrometry.net/annotated_display/1493115
- http://nova.astrometry.net/red_green_image_display/1493115
- http://nova.astrometry.net/extraction_image_display/1493115

5.13 Misc Notes

The API and other URLs are defined here:

> https://github.com/dstndstn/astrometry.net/blob/master/net/urls.py#L146

libkd documentation

6.1 C API

`kdtree_t* kdtree_build(kdtree_t* kd, void *data, int N, int D, int Nleaf, int treetype, uns`

Build a tree from an array of data, of size N*D*sizeof(data_item).

***kd*: NULL to allocate a new *kdtree_t* structure, or the address** of the structure in which to store the result.

***data*: your N x D-dimensional data, stored in N-major direction:** data[n*D + d] is the address of data item "n", dimension "d". If 3-dimensional data, eg, order is x0,y0,z0,x1,y1,z1,x2,y2,z2.

N: number of vectors

D: dimensionality of vectors

***Nleaf*: number of element in a kd-tree leaf node. Typical value** would be about 32.

***treetype*:**

- if your data are doubles, *KDTT_DOUBLE*
- if your data are floats, *KDTT_FLOAT*

For fancier options, see *kd_tree_types*.

***options*: bitfield of *kd_build_options* values. Specify one of:**

- *KD_BUILD_BBOX*: keep a full bounding-box at each node;
- *KD_BUILD_SPLIT*: just keep the split dimension and value at each node.

see *kd_build_options* for additional fancy stuff.

NOTE that this function will *permute* the contents of the *data* array!

When you're done with your tree, be sure to *kdtree_free()* it.

Example:

```
double mydata[] = { 1,1, 2,2, 3,3, 4,4, 5,5, 6,6, 7,7, 8,8 };
int D = 2;
int N = sizeof(mydata) / (D * sizeof(double));
kdtree_t* kd = kdtree_build(NULL, mydata, N, D, 4, KDTT_DOUBLE, KD_BUILD_BBOX);
kdtree_print(kd);
kdtree_free(kd);
```

`void kdtree_free(kdtree_t *kd);`

Frees the given *kd*. By default, the *kd->data* is NOT freed. Set *kd->free_data = 1* to free the data when *kdtree_free()* is called.

6.2 Python API

6.3 Code Internals

CHAPTER 7

Astrometry.net code API reference

- *WCS handling*

7.1 Flat list

- `Tan`

CHAPTER 8

Setting up a copy of the web service nova.astrometry.net

These are instructions for how to set up a web service like our http://nova.astrometry.net . This requires a bit of sysadmin savvy.

8.1 Roadmap

The code for the web service lives in the "net" subdirectory of the git repository; https://github.com/dstndstn/astrometry.net/tree/master/net . It is *not* included in the source code releases, so you'll need to *git clone* the code.

The web service has several parts:

- the web front-end. This is a Django app that we run in Apache via WSGI. Other deployment options are possible but untested.
- the database. The Django app uses a database. We use postgres. Other databases (sqlite3, mysql) would work but are, you guessed it, untested.
- front-end processing. The front-end server has to do some asynchronous processing. That is, the web-facing part of it queues submissions that are processed in a separate process, called *process-submissions.py*. On nova, we run this inside a *screen* process on the web server.
- the solve server. On nova, we have the web front-end running on one machine, and the "engine" running on another machine; the web server runs *ssh* to connect to the solve server.

8.2 Setup – web front-end

I would recommend creating a new user on your system, running the Apache server as that user, and creating a database account for that user. On nova that user is called (you guessed it), "nova".

As that user, you'll want to check out the code, eg:

```
cd
git clone https://github.com/dstndstn/astrometry.net.git nova
```

See *Running on a local machine (your laptop, say)* for how to run the web server using sqlite3 and Django's development web server.

For "real" use, you may want to set up a postgres database and run the web service via Apache.

Notice that in the *net/* directory we have a bunch of *settings_XXX.py* files. Each of these describes the setup of a deployment of the web site. We use symlinks to determine which one runs, eg, on the nova server we have:

```
ln -s settings_nova.py settings.py
```

Note also that we store the database secrets in a separate, secret SVN repository, which we check out into the directory *net/secrets*; ie, on the nova server:

```
$ ls -1 net/secrets/
django_db.py
__init__.py
```

where *__init__.py* is empty, and *django_db.py* contains:

```
DATABASE_USER = 'nova'
DATABASE_PASSWORD = 'SOSECRET'
DATABASE_HOST = 'localhost'
DATABASE_PORT = ''
```

Setting up your database is sort of beyond the scope of this document. The django docs should have some material to get you started.

The following *may* work to set up a postgres database:

```
# as root, run the "psql" program, and enter:
create role nova;
alter role nova createdb;

# as user "nova", run "psql" and
create database "an-nova";
```

Then, to initialize the database, cd into the *astrometry/net* directory and run the Django setup scripts:

```
python manage.py syncdb
python manage.py migrate
```

and test it out:

```
python manage.py runserver
```

You probably want to run the web app under Apache. The Apache configuration files for nova are not public, but your *apache2.conf* file might end up containing entries such as:

```
User nova
Group nova
Include /etc/apache2/mods-available/wsgi.load
Include /etc/apache2/mods-available/wsgi.conf
WSGIScriptAlias / /home/nova/nova/net/nova.wsgi
```

See the Django deployment docs for much more detailed setup help.

8.3 Setup – front-end processing

You need to run the *process_submissions.py* script on the web server to actually process user submissions. On nova, we run this inside a *screen* session; unfortunately this means we have to manually start it whenever the web server is rebooted. cd into the *astrometry/net* subdirectory and run, eg:

```
python -u process_submissions.py --jobthreads=8 --subthreads=4 < /dev/null >> proc.log 2>&1 &
```

8.4 Setup – solve-server processing

For actually running the astrometry engine, the web server uses *ssh* to connect to a solve server. One could probably use a local version (running on the web server) without *too* many changes to the code, but that has not been implemented yet.

When the web server wants to run the astrometry engine, it executes the following crazy code (https://github.com/dstndstn/astrometry.net/blob/master/net/process_submissions.py#L288):

```
cmd = ('(echo %(jobid)s; '
       'tar cf - --ignore-failed-read -C %(jobdir)s %(axyfile)s) | '
       'ssh -x -T %(sshconfig)s 2>>%(logfile)s | '
       'tar xf - --atime-preserve -m --exclude=%(axyfile)s -C %(jobdir)s '
       '>>%(logfile)s 2>&1' %
       dict(jobid='job-%s-%i' % (settings.sitename, job.id),
            axyfile=axyfn, jobdir=jobdir,
            sshconfig=settings.ssh_solver_config,
            logfile=logfn))
```

So it first sends the job id, then a *tar* stream of the required input files, and pipes that to *ssh*. It streams the error output to the *logfile*, and pipes the standard out to *tar* to receive the results. It's sweet.

Notice that *sshconfig* string there, which come from the *settings.py* file. For nova, for example, *ssh_solver_config* = *'an-nova'*. We then have an entry in the *nova* user's *~/.ssh/config* file:

```
Host an-nova
Hostname solveserver.domain.org
User solve
IdentityFile ~/.ssh/id_nova_backend
```

And, naturally, we use SSH keys to automate the login.

On the solve server, the *solve* user has an entry in *~/.ssh/authorized_keys* for the *id_nova_backend.pub* public key, that tells the *ssh* server what should be run when that key is used to log in:

```
# id_nova_backend
#
command="cd /home/solve/nova/blind; ../net/testscript-astro",no-port-forwarding,no-X11-forwarding,no-
```

That script (https://github.com/dstndstn/astrometry.net/blob/master/net/testscript-astro) first reads the job id, creates a working directory for the job, uses *tar* to receive the input files, and then runs the *astrometry-engine* program to actually run the requested job. Finally, it uses *tar* to bundle up and send back the results.

(Note that, at present, the *testscript-astro* script still tries to run the *astrometry-engine* by its old name, *backend* ... we haven't updated that script in a while. That script also includes hard-coded paths, so you will have to edit for your site.)

Once-Asked Questions

9.1 Q: I don't have *numpy*. What do I do?

"import error: no module named numpy"

9.2 A: Disable things that require numpy.

Some parts of the code need the "numpy" python package. To disable things that need numpy:

```
solve-field --no-fits2fits --no-remove-lines --uniformize 0  [....usual arguments...]
```

9.3 Q: Is there a way to plot a grid of RA and Dec on the images?

9.4 A: Yes

You'll have to run the "plot-constellations" program separately. For example, if you have an image 1.jpg and WCS 1.wcs:

> jpegtopnm 1.jpg | plot-constellations -w 1.wcs -o grid.png -i - -N -C -G 60

will plot an RA,Dec grid with 60-arcminute spacings. Unfortunately they're not labelled...

[Note, see *plotann.py* also for more annotation options.]

9.5 Q: Is there a way to get out the center of the image (RA,Dec) and pixel scale of the image?

9.6 A: Yes, with the *wcsinfo* program

Yes, run the "wcsinfo" program on a WCS file – it prints out a bunch of stats, in a form that's meant to be easy to parse by programs (so it's not particularly friendly for people). "ra_center" and "dec_center" (in degrees) and "pixscale" (in arcsec/pixel) are what you want.

9.7 Q: Is there a way to plot N and E vectors on the image?

9.8 A: Not yet.

9.9 Q: Is there a way to plot a list of your own objects on the image by inputing RA,Dec?

9.10 A: Check out *plotann.py*, or try these older instructions...

Yes – but it's roundabout...

First, project your RA,Dec objects into pixel x,y positions:

```
wcs-rd2xy -w 1.wcs -i your-objs.rd -o your-objs.xy
```

Then plot them over the image (or annotated image). There's not currently a way to label them.

:

```
pngtopnm grid.png | plotxy -i your-objs.xy -I - -x 1 -y 1 -s X -C green -b black > objs.png
```

The "-x 1 -y 1" compensate for the fact that FITS calls the center of the first pixel (1,1) rather than (0,0).

9.11 Q: Would your code work on all-sky images?

9.12 A: Not very well

We assume a TAN projection, so all-sky images typically don't work, but it should certainly be possible with a bit of tweaking, since all-sky is really a much easier recognition problem! One thing you can try, if your image is big enough, is to cut out a small section near the middle.

9.13 Q: I want to build an index from my own catalog. How do I proceed?

9.14 A: See genindex

CHAPTER 10

Astrometry.net code structure

This is meant to be an introduction to what parts of the codebase run during a solve.

10.1 *blind/solve-field.c*

- parses command-line args
- chooses output filenames
- downloads URL inputs
- decides whether inputs are FITS x,y lists or images
- calls *augment_xylist()* to produce *.axy file
- runs *astrometry-engine* to actually do the solve
- produces plots (*-objs.png, *-ngc.png, etc)

10.2 *blind/augment-xylist.c*

A field to solve is encapsulated in an "axy" file, which is a FITS binary table containing X,Y star positions, and well as FITS header cards that describe the command-line arguments and other information we have about the image. "axy" is short for "augmented x,y list", and we often abbreviate "x,y list" to "xylist". *augment-xylist.c* creates these "axy" files.

- run *image2pnm.py* to uncompress and convert images to PNM.
- for non-FITS images, run *ppmtopgm* and *an-pnmtofits* to produce FITS
- run *image2xy* (or SourceExtractor) to generate list of (x,y) star coordinates (xylist)
- for FITS files, run *fits2fits.py* to clean file
- run *removelines.py* to remove lines of sources from the xylist
- run *resort_xylist()* to sort by a combination of brightness and background
- run *uniformize.py* to select a spatially uniform subset of stars
- add headers to xylist to create axy file

10.3 *blind/engine-main.c*

This is the *astrometry-engine* executable.

- reads *astrometry.cfg* file
- finds index files
- reads axy file
- runs *engine_run_job()* to actually do the solve

10.4 *blind/engine.c*

engine_run_job()

- parses axy file
- based on range of image scales, selects index files to use
- calls *blind_run()*

10.5 *blind/blind.c*

blind_run()

Runs a set of fields with a set of index files.

- reads xylist
- runs any WCS headers to verify (solve-field –verify)
- depending on whether running with *inparallel* or not, loads one or all index files and calls *solve_fields()*
- records good matches that are found (writes WCS, rdls, match, corr files)

solve_fields()

- calls *solver_preprocess_field()*
- calls *solver_run()*

10.6 *blind/solver.c*

Runs a single field with a set of index files.

solver_run()

- load index files
- compute scale ranges of field and index files
- looks at pairs of stars A,B forming the “backbone” of the quadrangle, precomputing geometry and deciding which stars can be C,D
- adds one star at a time, forming all quadrangles where that star is A,B or C,D, and for each index, calls *add_stars()*

add_stars()

- select stars that will form the quadrangle (or triangle or pentagon)
- calls *TRY_ALL_CODES() = try_all_codes()*

try_all_codes()

- tests permutations of the C,D stars that are valid (satisfy Cx<Dx constraints), with different parities
- calls *try_all_codes_2*

try_all_codes_2()

- tries different permutations of A,B stars
- calls *try_permutations()*

try_permutations()

- recursive
- tries different permutations of C,D stars, checking for cx <= dx constraint
- searches code KD-tree for matches, calls *resolve_matches()* if found

resolve_matches()

- given a code match between a field quadrangle and the index,
- looks up the index star numbers forming that quadrangle (in the quadfile)
- retrieves the index star RA,Dec positions for these stars (in the star KD-tree)
- fits a TAN projection to the matched quadrangle
- calls *solver_handle_hit()*

solver_handle_hit()

- calls *verify_hit()* to confirm the match
- if matched, calls *solver_tweak2()* to compute SIP coefficients

10.7 *blind/verify.c*

verify_hit()

- searches for stars within the field in the star KD-tree
- calls *real_verify_star_lists()* to do the model comparison between true match and false match.

CHAPTER 11

Nova.astrometry.net orientation

Getting started:

- you'll need an account on `astrometry.net` (for svn), and to have your ssh public key added to the authorized_keys for `nova@oven.cosmo.fas.nyu.edu`
- can you ssh in as `nova@oven.cosmo.fas.nyu.edu`?
- can you `svn checkout` the Astrometry.net code? (see svn_checkout)

11.1 Nova

As for the nova website code: if you svn checked out the Astrometry.net code, you should have a `net` directory – that's where the server-side code that runs the nova.astrometry.net site lives. It's written in python using the Django framework, which provides nice web and database routines.

We actually run three copies of the nova website:

- supernova.astrometry.net – "trunk", for testing and development
- staging.astrometry.net – test-flight area when we think we have a releaseable version.
- nova.astrometry.net – the production site. This runs off an "svn tagged" version – not trunk

The web sites are run via the apache web server. You probably won't have to deal with that side at all. Each one runs out of a directory within `nova@oven`'s home directory. They're called (this may shock you), "supernova", "staging", and "nova". There is a server-side process that has to be running in order for the web site to do anything with images that people submit. That's called `process_submissions.py` (you can see the code in `net/process_submissions.py`), and for each site we run it in a `screen`. `screen` is an old-school utility that lets you create a "virtual terminal" that you can "attach" to and "detach" from. It's a handy way of keeping a program running on a server when you're not logged in. Anyway, if you're logged in as `nova@oven` you can do:

```
screen -ls
```

to see the currently-running screens;

```
nova@oven:~$ screen -ls
There are screens on:
    20579.supernova (03/18/2012 08:15:43 PM)        (Detached)
    27698.nova      (03/14/2012 06:11:25 PM)        (Detached)
2 Sockets in /var/run/screen/S-nova.
```

which says there are two screens, `supernova` and `nova`.

You can "attach" or "rejoin" with:

```
screen -R supernova
```

and you should see "process_submissions.py" doing its thing:

```
Checking for new Submissions
Found 0 unstarted submissions
Checking for UserImages without Jobs
Found 0 userimages without Jobs
Submissions running: 0
Jobs running: 0
```

anyway, to "detach" from that screen, do:

```
ctrl-a d
```

Ok, so coming back to the code, let's find out where the data live.

Each site has a `settings` file. So, as `nova@oven`:

```
nova@oven:~$ cd supernova/net
nova@oven:~/supernova/net$ ls -l settings.py
lrwxrwxrwx 1 nova nova 21 2011-06-09 19:16 settings.py -> settings_supernova.py
```

So in the `supernova` directory, `settings.py` is a symlink to `settings_supernova.py` . Let's look in there:

```python
# settings_supernova.py
from settings_common import *

DATABASES['default']['NAME'] = 'an-supernova'

LOGGING['loggers']['django.request']['level'] = 'WARN'

SESSION_COOKIE_NAME = 'SupernovaAstrometrySession'

# Used in process_submissions
ssh_solver_config = 'an-supernova'
sitename = 'supernova'
```

oh, it hardly has anything. But at the top it imports everything from settigs_common.py. Looking there, we see:

```python
...
WEB_DIR = os.path.dirname(astrometry.net.__file__) + '/'
DATADIR = os.path.join(WEB_DIR, 'data')
...
```

so it figures out the directory it is currently running in based on the location of the `astrometry.net` python package (`WEB_DIR`), which for supernova will be:

```
/home/nova/supernova/net
```

and the `DATADIR` is that + `data`. If you look in that `data` directory, you'll see:

```
nova@oven:~/supernova/net$ ls data
00  08  12  19  25  2c  34  3a  41  49  50  57  60  67  6f  74  7e  8a  91  9a  a3  ad  b4  bf  c6
03  0b  14  1a  27  2e  36  3b  43  4a  52  59  62  6a  70  75  85  8b  92  9d  a6  ae  b5  c0  c7
04  0c  15  20  29  2f  37  3d  44  4b  53  5a  63  6b  71  76  86  8c  94  9e  aa  b0  b7  c3  ca
06  0d  16  23  2a  32  38  3e  45  4c  55  5b  65  6c  72  79  87  8d  98  9f  ab  b1  b9  c4  cb
07  11  18  24  2b  33  39  3f  46  4e  56  5e  66  6d  73  7a  89  8f  99  a2  ac  b3  ba  c5  d1
```

ok, a bunch of directories. What's in them?

```
nova@oven:~/supernova/net$ find data/00
data/00
data/00/32
data/00/32/84
data/00/32/84/00328489cbdfce1a99ebbf1078c95669e39fa8a7
```

```
nova@oven:~/supernova/net$ file data/00/32/84/00328489cbdfce1a99ebbf1078c95669e39fa8a7
data/00/32/84/00328489cbdfce1a99ebbf1078c95669e39fa8a7: JPEG image data, JFIF standard 1.01
```

And check this out:

```
nova@oven:~/supernova/net$ sha1sum data/00/32/84/00328489cbdfce1a99ebbf1078c95669e39fa8a7
00328489cbdfce1a99ebbf1078c95669e39fa8a7  data/00/32/84/00328489cbdfce1a99ebbf1078c95669e39fa8a7
```

So the files are named according to a cryptographic hash of their contents (SHA-1), and sorted into subdirectories according to the first three pairs of hexadecimal digits:

```
AA/BB/CC/AABBCC....
```

(We sort them into subdirectories like that to avoid having a huge number of files in a single directory.)

11.2 Running on a local machine (your laptop, say)

Prerequisites:

- django
- python-openid
- django-openid-auth
- South
- PIL
- *sqlite3*
- simplejson

Once you've got the code checked out, you need to symlink one of the `settings_*.py` files to `settings.py`:

```
ln -s settings_test.py settings.py
```

for local testing, `settings_test.py` is the best bet, since it uses a local sqlite3 database rather than a real external database.

Initialize the database:

```
python manage.py syncdb
python manage.py migrate
```

Then you can run the Django web server by running:

```
python manage.py runserver
```

And then go to: http://localhost:8000

CHAPTER 12

Nova.astrometry.net: Data model

- Submission: solve parameters + DiskFile upload -> set of UserImages
- Job: UserImage + Astrometry.net processing parameters + Calibration result
- UserImage: an Image submitted by a User. (Multiple Users could submit the same Image).
- DiskFile
- Image (DiskFile + image-specifics)
- Calibration

Backups of Astrometry.net stuff

- **Subversion** (svn) repo. dstn maintains a mirror (at `dstn@astro.cs.toronto.edu:svn-backup-astrometry`) using the code in `trunk/scripts/svnsync` (here). It is updated and md5sum-checked hourly, via a cron job on `astro.cs.toronto.edu`:

  ```
  # Sync hourly
  @hourly   /u/dstn/svn-backup-astrometry/sync.sh
  # Check hourly, at 5 minutes past (to let the sync finish)
  5 * * * *   /u/dstn/svn-backup-astrometry/sync.sh && /u/dstn/svn-backup-astrometry/check.sh
  ```

  That directory also contains dumps of series of 1000 revs, named like `svn-backup-10k-to-11k.dump.bz2`

- **Trac**. Backed up in `svn:secure/trac`. The database gets dumped and committed daily at 4am by a cron job by `astrometry@astrometry.net`. Attachments do **not** get automatically added.

- **Forum**. Nothing?

- **Live**. dstn has backups in `dstn@broiler:/data1/live`. These are monthly tarballs, created manually as part of the perpetual data-juggling on oven.

  Hogg has an independent backup of this.... RIGHT?

- **Nova**. dstn has a snapshot (2012-06-05) of the nova/net/data directory at `dstn@broiler:/data2/dstn/BACKUP-nova/nova-data`:

  ```
  cd /data2/dstn/BACKUP-nova
  rsync --progress -Rarvz nova:nova/net/./data/ nova-data
  rsync --progress -Rarvz nova:nova/net/jobs/./ nova-jobs
  ```

  (There is a snapshot dump of the nova database there too.)

  There is a snapshot (snapshot-2013-10-14.tgz) on bbq, as well as daily backups, in `bbq:/export/bbq1/nova/nova-data`:

  ```
  # Backup nova@broiler nova.astrometry.net data
  0 4 * * * rsync -ar /data2/nova/nova-data nova-data-rsync:nova-data-backup/
  0 4 * * * pg_dump an-nova | ssh nova-data-rsync 'cat > nova-data-backup/nova-database/an-nova.sq
  ```

CHAPTER 14

Astrometry.net python API reference: astrometry.util

14.1 WCS handling

Other places:

- This document on the web
- home page
- web service

Internal docs:

- *Nova.astrometry.net orientation*
- *Backups of Astrometry.net stuff*

Indices and tables

- genindex
- modindex
- search