Astrometry.net

*Input an **image** and we'll give you back astrometric calibration meta-data, plus lists of known objects falling inside the field of view.*

Web site: https://astrometry.net

- Usage option 1
  - Web service: upload / send an image and get the result
- Usage option 2
  - Download the code, compile and install
  - Download index files
  - Run `solve-field` with parameters to speed up the process

Index files

- Depending on the field size of the image, different index files must/can be used
- Index files related info read from a configuration file
- Index catalogues can have just the positional information or any additional info
- Default for images > 1 deg: Tycho-2 catalogue
- Default for images < 1 deg: Tycho-2 + Gaia-DR2
- For (almost) any image size: 2MASS
- Additional catalogues available, and can also be built by the user

Astrometry.net

*Input an **image** and we'll give you back astrometric calibration meta-data, plus lists of known objects falling inside the field of view.*

Source detection methods

1. Built-in `image2xy` extractor
2. Bertin's source extractor (SExtractor)
   - Can pass PATH to executable and CONFIG file

`solve-field` useful parameters to speed-up the process

- Guessed position
- Field size
- Pixel scale – given as min/max range

`loastrom.sh` bash script to process Loiano images uses this astrometry.net command

```
solve-field --config $CNF --depth $NSTARS --objs $NSTARS --scale-low $SCALE_L --scale-high $SCALE_H
  --radius $SEARCH_R --ra $radeg --dec $dedeg
  --scale-units arcsecperpix --downsample 2 --source-extractor-path /usr/local/bin/sex
  --source-extractor-config /usr/local/astrometry/etc/def.sex
  --x-column X_IMAGE --y-column Y_IMAGE --sort-column MAG_AUTO
  --overwrite --no-plots --sort-ascending
  $FITSFILE
```

Astrometry.net

*Input an **image** and we'll give you back astrometric calibration meta-data, plus lists of known objects falling inside the field of view.*

- `loastrom.sh` bash script runs in a CygWin instance
- It can be invoked directly form Windows using a script: `loastrom.cmd`

```
@echo off
setlocal

set _CYGBIN=C:\cygwin64\bin\

:: Resolve ___.sh to /cygdrive based *nix path and store in %_CYGSCRIPT%
for /f "delims=" %%A in ('%_CYGBIN%cygpath.exe "%~dpn0.sh"') do set _CYGSCRIPT=%%A

:: Throw away temporary env vars and invoke script, passing any arguments that were passed
endlocal & %_CYGBIN%bash --login "%_CYGSCRIPT%" %*
```

loastrom.sh script

Usage

```
loastrom.sh image.fits [RA_deg Dec_deg]
```

If RA/Dec not passed then try to read them from the header keywords CRVAL1/2

Default settings

- NSTARS = 80
- SCALE_L = 0.05
- SCALE_H = 0.10
- SEARCH_R = 0.5
- do_overwrite = true ⟹ if existing output file(s) can be overwritten
- clean = true ⟹ if astrometry.net produced auxiliary files should be removed
- coords = true ⟹ if user given coordinates is assumed
- rename = true ⟹ if rename output file (default extension is .new) and move it to new dir.

- DIROUT = $HOME/Astrometry_out ⟹ output images directory

loastrom.sh script

Procedure

- Read image size keywords NAXIS1/2 and compute image centre pixel
- Try to solve the field using the Gaia-DR2 index files, if it fails try with 2MASS
- If successful add/update header keywords of the output image

RA [deg]	⟹ RA of the image centre
DEC [deg]	⟹ DEC of the image centre
RAdiff [sec]	⟹ RA offset (Orig-New)
DECdiff [arcsec]	⟹ DEC offset (Orig-New)

Note: process output go to a log file.

Source code is here.

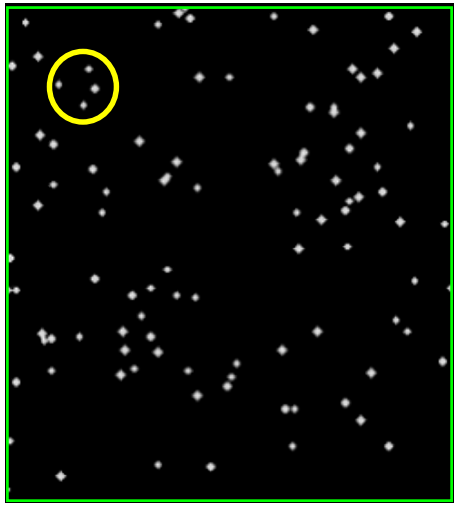
… from the Astrometry.net presentation

Solving the search problem: (Inverted) Index of Features

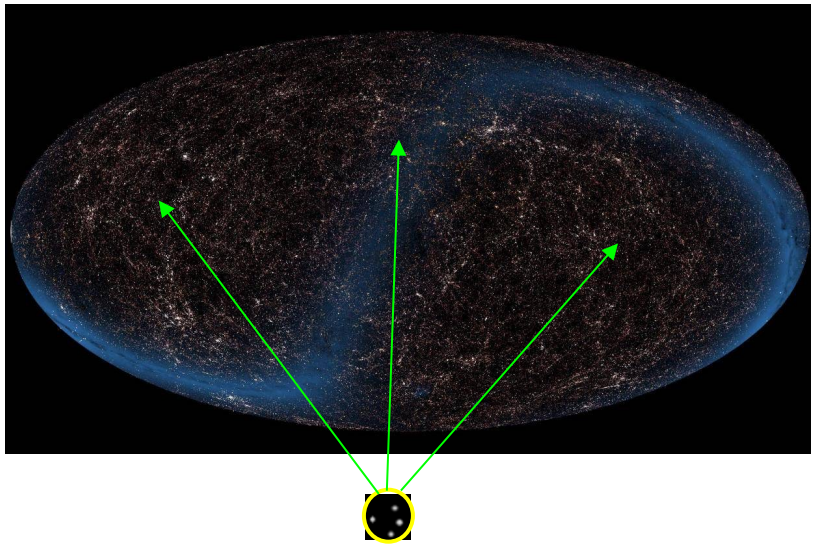
Even if we can succeed in finding a good robust matching algorithm, there is still a huge search problem.

Which proposed location should we match to?

- To solve this problem, we will employ the classic idea of an "inverted index".
- We define a set of "features" for any particular view of the sky (image).
- Then we make an (inverted) index, telling us which views on the sky exhibit certain (combinations of) feature values.



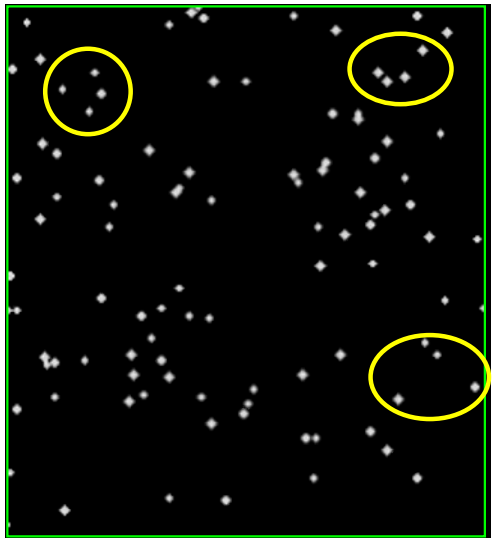
- This is like the question:
  Which web pages contain the words "machine learning"?



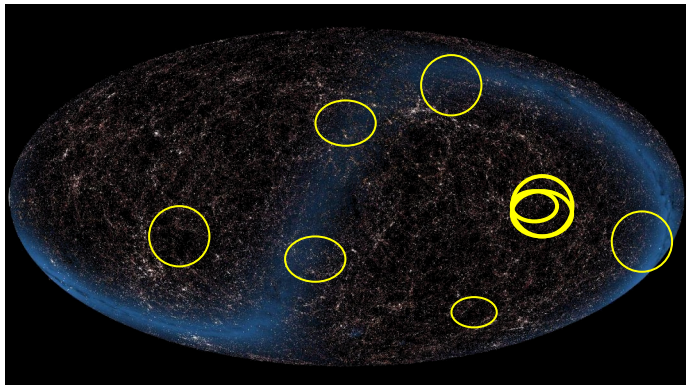
… from the Astrometry.net presentation

Solving the search problem: (Inverted) Index of Features

- When we see a new test image, we compute which features are present, and use our inverted index to look up which possible views from the catalogue also have those feature values.
- Each feature generates a candidate list in this way, and by intersecting the lists we can zero in on the true matching view.



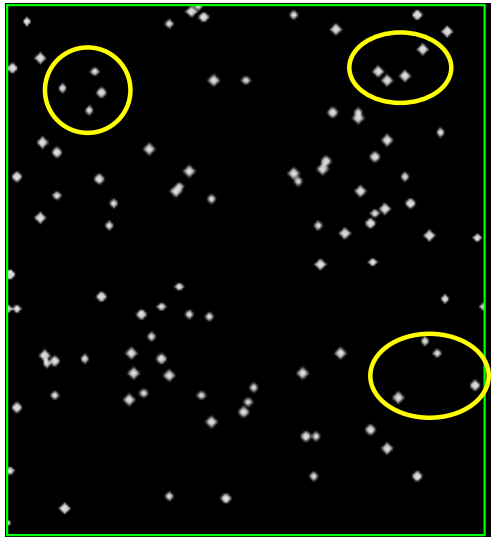
- The features in our inverted index act as "hash codes" for locations on the sky.
- The idea of an inverted index is that is pushes the computation from search time back to index construction time.



… from the Astrometry.net presentation

Quads as Robust Features

- In simple search domains like text, the inverted index idea can be applied directly.
- However, in our star matching task, the features we chose must be invariant to scale, rotation and translation.
- They must also be robust to small positional noise.
- Finally, there is the additional problem of distractor & dropout stars.

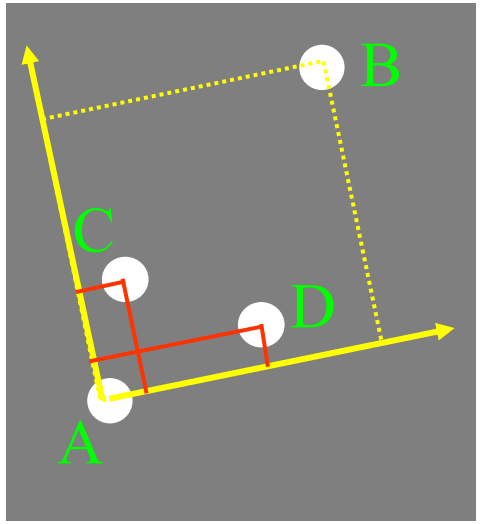


The features we use are the relative positions of nearby quadruples of stars

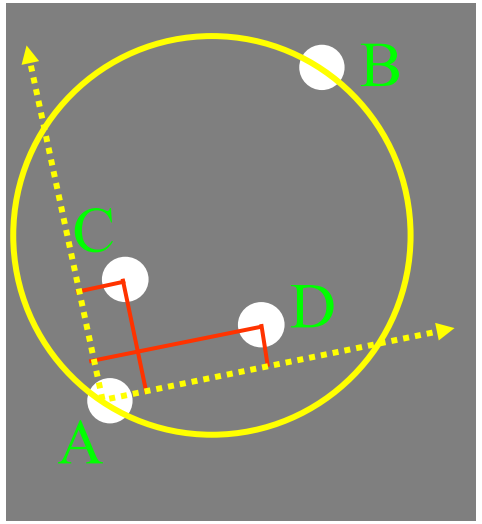
… from the Astrometry.net presentation

Quads as Robust Features

- We encode the relative positions of nearby quadruples of stars (ABCD) using a coordinate system defined by the most widely separated pair (AB).
- Within this coordinate system, the positions of the remaining two stars form a 4-dimensional code for the shape of the quad.
- Swapping AB or CD does not change the shape but it does “reflect” the code, so there is some degeneracy.



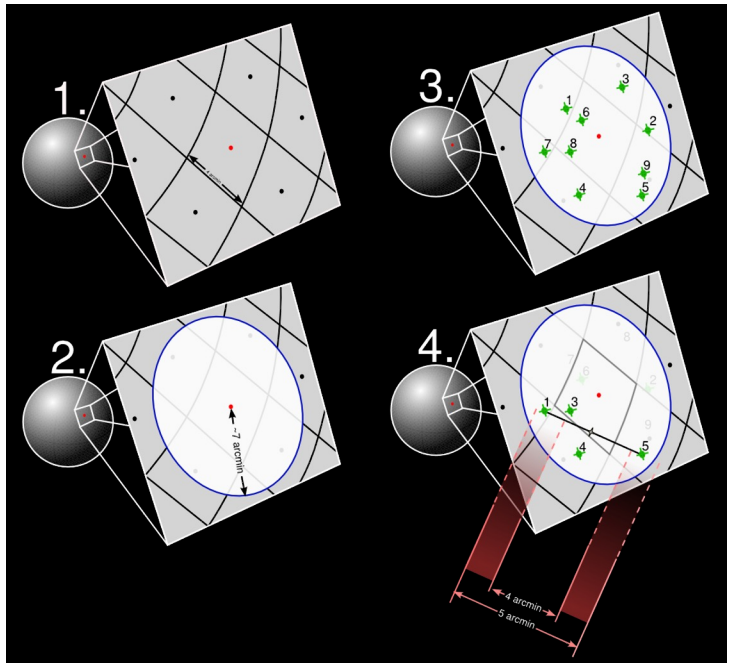
- This geometric hash code is invariant to scale, translation and rotation.
- It also has the property that if stars are uniformly distributed in space, codes are uniformly distributed in 4D.
- We compute codes for most nearby quadruples of stars, but not all; we require C&D to lie in the unit circle with diameter AB.

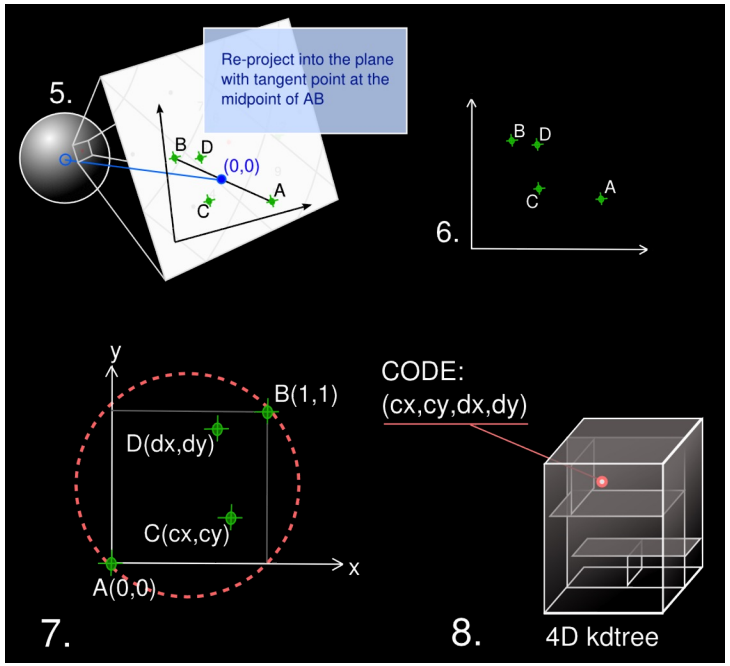


… from the Astrometry.net presentation

Building the index

- Start with the catalogue; build a kdtree on the 3D object positions.
- Place a fine healpix grid on the sky. Within each pixel, identify a valid quad whose size is near the target scale for the index.
- Compute 4D codes for those quads; enter them into another kdtree remembering their original locations. This is the index.





… from the Astrometry.net presentation

Solving a new image

- Identify objects (stars+galaxies) in the image bitmap and create a list of their 2D positions.
- Cycle through all possible valid* quads (brightest first) and compute their corresponding codes.
- Look up the codes in the code KD-tree to find matches within some tolerance; this stage incurs some false positive and false negative matches.
- Each code match returns a candidate position & rotation on the sky. As soon as 2 quads agree on a candidate, we proceed to verify that candidate against all objects in the image.