Questions About Star_Match

What does star_match do?

Two things:

1. Star_match identifies some of the stars in an image. Star_match works from a catalog (more on this below), and will match stars in the image with stars in the catalog. Depending on sky conditions (and what I'm trying to image), it's possible for *either* the image or the catalog to contain more stars than the other. (Although, *usually* there are more stars in the image than in the catalog. It's only for short exposures or bad weather that the image has fewer stars than the catalog. However, it's still important for star_match to work correctly when that happens.)

2. Star_match creates a coordinate system transformation function that converts back and forth between pixel coordinates (row, column) and sky coordinates (declination, right ascension) for that image.

What input data does star_match use?

- **Image rotation angle:** At the time an image is captured from the camera, one of the pieces of metadata that is attached to the image is called ROTATION and either has the value 0 or the value π. This rotation angle is taken as an *approximate* measure of the camera's orientation relative to north. In half of the sky, my telescope mount holds the telescope/camera flipped upside down relative to the other half of the sky. This is captured in that ROTATION metadata. *It does not describe the precise orientation of the camera. That is part of what star_match will figure out during its pattern matching.*

- **Pixel scale:** Again, at the time an image is captured from the camera, another piece of metadata is attached with an approximate measure of the amount of sky that each camera pixel covers. This data is spread across two metadata variables, CDELT1 and CDELT2, which describe the span of each pixel in the x-direction and y-direction, respectively. However, I will never (again) use a camera with non-square pixels, so CDELT1 and CDELT2 are always the same. A typical value for CDELT1 is 1.52 arcseconds per pixel. Again, this number is approximate, since with the telescope type that I'm currently using, changing the focus will change the pixel scale, and there's no way to know by how much until star_match has done its thing.
- **Observed Object:** This is passed (by *name*) to star_match when it is run. Star_match will use the star's name to look up the astronomical coordinates of the assumed near-center of the image. Star_match works correctly as long as the *actual* coordinates of the center are within about ½ degree of the assumed near-center. Note that this is actually a bit messy for Pluto, since Pluto moves (more on this later).

- **A star catalog:** I start with the Guide Star Catalog - Astrographic Catalog/Tycho (GSC-ACT). That catalog covers the entire sky. For each star that I observe, I create an extract from GSC-ACT that covers just that small patch of sky. I then add (by hand) additional stars to the extract to cover important things that are missing from GSC-ACT. I sometimes will also delete stuff from GSC-ACT if there are catalog errors in that sky region. Because Pluto moves, the star catalog extract that I call "pluto" is re-calculated from the big GSC-ACT every night. Although it isn't used for Pluto/Charon, the star catalogs also usually contain precision brightness and color information that is used to establish stable brightness reference points for my variable star

observing. This information comes from yet another source (the AAVSO) that I merge with the GSC-ACT data during creation of the catalog extracts.

- **An image starlist:** The star_match program doesn't actually use the image itself. Instead, it starts with the list of stars that was previously created from the image by the program find_stars.
- **An optional bias file:** (We will never use this for Pluto/Charon.) Every once in a while I need better positional accuracy (astrometry) than is possible from the GSC-ACT catalog. Star_match will accept an optional bias input file that contains star-by-star adjustments to published catalog locations.

What coordinate systems are found in star_match?

- Pixel coordinates: This is a star's position as measured using the camera's row and column coordinate system. Sometimes, a dedicated C++ class is used for this coordinate system (named PCS).
- Astronomical coordinates: This is a star's position as measured using two angles: declination angle and right ascension. Whenever astronomical coordinates are used in star_match, it is done using the J2000 epoch coordinate system. (Telescope pointing is actually done using epoch of the day.) The C++ class named DEC_RA is used for this coordinate system.
- Tangent Coordinate System: The C++ class named TCS plays an important role in star_match. It's used for simplification to address two non-linearities with using declination and right ascension throughout star_match: lines of constant declination are not straight lines on the camera's sensor (same way lines of latitude on the Earth trace out circles around the globe) and a given right ascension angle (say, one degree, for example) covers a *different* amount of sky depending on your declination. Although the math to correct for these nonlinearities is well defined, star_match does a lot of math and the number of trigonometric functions that would have to be evaluated would simply take too long. And so star_match makes extensive use of a clean, flat, rectangular sky coordinate system that is tangent to the celestial sphere at a point called the TCS's origin. Distances in TCS are measured in arcseconds (or radians).

What algorithm does star_match use?

Star_match does the following:

- It creates two local star lists. One contains the catalog stars and one contains the image's stars. The 10 brightest stars in the image are flagged (after sorting the image stars by brightness), and the generally-bright stars in the catalog are flagged.
- Each of the 10 brightest stars in the image is paired up with each of the bright stars in the catalog. This will typically create several hundred possible pairings.
- For each one of those pairings, each of the remaining 9 brightest stars in the image is paired up with each of the remaining bright stars in the catalog. This creates a candidate "pair of pairs" set (two image stars paired with two catalog stars). Some quick reasonableness checks are run to discard unreasonable pair-of-pairs combinations as quickly as possible. However, several thousand (or tens of thousands) of candidate pair-of-pairs can't be immediately dismissed.

- (In the source code, you'll find references to the pair-of-pairs using the four variables *image_ref*, *image_alt*, *cat_ref*, and *cat_alt*.)
- For each of the candidates that survive that initial pruning, a TCS is defined that contains precise rotation angle and pixel scaling data. Each catalog star and each image star is translated into TCS coordinates and they are checked against each other to count matches. A "slop factor" in allowed at this point. The total number of matches is calculated, and this becomes the "score" for that candidate pair-of-pairs pairing.
- The scores are used to find the best pair-of-pairs. The resulting matchings (which will typically have a few dozen matches found between image and catalog) are then sent to a function called "high_precision_fixup()". This function performs a more formal calculation of the coordinate system transformation function that is based on minimizing least-squares errors between the measured star positions in the image and the catalog positions. (This is also where predefined position bias numbers are added in.) The function is performed twice. The first time finds a good coordinate system transformation function. After that is established, it will go through and do a second pass of matching image stars to catalog stars, using more stringent matching criteria. This will usually result in many more matches than happened the first time. The additional matches are used going into the second cycle of calculating the coordinate system transform. (Sometimes the least-squares estimator will have over a 100 matches contributing to the calculation of the coordinate system transform.)

Pluto is moving across the sky (albeit slowly). How does this affect the process?

Chuckle. It's kind of ugly. First rule was:

Make no change to star_match to accommodate Pluto-Charon.

And that really only leaves one other place to influence things: the catalog file. I make the assumption that Pluto's motion during any one night is not enough to significantly affect star_match. (Pluto moves about 20 arcseconds per 24 hour day.) And so, the second rule is:

Do not try to correlate Pluto-Charon. It will be an unmatched star when star_match finishes.

The following process is used each night (this is all contained in the Python script pluto_setup.py, which I've included):

- A current position for Pluto is fetched online. (The accuracy of this position is unknown, but probably only accurate to within 1 or 2 arcminutes.)
- Pluto is treated by my system as a "star" with a position listed in my list of stars that I care about, "star_list.txt". The existing entry for Pluto in star_list.txt is deleted.
- A new entry for Pluto is added to the end of star_list.txt, using the current Pluto position.
- The existing catalog extract for Pluto is deleted, and a new extract catalog for Pluto is created.

What are all the pieces of this process in the source code?

Most of my code is in C++. Most programs make extensive use of a series of four custom libraries:

- IMAGE_LIB: This implements key classes for working with images. Classes include
  - "Image",
  - "ImageInfo" (metadata attached to each image),

    - "IStarList" (list of stars also attached to each image),
    - "BadPixels" (management of known bad pixels in the camera)
    - "Filter" (key definitions associated with the photometric filters I use with variable star observations)
    - "WCS" (handling of coordinate system transformations embedded in image metadata)
- DATA_LIB: This implements classes used for databases and for some odds and ends
    - BrightStar (a list of bright stars in the sky – naked eye visible)
    - HGSC (the Guide Star Catalog)
    - The TCS and PCS coordinate systems are implemented here
    - SystemConfig (current optical and mechanical configuration of the telescope)
- SESSION_LIB: This provides classes used to manage overnight imaging sessions
    - Session (overall sequencing of observations during a session)
    - Scheduler (optimizes a schedule for a session using genetic optimization)
    - Schedule (what will be done during a session)
    - Strategy (how each star will be handled)
    - ExposurePlan (manages exposure times, camera gain, and exposure quantities)
    - FocusManager (manages telescope focus during an overnight session)
- REMOTE_LIB: Provides classes to interface with the camera, mount, focuser, …
    - Julian, DEC_RA, ALT_AZ: Utility classes to handle time and position
    - Camera_API: Interface to the camera
    - Scope_API: Interface to the telescope mount
    - Various messaging interfaces to handle message exchanges between "outside" computer and "inside" computer

Multiple files implement the star_match program:

- star_match.cc – This is the main program. It handles command-line options.
- correlate1.cc – This is the heart of the correlation processing.
- high_precision.cc – This implements the high-precision minimization of total correlation variance using linear least-squares.
- aperture_phot.cc – This is a quick-and-dirty implementation of aperture photometry to support star_match. Its only use is to support finding the 10 brightest stars in the image.