Automatic Star-tracker Optimization Framework

Andrew Tennenbaum
The State University of New York at Buffalo
aztennen@buffalo.edu

Faculty Advisor: John Crassidis
The State University of New York at Buffalo

ABSTRACT

The problem of creating a fast, robust startracker optimized for a specific camera is both of great importance, and great difficulty for smallsat missions. A cycle of algorithm development, camera calibration, validation, and database tuning typically requires several iterations, and consumes a great deal of time and effort. Unfortunately this is often the only option for smallsat missions, where financial constraints put commercial star trackers out of reach.

In this paper, we present a framework which automates the process from end to end. Given a set of sample star images taken with a given camera at the desired exposure time, we can automatically calculate all the parameters, generate the optimal startracker database, and perform testing and validation. Depending on the camera, this results in a startracker with performance that has been shown to be on par with, or in many cases better than existing commercial startrackers. A startracker produced using this method has been tested and will soon have flight heritage, at which point this software will be made available to the smallsat community via an open source license.

Introduction:

The problem of identifying orientation given an image of the stars is of great importance in both astronomy and spacecraft guidance. Up to this point in time, the two communities have developed different algorithms to meet different sets of constraints.

In the astronomical community, images can come from many different sources and CPU time is virtually unlimited. State of the art astronomy-type algorithms such as astrometry.net are able to not only automatically solve for orientation, but also camera field of view, photon sensitivity, signal to noise ratio, and first order lens distortion parameters. On the other hand, in spacecraft guidance, it is necessary to rapidly solve images with limited computational resources. The tradeoffs needed in order to meet the requirements of this class of algorithms means that additional time and/or money has to be put into improving the performance of the camera itself. This poses a problem for nanosatellite missions, where both time and money are in short supply.

The framework we have developed combines the best of both worlds, enabling the rapid development of low cost, high performance star-trackers

Framework components:

The framework we have developed consists of three components:

1. A novel state of the art high speed star-tracker algorithm which incorporates relative brightness and scale information.
2. An optimal star-tracker database generation framework which uses astronomy-type algorithms to estimate calibration parameters from a set of sample images.
3. An automatic test suite which validates the star-tracker under a variety of test conditions, and determines expected slew tolerance and sky coverage.

Star tracker algorithm:

Current state of the art, fast star-tracker algorithms can trace their origins back to the work of Dr. Daniele Mortari from Texas A&M, who developed the Pyramid Star Identification Technique for Lost in Space star identification[1]. Lost in Space refers to the situation where there is no prior knowledge of satellite orientation. In essence, the pyramid technique works as follows:

1. The positions of stars in the image are determined by the centroiding technique. This is detailed in the image preprocessor section.
2. Possible four star constellations are generated, and checked to match a reference constellation database
    a. A pilot star is selected from an image
    b. Three reference stars are selected from the image
    c. The distance between the pilot star and the three reference stars are looked up in a "K-vector" database (a variant of the hash-table devised by Dr. Mortari and commonly used in the spacecraft guidance community)
    d. If a constellation has been uniquely identified, solve for spacecraft orientation, and identify the remaining stars by looking them up in a star database based on their position in the sky. Otherwise select the next constellation in the image
    e. Repeat until identification has occurred, or all combinations have been exhausted.

In the original paper, this technique was tested with a 12 degree field of view, and a database consisting of stars of magnitude greater than 5.5 magnitude. The algorithm is fine in this case, however the constellation database size quickly balloons and starts producing false positives when dimmer stars are included. It is necessary to include dimmer stars in order to ensure complete sky coverage with lower field of view sizes. Lower Field of view sizes are desirable due to the fact that they are the cheapest way to lower the number of degrees per pixel, which improves the accuracy of the final position estimate.

This issue was addressed by Dr. Mortari's student Benjamin Spratling in his PHD thesis, describing a new startracker algorithm called Star-ND[2]. In it, he proposes a new technique which involves selecting the three stars that are nearest to the pilot star, and simply using those as the basis for the star triangle. This allows each pilot star to have a single, unique constellation, which drastically reduces database size (cases where the nearest three stars might be ambiguous are handled by checking both cases). Finally, rather than performing three separate lookups and cross checking the results, it is possible to drastically increase performance by using a hash function which takes the angles between the star furthest from the pilot star and the two middle stars, and produces a number which is unique to each constellation. The end result is that it is possible to increase both accuracy and performance. This was tested in flight with an 8x8 fov camera, and a star tracker database consisting of 7.5 magnitude stars. However, it is assumed that centroiding accuracy is on the order of within 1/20 of a pixel of the database position, which requires that the camera be intentionally defocused in order to spread the light around among multiple pixels, as well as a very accurate database.

In order to meet our goals, we need another order of magnitude Field of View improvement above Star-ND, while simultaneously reducing the accuracy requirement. This is accomplished by the following refinements to the algorithm:

1. Rather than using the method for generating the database lookup key described in Star-ND, we order the stars by brightness, and use the brightest star as the pilot. We then simply measure the distance between pilot and the 2nd, 3rd and fourth brightest stars respectively, and use those values to form a key for the k-vector lookup. Cases where the relative brightness is ambiguous is handled similarly to the distance ambiguity, by inserting it into the database both ways.
2. Memory usage is dramatically reduced by memory mapping the constellation database. This is a programming technique which intelligently loads bits and pieces of a file into memory as needed rather than reading the entire thing into memory.

For a case where average constellation width is 500 pixels, this results in a 1000x improvement in the maximum number of constellations which can quickly be looked up (resolving the ambiguity of the ordering of the values results in a 2x improvement, and adding a third value results in a 500x improvement).

This can also be traded for a 10x improvement in the amount of tolerable star position error.

Automatic star tracker calibration and database generation

The process for automatically calibrating the star tracker is divided into three parts, as follows:

1. Star tracker calibration.
    a. Generate background subtraction mask & calculate image statistics
    b. Solve test images using astrometry.net & compute average

position error in star database using solved sample images
    c. Determine variance of star magnitude between stars as seen in the image vs the database
2. Star database preprocessing
3. Constellation database generation

Star Tracker calibration:

Generate background subtraction mask

The first step in preprocessing star images is to subtract out the average noise background, as well as any dead pixels that may be present. In order to generate a background image, we remove stars by taking the median of multiple test images containing stars in different positions of the sky. This produces a clean test image containing just the background, with no stars present. We are then able to use this background subtraction mask to reduce background noise in future images.

Since this process is robust to stars in the image, it can also be used for inflight recalibration (ie. to fix dead pixels)

During this process, we also calculate image noise variance, which will be used later in the database generation process

Solve test images using astrometry.net

A piece of software called astrometry.net[3] is used in this stage of the process to solve the orientation and field of view size of our images. Using the orientation information, we line up the stars in the image with the stars in our database, and compute the average position error between them. The field of view size of the calibration image which matches the best is used later in the database generation process

Star database preprocessing:

The star database is derived from a standard catalog such as that hipparcos star survey. We load this catalog & flag stars that are unsuitable for inclusion in the database due to measurement error, being too close to a neighbor star, high levels of brightness variability, or being too dim for us to see. We then update their positions to the current year based on the motions observed by the hipparcos mission, and store the position, brightness, and brightness variability of the stars, as well as the suitability flag.

Constellation database generation:

Stars which have been flagged as suitable for inclusion in the star database are used to generate constellations in two ways: First, for each suitable star select the three nearest suitable stars. Second, we need to add entries to handle the case where we have enough stars to match, but do not have any complete constellations of the first type in our field of view. To do this, we generate a list of overlapping fields of view that cover the sky, select the star closest to the center of each, and generate a constellation consisting only of suitable stars currently in the field of view. In both cases, we handle ambiguities in determining the three closest stars by generating multiple constellations. Next, we sort all of the stars in each constellation by brightness (duplicating when ambiguous) and filter out non-unique constellations. Finally, we save the star id's and distances between them to a binary k-vector file. A technique called memory mapping allows us to quickly access the data without having to pay to load it into memory. This process is sped up by loading the star positions into a kd-tree which is a data structure for rapidly searching objects in n-dimensional space

Image preprocessor:

The image preprocessor is the component which extracts the positions of stars from an image. In our case, we also need to compensate for motion blur due to spacecraft rotation. Thus, image preprocessing consists of two steps:

1. Background subtraction
2. Star position extraction & projection

Background subtraction:

We subtract the noise background which was computed previously. This lowers the noise floor, removes dead pixels and certain other artifacts, to produce a cleaner image

Star position extraction & projection:

The centroiding technique is used to extract the positions of stars. This technique works as follows:

1. Set a threshold brightness. Any pixel which falls above this threshold is considered to be part of a star, while any pixel which falls below this threshold is considered to be sky
2. Find the brightness weighted centroid of each star

3. Refine the position of the centroid by including pixels along the boundary of the star and recalculating the brightness

Finally we project the extracted star centroids onto the celestial sphere. From there we are able to feed these star positions into the star-tracker.

Automatic test suite:

The final piece of the framework is the automatic test suite. During the first stage, we validate against an additional set of test images taken with the startracker camera under realistic conditions to insure that we can solve them quickly. Finally a GPU based, hardware in the loop camera image simulator generates realistic moving starfields which are displayed on a projector, which is used as an input for the startracker camera. This is used to determine expected slew tolerance of the star-tracker under a variety of conditions

Results:

The complete process of calibration, database generation, and validation is run automatically via a single shell script. Validation completed successfully with both the consumer grade Logitech HD C270, as well as our higher quality Gigevision Smartek flight camera. We present a step by step breakdown of our results for the Logitech HD C270 below, as it is the more difficult of the two tests:

./startracker_unit_test.sh

At this point, we generate the background subtraction mask, calculate image noise statistics, and solve the test images using astrometry. We then match up the stars in the test images with the stars in the database, and calculate the average error between them (in pixels)

DEC=60.4992808694
RA=179.143292443
ORIENTATION=179.092210058
Stars found: 13
Average err: 0.816428478937

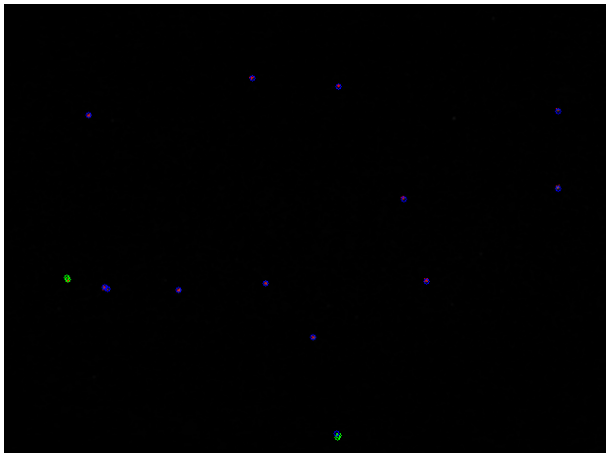

Figure 1: Centroids (red) vs Database (blue)

...
DEC=76.024542615
RA=-179.46769828
ORIENTATION=179.392884815
Stars found: 11

Next, we calculate lower, upper and median bounds which relate the brightness of stars in our database to stars in the image. Due to the low quality of the camera, there is a large variance. The calibration process compensates for this automatically by generating a database in which very low emphasis is placed on star brightness.

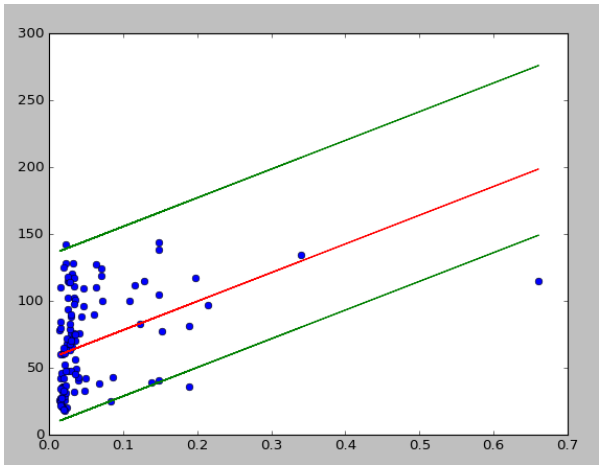

Figure 2: Star brightness vs. Database

Average err: 0.778982947253

Next we generate the star database. Statistics for sky coverage, number of constellations, and maximum constellations (for this particular camera) are generated

Database coverage: 98.2843137255% percent of the sky

PARAM1=75
PARAM2=75
PARAM3=75
NUMCONST=14280
MAXCONST=52734
...

Finally, we validate our image against actual test data. The startracker is able to solve the lost in space problem for these images in roughly 30 milliseconds (that's fast!)

Using OpenCV centroiding:

bg_sample/w1.png
Time: 0.0321140289307
DEC=60.4956492831
RA=179.157306008
ORIENTATION=178.979274135

….
bg_sample/w8.png
Time: 0.0311989784241
DEC=76.0048732316
RA=-179.516300048
ORIENTATION=179.343238004
andrew@spaaace:~/startracker$

Final remarks:
Many of the pieces necessary for a high performance automatic star-tracker optimization framework have been developed independently by different research groups, but they have never been unified into a single, easy to use framework for automatic star-tracker optimization. It is our hope that this work will put high performance star trackers within reach of college or even high school cubesat projects, making missions requiring high precision attitude determination accessible for all.

References

1. Search-Less Algorithm for Star Pattern Recognition. 1997, D. Mortari et. al,
2. Star-ND (Multi-Dimensional Star-Identification). 2012, Spratling, Benjamin et. al,
3. ASTROMETRY.NET: Blind Astrometric Calibration of Arbitrary Astronomical Images, 2010, Dustin Lang,