SSC17-VIII-3

TETRA: Star Identification with Hash Tables

Julian Brown, Keaton Stubis
Massachusetts Institute of Technology, Space Systems Laboratory
Bldg. 37-350, (831) 747-5366
Brownj4@mit.edu

Faculty Advisor: Kerri Cahoy
Massachusetts Institute of Technology, Space Systems Laboratory

ABSTRACT

We present Tetra, a star identification algorithm that uses the minimum possible computation time and number of database accesses to solve the calibrationless lost-in-space problem. To solve the calibrationless lost-in-space problem, a star tracker must determine its attitude with no a priori knowledge, not even lens parameters such as the field-of-view or distortions. Tetra is based on a directly-addressed hash table data structure, which enables it to identify star patterns with a single database access. We prove a tight bound on Tetra's false positive rate and empirically compare Tetra's runtime, centroiding error sensitivity, and field-of-view error sensitivity with earlier lost-in-space algorithms: Pyramid and Nondimensional Star ID. We also compare Tetra with hash table based modifications of Pyramid's cross-referencing step and Nondimensional Star ID's database lookup, which improve Pyramid and Nondimensional Star ID's runtimes by an order of magnitude without otherwise impacting their performance.

We find that Tetra outperforms the earlier algorithms and their improved counterparts in every measured metric: runtime, centroiding error sensitivity, and field-of-view error sensitivity. Tetra, our free software alternative (https://github.com/brownj4/Tetra), will enable the next generation of spacecraft navigation technology.

INTRODUCTION

Star trackers are the backbone of every high accuracy Attitude Determination and Control Subsystem (ADCS). Star trackers leverage an advanced form of celestial navigation to provide satellites with pointing information accurate to arcseconds. But before a star tracker can orient itself relative to the stars, it must first identify the stars it can see. This is known as the lost-in-space problem. Modern compact star trackers typically take four seconds[1] or more[2] to solve the lost-in-space problem.

Daniel Hegel of Blue Canyon Technologies describes star identification as "one of the most important (and most difficult) steps in star tracker operation."[1] The importance and difficulty of solving the lost-in-space problem has prompted the development of a wide variety of star identification algorithms, from the popular Pyramid algorithm[3] to neural network-based approaches.[4] The algorithms to date vary wildly in their tradeoffs between what we'll call The Three R's: Runtime, Robustness, and (hardware) Requirements. Tradeoffs between The Three R's directly correspond to the old rule of "Fast, Good, Cheap. Pick two." This paper explores methods for bending and breaking this rule to create star identification algorithms that are simultaneously fast, robust, and runnable on cheap hardware.

Paper Organization

This paper begins by formalizing the lost-in-space problem and analyzing existing algorithms: Pyramid[3], Nondimensional Star ID[5], and Astrometry.net.[6] The rest of the Background section describes hash tables in preparation for the Methods section, which describes our hash table optimizations to Pyramid[3] and Nondimensional Star ID[5], and presents Tetra, our free and open source star identification algorithm. The Analysis section provides a deep mathematical analysis of Tetra's robustness. The Results section compares runtimes and robustness of Tetra with C implementations of Pyramid[3] and Nondimensional Star ID[5] and their hash table-optimized revisions over millions of simulated images given both star position (i.e. centroiding) and field-of-view (FOV) estimate error. The Results section also presents results from entering Tetra into "Star Trackers: First Contact," a star identification algorithm competition created by the European Space Agency. Finally, the Results section compares runtimes and robustness of Tetra with our

optimized revisions of Pyramid[3] and Nondimensional Star ID[5] over 472 night sky images taken with an uncalibrated Aptina MT9P011 CMOS imager. This paper's Conclusion summarizes our experimental results and details future work.

BACKGROUND

The lost-in-space problem requires identifying stars in an image of the night sky with known FOV. When the FOV estimate is inaccurate, the lost-in-space problem becomes the calibrationless lost-in-space problem. To simplify formalization of the problem, we will focus solely on the stars in the image, ignoring much of the information which may be available in an arbitrary night sky image, such as the moon, planets, and asteroids. Additionally, we will only consider the positions of the stars in the image, ignoring values like star brightness and color, which are typically not accurately measurable. Given these assumptions and n stars in the image, the useful information in the image can be represented as $2n$ independent real numbers corresponding to the x and y pixel coordinates of the stars in the image. The unknown camera orientation removes three degrees of freedom: x translation, y translation, and rotation, giving $2n-3$ independent degrees of freedom.[7] In the calibrationless case, the FOV is also unknown, so image scale must also be removed, resulting in $2n-4$ degrees of freedom. The performance of a lost-in-space algorithm comes down to how it extracts and uses the image's $2n-3$ or $2n-4$ degrees of freedom.

Pyramid[3]

In 2004, Mortari published the Pyramid algorithm,[3] which identifies patterns of four stars by measuring the distances between pairs of stars, finding lists of potential matches based on the distances, then cross-referencing the lists to find the unique solutions for all 4 stars. More explicitly, Pyramid solves the lost-in-space star identification problem using a catalog of every pair of stars the star camera might see. Pyramid sorts its database by the distance between the star pairs. To identify a given pair of camera stars, Mortari uses his $O(1)$ k-vector search technique to retrieve the contiguous chunk of memory containing all possible matches within a given tolerance. As there may be thousands of possible star pair matches, and the ordering of the star pair is unknown due to symmetry, multiple star pairs must be used to make a unique identification. To this end, Pyramid constructs a pyramid of four camera stars and cross-references the possible match lists of the pyramid's six star pairs. The majority of Pyramid's running time is taken up by the $O(k^2)$ cross-referencing step, where k is the number of possible matches in each match list.[7]

The distance between a pair of stars is precisely the degree of freedom left over after removing translation and rotation. However, information in the pairs is redundant. There are 6 pairs in a four star pattern, but only $2n-3 = 5$ degrees of freedom. By the pigeonhole principle, some of the information must be redundant.

More intuitively, the redundancy is most apparent when the stars form a line, as in Figure 1. The light gray circular regions around the outer stars represent the possible locations of the third star given error in their distance measurements. The dark grey region at their intersection is the union of the information. The region is much larger than the actual positional error in the image, represented by the small white circle around the middle star. The extra space in the dark gray region indicates Pyramid's extraction method can cause it to mismatch star patterns.

An additional problem results from Pyramid discarding reflective matches called specular triangles.[3] Small centroiding errors can cause star triangles that lie along a line to become specular. Pyramid may then throw out the correct match and incorrectly match to a different triangle. While the problem of degenerate triangles is mentioned in the original paper, it is not solved.[3]

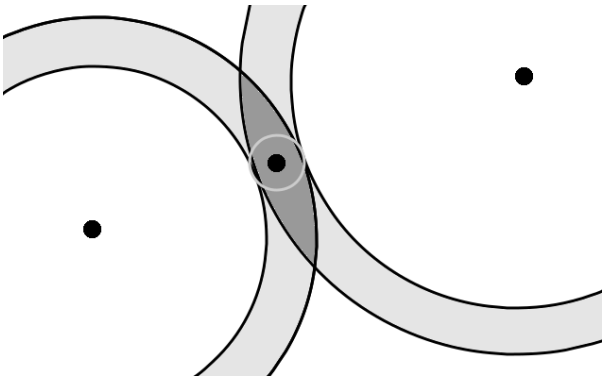

Figure 1: Pyramid Error vs. True Error

Non-dimensional Star ID[5]

In 2006, Samaan, published the calibrationless lost-in-space Non-dimensional Star ID (ND Star ID) algorithm.[5] ND Star ID identifies patterns of four stars by measuring the internal angles of the triangles formed by star triplets, finding lists of potential matches for the triangles, then cross-referencing the lists to find the unique solution for all four stars. More explicitly, ND Star ID solves the calibrationless star identification problem using a catalog of every triplet of stars the star camera might see. ND Star ID sorts its catalog by the smallest interior angle of the triangle formed by each star triplet. To identify a given triplet of camera stars, Samaan uses Mortari's O(1) k-vector search technique

to retrieve all possible catalog triplets with a smallest interior angle within a given tolerance. Samaan then iterates over the retrieved triplets, of which there may be thousands, selecting only those triplets that also match the camera triplet's largest interior angle to within a given tolerance. The few remaining possible triplet matches are confirmed against a fourth camera star by checking the three triplets formed by star pairs from the old triplet with the new fourth star against the catalog with the same method as before. The majority of ND Star ID's runtime is taken up by the *O(k)* iteration over the retrieved triplets, where *k* is the number of retrieved triplets.

There are four triangles in a four star pattern, each of which has two independent internal angles, resulting in 8 extracted parameters. As calibrationless algorithms only have *2n-4 = 4* degrees of freedom for four stars, ND Star ID's extraction method is highly redundant. Additionally, when stars are close together, small centroiding errors can cause large errors in the measured angles, as depicted in Figure 2. This arrangement of stars can cause ND Star ID to misidentify star patterns.

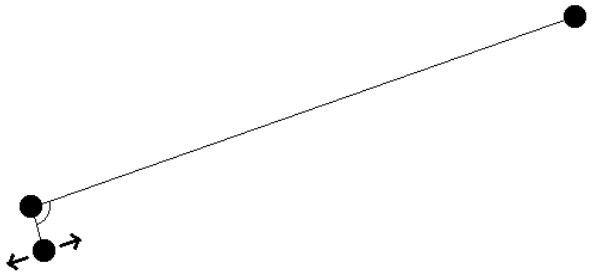

Figure 2: ND Star ID Internal Angle Error

***Astrometry.net*[6]**

In 2010, Lang published Astrometry.net, a tool primarily designed to calibrate backlogs of telescope data.[3] Astrometry.net is a calibrationless lost-in-space algorithm, which identifies patterns of four stars using the coordinate system depicted in Figure 3. The pair of stars furthest apart are placed at (0, 0) and (1, 1) and the extracted parameters are the transformed x and y coordinates of the remaining two stars. Astrometry.net's extraction method produces 4 independent values, which directly correspond to the *2n-4 = 4* degrees of freedom in the image stars.

Based on the extraction analysis alone, Astrometry.net appears to be the perfect calibrationless lost-in-space algorithm. While it's certainly robust, its pitfall lies in the other two R's: runtime and (hardware) requirements. It often takes tens of seconds to run on desktop hardware and requires large amounts of RAM to store its database. The database is a kd-tree, which is a multidimensional binary tree. Finding matches for a given pattern takes *O(log(n))* time, where *n* is the number of patterns stored in the database. Although logarithmic runtimes are generally considered fast, after accounting for constant factors and millions of patterns stored, a single lookup can require many dozens of database accesses.

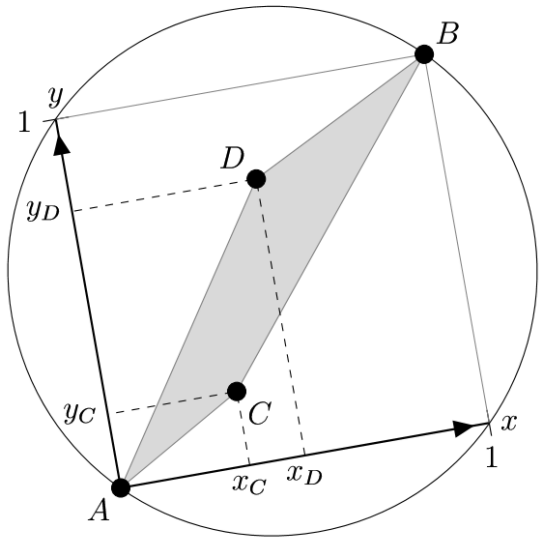

Figure 3: Astrometry.net Parameter Extraction[6]

***Hash Tables*[9]**

Hash tables are a generalization of arrays, implementing *O(1)* data lookup given arbitrary data instead of an index. They operate by producing a deterministically random array position from the input data, then resolving any collisions when multiple inputs map to the same array position. Directly storing data in the array positions is known as open addressing.[9] Open addressing can be faster than storing a pointer to a list, but requires searching for another empty location when collisions occur, in a process known as probing. Quadratic probing attempts to place collisions in array position determined by a quadratic function. For example, if the original array position is occupied when inserting a new item, the following array position is checked, then an array position three away from the original array position, then six, and so on until an empty array position is found.

Quadratic probing has the property of locality, meaning it places collisions nearby their original array position. Accessing any amount of data from disk typically requires reading in and caching an entire 4 KB block. Probing locality ensures that even if the stored data isn't in its original array position, it will very likely

occupy the same block and have already been cached in the original read. Therefore, a hash table stored on disk that uses open addressing combined with quadratic probing often requires exactly one disk access to perform data lookup.

Operations that require disk accesses are typically avoided when writing fast code, as disk operations are orders of magnitude slower than memory accesses. But when only one or a few data lookups are needed and the data structure is a hash table with locality, disk accesses become a viable option. Disk operations become especially worthwhile in hardware and power constrained environments like space.

APPROACH

The fundamental idea in this paper is that hash tables can be used to make fast star identification algorithms. In our modification of Pyramid, hash tables are used to efficiently perform a necessary cross-referencing step. In our modification of ND Star ID, a hash table replaces the database to speed up data lookup. Tetra combines both principles by using a hash table to store the database in a way that eliminates the need for cross-referencing.

Pyramid Improved

The algorithmic problem behind Pyramid's cross-referencing step is to find all valid star triangles that can be formed given three lists of possible star pairs, corresponding to the triangle's sides. The original Pyramid algorithm performs cross-referencing in time proportional to the square of the list sizes. Using hash tables, cross-referencing can be performed in linear time.

The hash table maps potential star identification numbers (ids) to two lists of their paired star ids, one for matches from side A and one for matches from side B. Inserting the star ids from side A and side B into the hash table takes *O(1)* time per insertion over *O(k)* insertions for a runtime of *O(k)*, where *k* is the length of the star pair lists. Star pairs from side C are then looked up in the hash table, retrieving the list of side A's paired stars for one of side C's stars and the list of side B's paired stars for the other of side C's stars. The retrieved lists are cross-referenced to find matching star ids, which correspond to the star shared by sides A and B. The side C star pair used to retrieve the two lists are the remaining two stars. The retrieved list sizes are *O(1)* for reasonable parameter choices, so each of the *O(k)* retrievals and cross-references take *O(1)* time for a runtime of *O(k)*. Adding together the *O(k)* runtime of inserting the stars into the hash table and the *O(k)* runtime of retrieving and processing them gives a total runtime of *O(k)*.

ND Star ID Improved

The runtime of Samaan's ND Star ID is dominated by an iteration over retrieved catalog triplets. This iteration is meant to narrow down the retrieved catalog triplets from those that match the camera triplet's smallest interior angle to within a given tolerance to those that match both the smallest and largest interior angles. By using a hash table as the database instead of a k-vector[3], we can retrieve only the catalog triplets that match both angles, thereby eliminating the iteration entirely.

Hash tables cannot be directly indexed using values with error like the pair of real numbers needed to represent a star triangle's internal angles. The components with error must first be discretized in a binning process. An example of binning a point with circular error is shown in Figure 4. The union of bins overlapped by the error range is guaranteed to contain all values within range of the point. The bins will also contain a constant factor times as many values nearly within range, which can be filtered out. A good balance between having to check a large number of bins and having to filter out a large number of values places the bin sizes are on a scale proportional to the input value's error. In the case of ND Star ID, the error value is a constant 0.0005 radians, so we can use constant sized bins.[5]

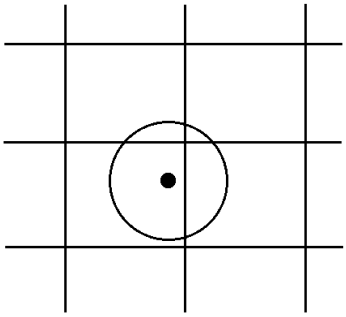

Figure 4: Binning in Two Dimensions

Tetra

Tetra solves the calibrationless lost-in-space algorithm in constant time and with a constant number of database accesses, in most cases exactly one. An asymptotically faster algorithm than Tetra cannot exist. Tetra achieves its seemingly impossible runtime by performing any difficult work during the creation of its database.

Tetra typically operates on patterns of three or more stars, but can operate on two star patterns given information that disambiguates the star pair, such as star brightness. The recommended number of stars is three or four. For simplicity of comparison with the other algorithms, simulation and real world testing were

done with four star patterns. Tetra starts by constructing a coordinate system similar to the one used by Astrometry.net in that they both orient and normalize using the largest edge. The primary differences are that Tetra places the largest edge along the x axis and does not restrict the remaining pattern stars to be within a circle. The coordinates are instead restricted by two quarter-circles formed by the largest edge constraint, as depicted in Figure 5. Coordinates cannot exist outside this region, as they would form an edge larger than the largest edge. Tetra also measures and uses the length of the largest edge to maintain precision when the FOV error is bounded or the FOV is accurately known.

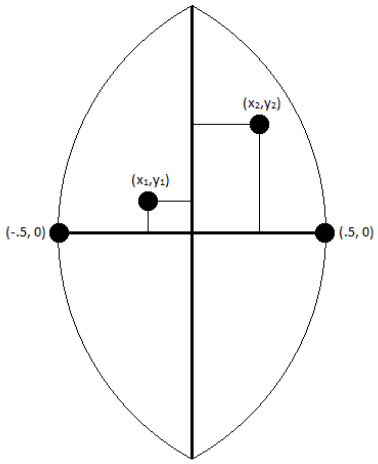

Figure 5: Tetra Parameter Extraction

Tetra's coordinate system was chosen to simplify computation and so that error in the coordinates and the largest edge is linear with respect to their magnitudes. The error equations are provided in the Analysis section. The coordinates and largest edge length are binned such that their error ranges overlap at most two bins in each dimension. Restricting the bin sizes to exactly meet this criteria ensures a good tradeoff between the database size and runtime. The error in Tetra's coordinates depends on the largest edge length, so the largest edge is binned first. The error in the x coordinates depend on the y coordinates, so the y coordinates are binned second and the x coordinates last.

Tetra inverts the standard binning procedure by inserting catalog patterns into all bins within their error range, then only looking up the single bin the image pattern occupies. Tetra thereby makes a direct tradeoff between database size and runtime, allowing it to match star patterns with a single database access. Reconstructing the bin a catalog pattern was placed into is accomplished using a single bit per dimension, specifying which of the two possible bins the pattern was placed into.

The stars in the pattern which aren't part of the largest edge are sorted by their bins, first by x bin, then by y bin. The 180 degree rotational ambiguity in Tetra's coordinate system is resolved by choosing the rotation which makes the largest magnitude x bin non-negative. The same star pattern will always produce the same coordinates with the exception of ambiguities resulting from measurement error. Although error tolerances are low in practice, meaning ambiguities are typically rare, resolving the ambiguities is necessary to ensure Tetra is always able to identify and therefore never misidentifies valid star patterns. The ambiguities are resolved by duplicating the catalog pattern in the database across the ambiguities, ensuring any of the possible image patterns resulting from measurement error will give the correct result. The ambiguities Tetra resolves are in: the largest edge, the 180 degree rotational ambiguity, and the star ordering given identical x and y bins.

ANALYSIS

For the analysis, we will be looking into three main things. First, we will give error bounds on the various parameters we work with. Despite their relative simplicity, proofs of these bounds take a significant amount of space, and so will be omitted. Second, we will describe how to use these error bounds to bin their corresponding parameters. Finally, we use the bounds to estimate Tetra's failure rate in identifying images.

Setup

We choose the following conventions. Let F_{est} be our estimate for the FOV of the imager, F_{act} be its actual FOV, and e_f be the field of view error, given by $1+e_f = \frac{F_{est}}{F_{act}}$. We will assume that the field of view error is bounded in absolute value by some number $e_{f\max}$. Let $e_{c\max}$ be the centroiding error as a fraction of the actual field of view. Let L denote the largest edge length of a pattern, also as a fraction of the field of view. We will be assuming that L is large compared to $e_{c\max}$. Finally, when examining a vertex in a pattern that isn't an endpoint of the largest edge, we will denote its unnormalized internal coordinates by x and y, and we will let $x' = x/L$ and $y' = y/L$ be its reduced coordinates, as given by Tetra's parameter extraction.

Bounds

Bound 1: If ΔL_c denotes the effects of centroiding error on the length of the largest edge, then

$$\Delta L_c \le 2\frac{F_{est}}{1-e_{f\max}}e_{c\max} \quad (1)$$

Bound 2: If Δy_c denotes the effects of centroiding error on an internal y coordinate, then to first order in $e_{c\max}$

$$\Delta y_c \le 2\frac{F_{est}}{1-e_{f\max}}e_{c\max} \quad (2)$$

Bound 3: If Δx_c denotes the effects of centroiding error on an internal x coordinate, then to first order in $e_{c\max}$

$$\Delta x_c \le \left(1+\sqrt{1+4\left(\frac{y}{L}\right)^2}\right)\frac{F_{est}}{1-e_{f\max}}e_{c\max} \quad (3)$$

Bound 4: If $\Delta y'$ denotes the effects of both error types on an internal y' coordinate, and we let $g=\frac{F_{est}}{\left(1-e_{f\max}\right)^2}e_{c\max}$ then to first order in $e_{c\max}$

$$\Delta y' \le \frac{2g}{L}|y'| + \frac{2g}{L} \quad (4)$$

Bound 5: If $\Delta x'$ denotes the effects of both error types on an internal x' coordinate, then to first order in $e_{c\max}$

$$\Delta x' \le \frac{2g}{L}|x'| + \frac{\left(1+\sqrt{1+4(y')^2}\right)g}{L} \quad (5)$$

Binning

When we bin a given coordinate, we are looking to satisfy a single rule we will call the binning constraint. The requirement is that the error range around a particular coordinate can only meet at most two bins in its dimension. As mentioned earlier, ideally we want the bins to be as small as possible, but still obey this constraint.

First note from bounds 4 and 5 that error in an internal coordinate is an increasing function that scales linearly as that coordinate moves away from 0. Since the error is an increasing function, as a coordinate grows in absolute value, so must the sizes of the bins. It is a surprising result that even though the error grows linearly, the bin sizes must grow exponentially. We will spend the rest of the subsection showing this. Since we have two different linearly growing functions to deal with, we will just consider a general function $f(x)=m|x|+b$. For our argument to work we will need $0<m<1$, $b>0$, but our functions do satisfy these relations since g is small compared to L.

To specify where our bins are, we choose to give the locations where one bin stops and the next starts. We will call these locations the bin divisions. A nice visual way to see where they must go is to start by graphing the error function $f(x)=m|x|+b$, as in Figure 6. Around each point on the graph, we center a horizontal strip whose width is twice the height of the function at that point. This strip will exactly represent the possible places our coordinate could end up due to error. We can now see that the binning constraint exactly says that if we put a vertical line at each bin division, at most two such lines will strike each horizontal strip. The figure below contains an example in which there are two bin divisions that are too close together, because they both intersect the red strip. The picture has been stretched vertically for visibility, and the region covered by all the horizontal error strips has been shaded.

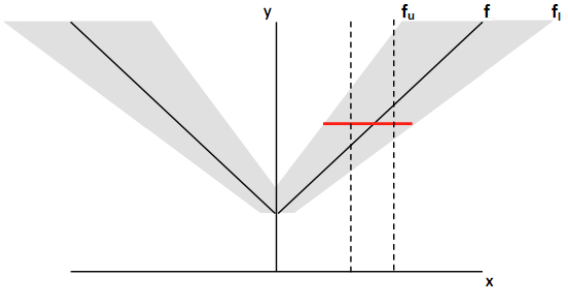

Figure 6: Tetra Error Regions

In our particular picture, the shaded region can be thought of as being bounded between two absolute value functions, one above it, and one below it. In fact, this will always be the case as long as $0<m<1$, $0<b$. Let us refer to the upper function as f_u, and the lower one as f_l.

Now, consider the following procedure for constructing the bin divisions. Start at (0,0) in the picture, and begin to move upwards, until you hit f_u. Stop, and move right until you touch f_l. Stop again, and begin moving upwards until you touch f_u again, and continue to alternate between these two movements indefinitely. This will create a staircase shaped path inside the gray region. In Figure 7, we can see the beginning of the path, which has also been mirrored on the other side of the y axis. It should be possible just by inspection to see that the places where the staircase turns form a choice of bin divisions that satisfy the binning constraint. In addition, notice that x_1 and x_2 are the endpoints of one of the horizontal error intervals. Thus, if we were to move x_2 to the left even slightly, we

would violate the binning constraint. Thus, the divisions we get this way are optimal.

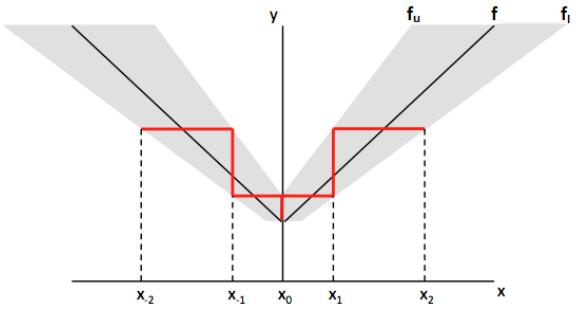

Figure 7: Tetra Bin Divisions

We see in Figure 7 that when one has a bin division located at some positive value x_i, the next possible place to put a division is at $x_{i+1} = f_l\left(f_u^{-1}(x_i)\right)$. A direct computation we choose to omit shows that if we let $g(x) = x - f(x)$, then we may rewrite our formula as below.

$$x_{i+1} = 2g^{-1}(x_i) - x_i \qquad (6)$$

This formula ends up holding for any error function f with strictly positive outputs that is increasing in $|x|$ and obeys $|f'(x)| < 1$. However, the formula does not hold when $x_i < 0$. Such cases can be dealt with by considering $f(-x)$ instead.

To wrap up our problem, we will choose our initial division to be $x_0 = 0$. We will then call subsequent divisions x_1, x_2, … and can recursively find their locations using equation (6). Note that we are not currently worried about divisions of negative index, since we will just have $x_{-i} = -x_i$. It is not too hard to verify that in our case, the solution is given by

$$x_i = \frac{b}{m}\left(\left(\frac{1+m}{1-m}\right)^i - 1\right) \qquad (7)$$

Now, we can just plug in the particular values of m and b that we computed in our bounds earlier to get the bin locations. This function is also easily solved for i in terms of x_i, so we have a way to figure out which number bin a given x value lies in. Also, as we claimed earlier, this is exponential. The easiest way to see why the answer takes on such a form is to imagine dropping the left hand side of Figure 7 and extending the graphs of f, f_u, and f_l as straight lines until they meet. All three lines will meet at a point on the x axis because the width of each horizontal strip equals twice its height. Thus, we can see that our staircase is bouncing in a region bounded by two straight lines. However, this shape is invariant under outward scaling. Thus, the resulting staircase inside will be self-similar, and must grow as some exponential.

Failure Rate of Patterns

Say that you photograph some real n star pattern q' in the sky, and end up with an image pattern that we call q. Tetra is designed such that despite error in the imaging process, q will always show up as one of the possible matches for q'. However, it is still possible for the algorithm to fail to identify q if more than one match happens to show up for it. Without additional stars in the image, it would be impossible to tell which one of the possible matches is truly the correct one. The chance of failure in such a case is an important number to the algorithm, and we call it the redundancy. The reason for this name is because it measures the chance that somewhere else in the sky is another pattern that just happens to match q by pure chance. If we let $p_{match}(q)$ be this chance for the particular image pattern q, and if we let $p_{image}(q)$ be the probability distribution for getting any particular image pattern, then we can compute the average redundancy as $\int_{Q_n} p_{image}(q)\,p_{match}(q)\,dq$, where we are letting Q_n denote the space of all possible n star image patterns. Another important quantity that we are interested in is the chance of random dust or debris being incorrectly recognized as real stars (the false positive rate). Say we have some fake pattern q. Then, we can define $p_{match,fake}(q)$ and $p_{image,fake}(q)$ as the chance of a match coming up for the fake pattern q and the chance of the debris forming any particular fake pattern q in the first place. So, the chance of the fake pattern getting a match is just $\int_{Q_n} p_{image,fake}(q)\,p_{match,fake}(q)\,dq$. However, it is a reasonable assumption that both debris and real stars should be uniformly randomly distributed in the night sky. Hence, we actually have that $p_{match,fake}(q) = p_{match}(q)$, and $p_{image,fake}(q) = p_{image}(q)$. Thus, this failure chance is also just the redundancy.

Below, we will attempt to derive an upper bound on the redundancy. To begin our estimations, consider the celestial sphere of unit radius, and consider n points

that are randomly distributed on the surface of the sphere. The distribution can be written as $\frac{1}{(4\pi)^n} dP_1 dP_2 ... dP_n$, where dP_i is an infinitesimal amount of area around the ith point. We are interested in how image patterns are distributed, and this lets us make a useful assumption. To be in an image, a pattern has to fit entirely within the FOV of an imager. However, the FOV of an imager is usually small enough that the part of the sphere it sees is approximately flat by the small angle approximation. Thus, we can reasonably assume that each image pattern is planar. Now we want to change coordinates into the coordinate system we are using to encode patterns. The first step is to convert to the coordinates $(P, L, q, x_1, y_1, ... x_{n-1}, y_{n-2})$ where P is the position of the first point of the largest edge, L and q are the length and direction of the largest edge, and the rest of the coordinates are the internal coordinate system that is created. The Jacobian will be L, since when we perturb L and q by amounts dL and dq, we get a wedge of area $LdLdq$, and when we perturb x_1 by an amount dx_1 and y_1 by an amount dy_1, we create a rectangle of area precisely $1 dx_1 dy_1$, (same holds for all dx_i, dy_i). Finally, the map from points on a sphere to this new coordinate system is an $n!$ to 1 map, since we can freely relabel which point is point 1, point 2, etc... without changing the shape of the pattern. So, the resulting distribution is $\frac{n! L}{(4\pi)^n} dP dL dq dx_1 dy_1 dx_2 dy_2 ... dx_{n-2} dy_{n-2}$. However, two patterns are actually the same if we can rotate and translate one onto the other. So, we have to integrate out dP and dq. If we do this and denote all of the x and y components using $dXdY$ for shorter notation, we get $\frac{2\pi n! L}{(4\pi)^{n-1}} dL dX dY$. We use reduced x and y coordinates, so we make one last change of coordinates to them. This then gives the formula for the chance of a random pattern on the sphere being some particular pattern q. However, along the way we assumed that q was an image pattern, so this is an approximation that only holds when q is small enough to fit in an image. Denoting this chance $p_{pattern}$,

$$p_{pattern}\left(\text{image} \quad \text{pattern} \right) = \frac{2\pi n! L^{2n-3}}{(4\pi)^{n-1}} dL dX' dY' \quad (8)$$

In order to obtain P_{image} from this, we need to model how a random image pattern is chosen. One reasonable way is to fix a distance L_m, and restrict attention to patterns such that each star lies within a distance L_m of all of the others. Out of all such patterns we then choose one uniformly at random. To be a good model, we now just have to choose L_m to reasonably capture the size of our FOV. An advantage to using this model is that it makes the space Q_n of image patterns particularly simple.

Notice that in a pattern, we can think of the coordinate L as giving the overall scale, whereas the coordinates x'_i, y'_i give its shape. There are a few restrictions on the values that x'_i and y'_i can take on. First of all, the edge between the points located at $(-L/2, 0)$ and $(L/2, 0)$ must be the largest edge in the pattern. There are also some choices we make in assigning our internal coordinates, specifically that the x'_i are sorted, and the largest x'_i in absolute value will be positive. We denote the $2n-4$ dimensional space of choices that respect these conditions Q'_n. Then, our model for choosing images has two implications. First, $Q'_n = [0, L_m] \times Q'_n$, where we are taking a Cartesian product with a closed interval. Second, P_{image} is obtained from $P_{pattern}$ by restricting the domain to Q_n and renormalizing.

From this last observation, we get $P_{image} = CL^{2n-3} dL dX' dY'$, where C is an unknown normalization constant that has absorbed all of the leading constants. The total probability must sum to 1, so when we integrate this over Q_n, we end up with $1 = \frac{C}{2n-2} L_m^{2n-2} \int_{Q'_n} dX' dY'$. If we let V'_n denote the volume of Q'_n given in the integral, we can solve for C and plug back into our original formula to get

$$p_{image} = \frac{2n-2}{V'_n} \frac{L^{2n-3}}{L_m^{2n-2}} dL dX' dY' \quad (9)$$

Going back to equation (8), we can also derive a bound for P_{match}. The first step is to bound a distribution we will call $P_{patmatch}(q)$, which measures the chance that when we randomly pick n stars on the sphere that it forms a pattern that matches q. A match occurs when a pattern falls within all of our error bounds of q. Since we are just upper bounding, in our bounds we can

assume the worst possible cases of $|x'| = L/2$ and $|y'| = L\sqrt{3}/2$. (The reason why these are the worst cases is because if either coordinate were larger, L would no longer be the largest edge.) From bounds 4 and 5, $\Delta y'_i \le \left(2+\sqrt{3}\right)\gamma/L$ and $\Delta x'_i \le 4\gamma/L$, which should be on average fairly generous bounds. As for L, it undergoes both field of view error and centroiding error, so $\Delta L = Le_f + \Delta L_c$. Normally, we would omit the lower order ΔL_c term, but in the case where the field of view of the imager is known, $e_{f\max} = 0$ and centroiding becomes the leading contribution to the error.

$p_{patmatch}(q)$ is the volume of the region of matches to q, so we can now use the distribution $p_{pattern}$ to arrive at the answer. First note that each of the x_i and y_i bounds are order $e_{c\max}$, and hence quite small. However, the bound of L can be big if $e_{f\max} \neq 0$. Thus, our region should look like a very thin tube. It's total volume is given by

$$\int_{L-\Delta L}^{L+\Delta L} \frac{2\rho n! L^{2n-3}}{(4\rho)^{n-1}} (2\Delta x')^{n-2} (2\Delta y')^{n-2} dL \qquad (10)$$

Note that we used twice our bounds, since a match can occur when x' is either smaller or larger than the given value. When we plug in for our Δ bounds and evaluate the integral, we obtain the bound

$$p_{patmatch}(q) \le n!(4.752)^{n-2} L\Delta L\gamma^{2n-4} \qquad (11)$$

In order to get a bound on p_{match}, let us take a pattern q, and let r_0 be the chance of no random matches in the entire sky, r_1 be the chance of exactly one random match, etc. If E is the expected number of matches, we have $E = r_1 + 2r_2 + 3r_3 \ldots \ge r_1 + r_2 + r_3 \ldots = p_{match}$. So, the expected number of matches is an upper bound. In practice, it should be a fairly good one, since random matches are rare to the extent that even getting two random matches is very unlikely.

We can now compute the expected number of matches as $\binom{N}{n} p_{patmatch}$, where N denotes the number of stars in the sky we are using. This gives us

$$p_{match} \le \binom{N}{n} n!(4.752)^{n-2} L\Delta L\gamma^{2n-4} \qquad (12)$$

Plugging into the equation for redundancy, we get $\int_Q \binom{N}{n} n!(4.752)^{n-2} L\Delta L\gamma^{2n-4} \frac{2n-2}{V'_n} \frac{L^{2n-3}}{L_m^{2n-2}} dL dX' dY'$ This is a big integral, but most of the terms are constants, and the $dX'dY'$ part just integrates out since the only variable the shows up is L. This will cancel out the factor of V'_n in the denominator, giving a result of $\binom{N}{n} n! \frac{(2n-2)(4.752)^{n-2}}{L_m^{2n-2}} \gamma^{2n-4} \int_0^{L_m} L^{2n-2} \Delta L dL$

There are two cases to consider. In the case of known FOV, $\Delta L = \Delta L_c$, which is a constant. Evaluating and substituting in for the definitions of γ and ΔL_c, we get the bound

$$2\binom{N}{n} n! \frac{(2n-2)}{2n-1} (4.752)^{n-2} \frac{(F_{est} e_{c\max})^{2n-3}}{(1-e_{f\max})^{4n-7}} L_m \qquad (13)$$

On the other hand if we don't know the FOV and $e_{f\max} > 0$, then we get $\Delta L = Le_{f\max}$ This gives a final bound on Tetra's failure or false positive probability of

$$\binom{N}{n} n! \frac{(2n-2)}{2n} (4.752)^{n-2} \frac{(F_{est} e_{c\max})^{2n-4}}{(1-e_{f\max})^{4n-8}} e_{f\max} L_m^2 \qquad (14)$$

RESULTS

We tested C implementations of the star identification algorithms in simulation and against real images of the night sky. We also entered Tetra into the European Space Agency's star identification competition, "Star Trackers: First Contact."

Simulation

Monte Carlo simulations were performed using a 2015 Asus K501UX laptop with an Intel Core i7-6500U CPU, a Hitachi HFS256G39MND-2300A solid-state drive, and 8 GB of DDR3 RAM. Simulations used a 6.2 minimum star magnitude for both the catalogs and generated images. Stars within .004 radians (0.229°) of each other were removed from both. The Yale Bright Star Catalog 5 was filtered with these settings, yielding 5,904 stars. We used the extracted stars in both the catalogs and generated images, providing 99.82% sky coverage of four or more stars. Test images were generated 1024 pixels square with 10° horizontal FOV, uniformly random attitude over the celestial sphere, and a maximum of 12 stars.

Pyramid and ND Star ID, as well as their improved counterparts, were tested against Tetra in runtime, failure rate, and false positive rate. The failure rate is how frequently the algorithms did not return a solution. And the false positive rate is how frequently the algorithms incorrectly identified stars in the image. All algorithms were tested against uniform centroiding error of given maximum pixel radius and fixed error in their estimates of the FOV. Gaussian centroiding error and false stars were not used in the simulations, as they would mask the infrequent errors caused by fundamental algorithmic issues, which this simulation was intended to detect. Additionally, the ESA Competition detailed in the next section already validates Tetra's performance in the presence of Gaussian error and false stars.

Up to one pixel of centroiding error was added to 10,000,000 images for the centroiding error test. Tetra and Pyramid were both configured to account for at most one pixel of centroiding error, while ND Star ID, which has unbounded error, was configured with the values given in its original paper. Figure 8 shows the failure rates of the algorithms. Tetra successfully identified every image, while Pyramid failed to identify more than 3,000 of the images, and ND Star ID failed to identify nearly every image given even small amounts of centroiding error. The majority of Pyramid and ND Star ID's failed identifications result from their requirement that star triangle matches be unique before searching for a fourth star match. Their comparatively loose error bounds are a secondary source for their failure rates. Figure 9 shows the false positive rates of the algorithms. ND Star ID quickly asymptotes to its failure probability given images consisting solely of fake stars, as it fails to identify most of the images. Pyramid's incorrect identifications result from degenerate triangles becoming specular. In contrast, Tetra correctly identified every image, as it resolves ambiguities like specularity.

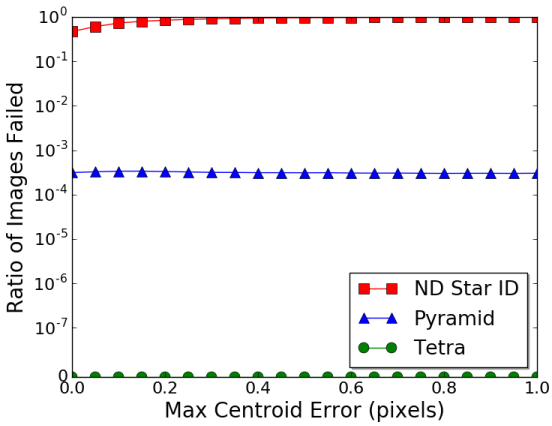

Figure 8: Centroid Error Failure Rates

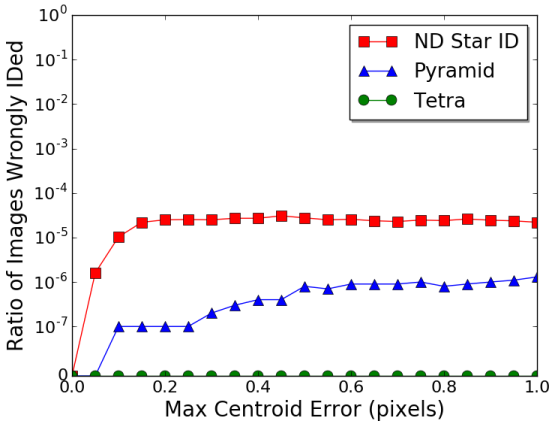

Figure 9: Centroid Error False Positive Rates

The algorithms' average runtimes per image in the absence of error are given in Table 1. The implementations of ND Star ID and Pyramid based on their original papers took around one millisecond to identify each image, while their hash table optimizations were approximately an order of magnitude faster. Tetra was the fastest, identifying images in a hundredth of a millisecond, approximately two orders of magnitude faster than ND Star ID and Pyramid.

Table 1: Simulation Runtimes Per Image

	ND Star ID	Pyramid	Tetra
Original Paper Implementation (ms)	1.67	0.91	0.014
Optimized Implementation (ms)	0.39	0.053	0.014

For the FOV test, uniform centroiding error of at most .2 pixels and FOV estimate error of up to 5% was added to 1,000,000 images. Tetra was reconfigured from only accounting for 1 pixel of centroiding error to also accounting for at most 5% FOV estimate error. Figure 11 depicts the algorithms' failure rates, while Figure 12 shows their false positive rates. ND Star ID failed to identify 85% of the images, increasing slightly to 87% at the maximum FOV errors. At 1% FOV error, Pyramid's runtime increased by two orders of magnitude and it returned as many false positives as correct answers. On the other hand, Tetra had no false positives and only failed approximately .1% of images.

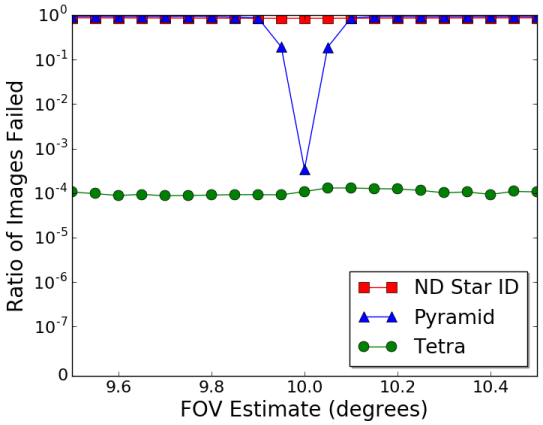

Figure 10: FOV Error Failure Rates

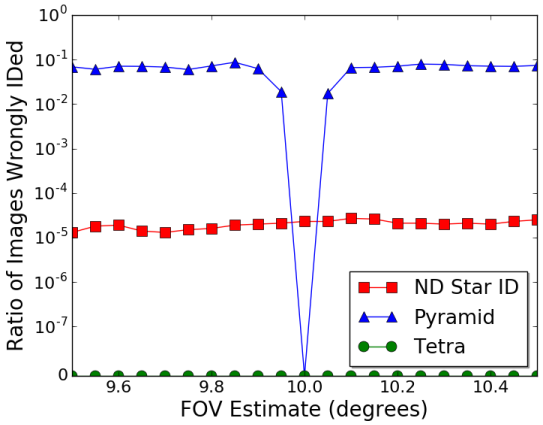

Figure 11: FOV Error False Positive Rates

ESA Competition

The European Space Agency is currently hosting a star identification algorithm competition, "Star Trackers: First Contact" (https://kelvins.esa.int/star-trackers-first-contact/). Participants are tasked with identifying 10,000 night sky images with 10 degree FOV containing an average of 5 real stars and 19 false stars. Position and magnitude information are given for each star and each image is worth 1 point. Of the 10,000 images, 9,921 contain at least one star, which means a perfect score is 9,921. The perfect score is shown at the top of the leaderboard in Figure 12. However, 213 of the images contain only one star, so the maximum reasonably obtainable score is 9,708. Furthermore, 36 of the two star images have magnitudes too close together to be reliably disambiguated, resulting in Tetra's score of 9,672. It's worth noting that the 36 ambiguous two star images could be identified by "guessing and checking." However, we felt this was not in the spirit of the competition and refrained from making repeated "guess and check" submissions. The current leaderboard is shown in Figure 12. Tetra is in first place with a score of 9,672, well above the competition.

Real World Testing

Real world testing was conducted using an Aptina MT9P011 CMOS with a FOV of approximately 10 degrees. 472 images were taken, 100 of which used sidereal tracking, and 150 of which used a smooth slew. 11 of the images contained fewer than the four stars necessary for high accuracy identification and were not included in the test. The data set should therefore be representative of the two possible calibrated, lost-in-space situations: when the spacecraft has recently been deployed and may be rotating, and when the spacecraft has already stabilized, but has been offline and thus does not know its attitude. Tetra and the hash table optimizations of Pyramid and ND Star ID were run on the CMOS images using the same databases and settings as in the FOV error simulation. Results are presented in Table 2. ND Star ID failed to identify the majority of the images, while taking about the same amount of time per image as Pyramid. Tetra identified the most images by a significant margin while also

Name	Submissions	Last Submission	Best Submission	Best Score
	Perfect			9921.0
Tetra	2	April 18, 2017, 6:45 a.m.	April 18, 2017, 6:45 a.m.	9672.0
wogan1	15	April 9, 2017, 1:29 p.m.	April 6, 2017, 2:59 p.m.	9483.0
tsiolkovski	5	April 17, 2017, 5:04 a.m.	April 17, 2017, 5:04 a.m.	9473.76796112384
TDelabie	3	Oct. 31, 2016, 3:49 a.m.	Oct. 31, 2016, 3:27 a.m.	9352.74615671006
Multi-Poles_Algorithm	19	April 4, 2017, 4:43 p.m.	April 4, 2017, 4:43 p.m.	9163.61985396354
	Pyramid			7930.45081683023
ineaugh	3	March 18, 2017, 4:29 p.m.	March 17, 2017, 4:05 p.m.	-9999.0

Figure 12: ESA Star ID Competition Leaderboard as of 4/18/17

running approximately five times faster than the hash table optimizations of Pyramid and ND Star ID.

Table 2: Real World Runtimes and Robustness

	ND Star ID (This Work)	Pyramid (This Work)	Tetra
Total Runtime (ms)	359	285	66
Runtime Per Image (ms)	0.779	0.618	0.14
Identified	8	403	440
Failed	453	58	21
Ratio Identified	1.8%	87.4%	95.4%

CONCLUSIONS

We have demonstrated theoretically and experimentally that current star identification algorithms' runtimes can be vastly improved by using hash tables in their implementations. We also detailed the theory and operation of Tetra, our free and open source calibrationless lost-in-space star identification algorithm. We bounded Tetra's error rates and experimentally demonstrated its robustness and speed in simulation, in a competition, and with real world images. In particular, we showed that even after improving previous algorithms' runtimes by an order of magnitude, Tetra still outperforms them in every measured metric.

Tetra opens the door for a new generation of star trackers. Acquisition time, which is the time it takes a star tracker to produce its first attitude solution after startup, is no longer limited by solving the lost-in-space problem. Using Tetra, spacecraft cameras are now capable of rapid self-calibration on-orbit, improving pointing precision without careful alignment on the ground. Tetra also enables star trackers to be built more compactly, as they no longer need RAM to operate. The possibilities are staggering.

FUTURE WORK

We plan to extend this work to recursive star identification algorithms, specifically by mapping stars into a hash table by their right ascension and declination values. Additionally, we plan to demonstrate improvements to the Spherical Polygon Search recursive star identification algorithm[7] using hash tables. Tetra is effectively a generalization of hash tables to sets of data containing noise and ambiguity, so we plan to apply it to fuzzy logic search problems beyond star identification. We are also currently building and open sourcing a compact, low-power star tracker based around Tetra which can be assembled for under two hundred dollars, three orders of magnitude less than the cost of a compact commercial star tracker.[9]

ACKNOWLEDGEMENTS

We would like to thank Professor Daniele Mortari and Doug Sinclair for their generosity and help in one-on-one discussions. We would also like to thank the entirety of the Space Systems Academic Group at the Naval Postgraduate School for motivating this work. And we extend our deepest appreciation to Professor Kerri Cahoy and MIT's Space Systems Lab for providing the direction and guidance that made this work possible.

REFERENCES

1. Hegel, D., "FlexCore: Low-Cost Attitude Determination and Control Enabling High-Performance Small Spacecraft" *AIAA/USU Conference on Small Satellites*, Aug. 2016.
2. Terma Space, "Terma HE-5AS Star Tracker Datasheet," Feb. 2012. Available: https://www.terma.com/media/101677/star_tracker_he-5as.pdf
3. Mortari, D., Samaan, M.A., Bruccoleri, C., and Junkins, J.L., "The Pyramid Star Pattern Recognition Algorithm," *ION Navigation*, Vol. 51, No. 3, Sep. 2004, p. 171-183.
4. Hong, J., and Dickerson, J. "Neural-network-based autonomous star identification algorithm." *Journal of Guidance, Control, and Dynamics*, 23.4, Jul. 2000, p. 728-735.
5. Samaan, M.A., Mortari, D., and Junkins, J.L., "Non-Dimensional Star Identification for Un-Calibrated Star Cameras," *AAS The Journal of the Astronautical Sciences*, Vol. 54, No. 1, Jan. 2006, p. 95-111.
6. Lang, Dustin, et al. "Astrometry.net: Blind Astrometric Calibration of Arbitrary Astronomical Images." *The Astronomical Journal* 139, 2010, p. 1782.
7. Spratling, B.B. and Mortari, D. "A Survey on Star Identification Algorithms," *Algorithms, Special Issue: Sensor Algorithms*, Vol. 2, No. 1, Jan. 2009, p. 93-107.
8. Cormen, Thomas H., et al. "Introduction to Algorithms Second Edition." 2001, p. 221-252
9. Sinclair Interplanetary, "ST-16RT2 Star Tracker Datasheet," Sep. 2016. Available: http://www.sinclairinterplanetary.com/startrackers/star%20tracker%20RT%202016c.pdf