STAR-ND

(MULTI-DIMENSIONAL STAR-IDENTIFICATION)

A Dissertation

by

BENJAMIN BARNETT SPRATLING IV

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2011

Major Subject: Aerospace Engineering

STAR-ND

(MULTI-DIMENSIONAL STAR-IDENTIFICATION)

A Dissertation

by

BENJAMIN BARNETT SPRATLING IV

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2011

Major Subject: Aerospace Engineering

ABSTRACT

Star-ND (Multi-Dimensional Star-Identification). (May 2011)

Benjamin Barnett Spratling IV, B.S., Auburn University

Chair of Advisory Committee: Dr. Daniele Mortari

In order to perform star-identification with lower processing requirements, multi-dimensional techniques are implemented in this research as a database search as well as to create star pattern parameters. New star pattern parameters are presented which produce a well-distributed database, required by the database search algorithm to achieve the fastest performance. To mitigate problems introduced by the star pattern selection, incorrect entries are added to the database, which reduces the number of iterations of the run-time algorithm. The associated algorithms, star pattern parameters, and database preparation are collectively referred to as Multi-dimensional Star-Identification (Star-ND).

The star pattern parameters developed may also be extended to star patterns with an arbitrarily large number of stars, while retaining the well-distributed property. The algorithm is contrasted with the current state-of-the-art star-ID algorithm, Pyramid. The database is found to grow linearly with the size of the star catalog, while Pyramid's database grows quadratically. The running time of Star-ND is found to be on average a factor of 25 times faster than the time for Pyramid.

DEDICATION

“I know the plans I have for you,” God said, “plans to prosper you and not for harm, plans for a future and hope.”

“I know that verse, and you weren’t saying that to me!”

“I’m saying it to you.”

“Ok, but only if you promise.”

“I promise. I love you, son.”

And for six years, that has been my core assurance.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Mortari, and my committee members, Dr. Junkins, Dr. Friesen, and Dr. Reed for their guidance and support throughout the course of this research, and to Dr. Turner for filling in at the last moment during the defense. Thanks especially to Dr. Reed for the encouragement to pursue and persevere in the Ph.D. program, and the opportunity to work on AggieSat-2, and to Dr. Mortari for both the flexibility to pursue my research interests and for the opportunity to share in his research. Thanks to Dr. Tom Pollock for his technical advice in image processing algorithms, I avoided many sleepless nights. Thanks also to Pedro Davalos for tirelessly regenerating VST images with lots of different camera specs.

Thanks especially for encouragement when the going was tough from my parents, Ryan Goodnight, Ben Grimme, Jeremy Schwab, Daniel Dixon, Fernando Morales, and Whit Meyer.

Thanks to the people and programs that kept my bills paid while I pursued my ever-changing research projects: my father, Dr. Reed with AggieSat-2, Dr. Mortari, and Chip Hill at SERC. And last, but certainly not least, to Karen Knabe, whose guidance through the maze of university policies and recipe for wheatless oat cookies, kept me sane.

NOMENCLATURE

d	number of dimensions of data
FOV	field of view
f	the number of stars in a FOV
k	the number of correct answers for a search
K-Vector ND	multidimensional k-vector
lg n	$\log_2 n$
n	number of items in a catalog or database
p	the number of stars in a star pattern
$\hat{s}$	a unit-vector in the direction of a star

TABLE OF CONTENTS

LIST OF FIGURES

Page

Page

LIST OF TABLES

CHAPTER I

INTRODUCTION: SPACECRAFT AUTONOMOUS ATTITUDE DETERMINATION

The requirement for attitude (orientation) information of a spacecraft has been the mother of invention of many devices and algorithms, notably the process of autonomously identifying stars (star-ID). Although there is a long history of devices used to identify stars and compute an attitude without using a star camera, this dissertation will focus on algorithms that use a star camera with an imaging array and an algorithm to match observed (body) directions of stars with catalog (inertial) directions of stars without requiring reorienting the camera or the spacecraft. Such algorithms fall into two basic categories, *lost-in-space* algorithms, in which no information regarding the attitude of the spacecraft is available, and *recursive* algorithms, in which some information regarding the attitude is available. In both cases, the star-ID algorithm typically uses the invariance of angles in orthogonal transformations. Specifically, it mainly uses inter-star angles (the angle between the line-of-sight of two stars from the perspective of a camera), but also the brightness of the stars, and some computations of these values to distinguish stars. A further subcategory of both categories is *non-dimensional* star-ID, in which the exact angular separations are not required but the values are normalized to be insensitive to poor camera calibration, or time-varying camera calibration parameters [1-4].

This dissertation follows the style and format of *AAS Journal of Astronautical Sciences*.

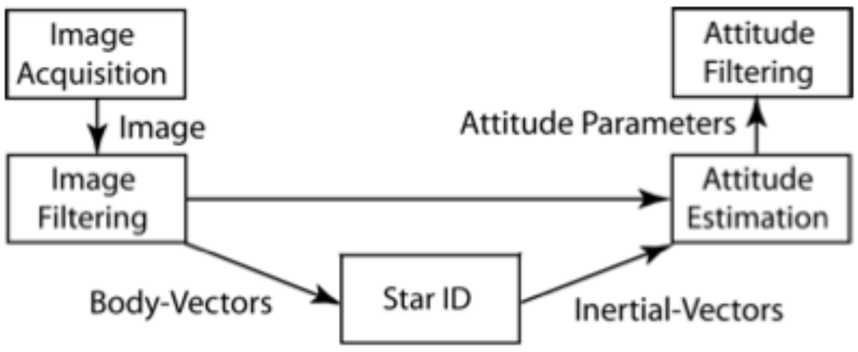

Figure 1 Typical spacecraft autonomous attitude determination process

Star-ID is just one step in the process of determining the attitude of a spacecraft, as depicted in Figure 1. The first algorithms in the sequence measure body-frame vectors for the locations to observed stars. The primary purpose of the star-ID process is to determine the inertial-frame vectors for some or all of the body-frame vectors given to it. Subsequent algorithms determine a suitable transformation that correctly maps the body vectors to the inertial vectors, thereby calculating the spacecraft's attitude. When sensing the stars with a CCD imaging array, the information available for the identification process is the brightness of the star and the angular separation between stars.

The star-ID task has three basic pieces, with an optional fourth, as illustrated in . First, an algorithm must extract features from a set of body vectors and associated brightness. Second, a database search matches a subset of the observations with entries in the database, and third makes some estimate as to the probability that they are correct. Optionally, the remaining body vectors are identified once an estimate of the attitude is available in a method called "recursive" star-ID. This fourth step typically implements a variation of the "direct match" technique in which stars are identified by their close proximity to their predicted location. The recursive mode is typically much faster than the first two, and can usually be repeated successively for additional observations with an a priori estimate of the attitude.

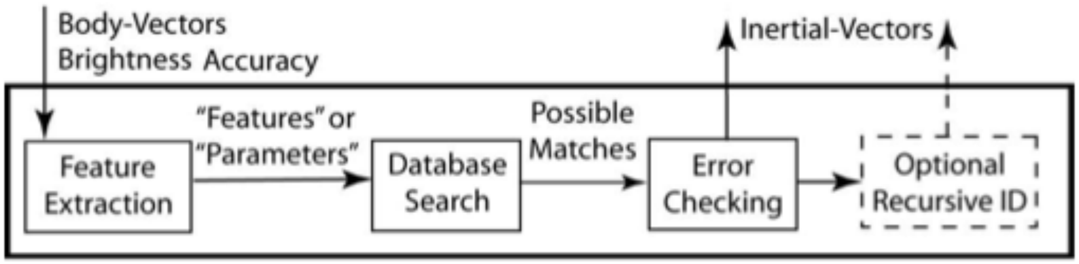

Figure 2 Typical star-ID process

The organization of this dissertation is summarized in the following:

Chapter II presents an analysis of the major advances in the history of star-ID since the first use of the star-camera for both the Lost-In-Space (LIS) case and the recursive case. The analysis reveals the deep connection between a star-ID algorithm's computational complexity, the star pattern, the star pattern parameters, and the database search algorithm. The available star-ID algorithms lack star pattern parameters suited to a multi-dimensional search, which could greatly reduce the computational complexity of the star-ID problem.

Chapter III presents a quantitatively comparative survey of available multi-dimensional range-search algorithms. The K-Vector-ND is selected as the fastest multi-dimensional range-search algorithm given one caveat: the dataset must be "well-behaved" or nearly-uniformly distributed in all dimensions in order to achieve the quoted performance. Given the history of star-ID algorithms, no such data sets are available.

Chapter IV introduces novel star-pattern parameters, using a traditional nearest-neighbor type star pattern. This pattern choice stands in stark contrast to other, more robust techniques, which list all possible combinations of observable star patterns. While such a star pattern has the benefit of a small catalog, owing to its one-entry-per-star policy, it also has a fundamental robustness problem associated with angular & brightness errors. Even foreseeable errors can cause excess iterations of the run-time algorithm or failure altogether. To mitigate these problems, the author suggests inserting carefully-crafted false entries into the database to reduce the number of iterations, and in many cases avoid failure. The final Multi-Dimensional Star-Identification (Star-ND) run-time algorithm and database building techniques are presented in Chapter V.

The Star-ND algorithm is tested for performance through several tests, including simulated images in Chapter VI. When compared to a modern cutting-edge algorithm, Star-ND proves to perform an average of a factor of 25 times faster. Guidelines for a star-tracker subsystem designer implementing the Star-ND algorithm are given in Chapter VII. Suggested directions for future research and concluding remarks are delivered in Chapter VIII.

CHAPTER II

LITERATURE REVIEW: STAR-IDENTIFICATION ALGORITHMS

The algorithms surveyed in this chapter are evaluated by the analytical asymptotic performance of

1. the feature extraction step,

2. database search, and

3. their utilization of independent pattern features in the star features based on how many stars are used in a pattern. Although a star pattern containing p stars has $p(p - 1)$ inter-star angles which may be measured, the number of independent 2 pattern features for a star pattern of p stars, is only $(2p - 3)$, since some inter-star angles are dependent on the values of others. By including star magnitude information, which is typically much less precise than inter-star angle information, an additional p independent pattern features may be used. Throughout this dissertation, the use of asymptotic notation [5], $O(k)$, is used to indicate the highest-possible order term in the running time of an algorithm. The accompanying notation, $\Theta(...)$, is used to represent a known highest-order term. Algorithms are analyzed based on the described implementation in the referenced papers. Any assumptions drawn about their performance time, when not directly stated in the paper, is based on the progression and availability of certain algorithms at the time, although alternative running times are discussed if it seems likely they could have been implemented.

2.1 The Beginning

After the first CCD-based star tracker was developed by Salomon in 1976 at JPL [6], Junkins, Turner, Strikwerda [7, 8], and others began work on implementing an algorithm that could identify stars in real-time. While they realized the benefit of using the easily-computable sine of inter-star angles as a pattern feature, the key problem that arose was the matching of observed inter-stars to the items in the database. After several years of work and a few conference papers, Junkins et al. published "*Star Pattern Recognition and Spacecraft Attitude Determination*" in 1981 [9]. Although the algorithm was able to identify star triplets, it had the primary limitation of requiring an *a priori* estimate of the spacecraft's attitude before it was able to perform in real-time. The reason is that Junkins had used "sub catalogs" of the sky, illustrated in Figure 3, each representing a portion of the sky, in order to accelerate the computation. Although the method was perfectly robust to non-stars because the catalog included all combinations of stars that might be observed, it only updated the attitude estimation algorithm once or twice a minute, as contrasted with the angular rate sensors, which updated at 1,000 times per minute. The majority of the attitude estimation was propagation of the angular rate sensors, and periodic checks were established to confirm and improve the propagated attitude. Junkins' feature extraction runs very fast, in O(p) time, since it may select any three of the observed stars and measure the sine of their inter-star angles. However, since his database search considers every possible permutation of stars available in a given region of sky, the search time is $\Theta(f^3)$. The time to perform star-ID increases if the stars are not located in the predicted field of view as other sub-catalogs are searched. The database itself would grow as O($n\ f^2$).

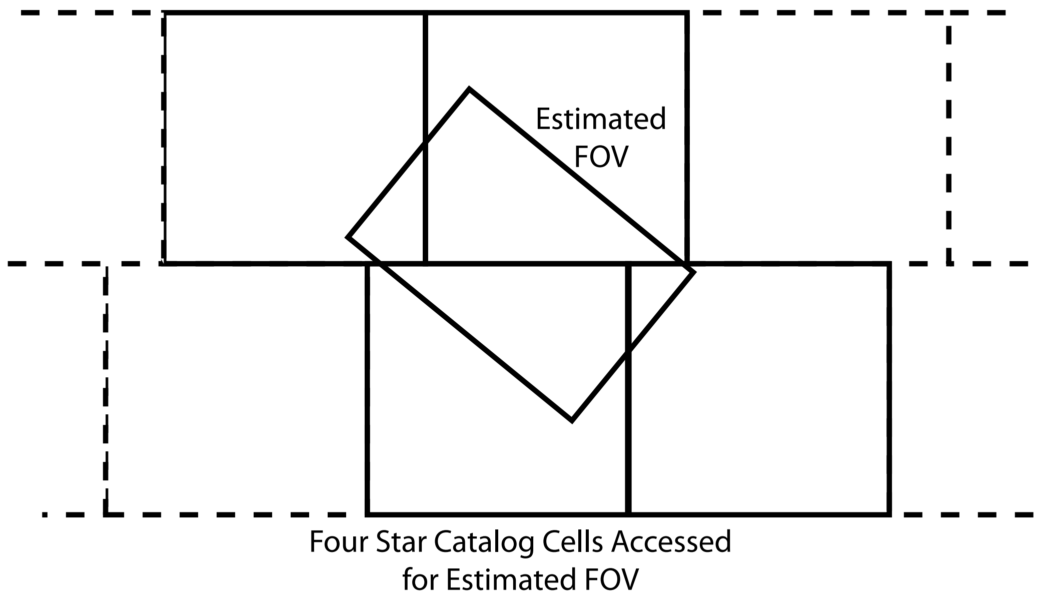

Figure 3 Field-of-view-sized sub-catalogs (from [9])

In 1986, Groth [10] suggested that a faster way to search the sub-catalogs would be to sort the triangles' sides in order based on permutation-invariant values such as the logarithm of the perimeter of a triangle. He admits, however, that his algorithm runs at high polynomial power of n, much as Junkins'. Groth's algorithm differs in that the performance has a lower constant factor. While the asymptotic order of the database size is identical to Junkins, there would be a constant factor increase of data included, associated with the permutation-invariant values.

In 1987, Sasaki [11] and others published a patent showing how to improve the search time by using the area of a star triangle and the sum of the luminous intensities as preliminary markers in performing the star identification, using $\Theta(p)$-time for star feature extraction. His method does not discuss the way in which the database is searched, requiring only that a "number of stars" will be selected from the database by a method using "parallel, serial, or the like" processors. It is not noted whether his database indeed contains as many star triplets as does Junkins' method, nor is the search procedure described.

Later, in 1989, Van Bezooijen [3] suggested in his dissertation that the speed of the star pattern recognition algorithm could be improved by making more use of the available information in the star patterns. Van Bezooijen discussed directly the relationship between the number of stars and the amount of information available from a star pattern with a given number of stars. His analysis also included a very detailed statistical probability that a star had been identified correctly. Unfortunately, Van Bezooijen's method sometimes required the spacecraft to slew in order to detect stars for his star-ID method, and as such, his work is not covered in depth here.

In 1991, with Junkins on his advisory committee, David Anderson [12] addressed the ambiguity of the order of star triplets by proposing a permutation matrix, and the development of star pattern parameters that were independent of the order in which the stars are selected. Sticking with the tried-and-true star-triple pattern, Anderson also proposed the use of an array processor to handle the matrix multiplications required to use his permutation matrices. Unfortunately, the database storage remained $\Theta(n f 2)$, and there was no advance made on the database search. Anderson suggested that an array processor be used to perform the matrix multiplication, decreasing the running time of the star-ID process. The design engineer should note that array processors, while performing a comparatively large number of computations when contrasted with a serial processor, also use a comparatively large amount of power, because they both tend to use identical amounts of energy to perform each computation.

2.2 Search Process Acceleration

The next year, Liebe [13] made the critical connection of the feature selection process to the database search time, making the Lost-In-Space problem tractable. Liebe suggested the use of the two closest stars to a selected star as components of the star pattern, and the angular separations from the two closest stars and the angle between them as his parameters, as illustrated in Figure 4, and addressed the situations in which predicted stars would not be seen due to their magnitude being very close to the detection threshold. Liebe also addressed the situation in which small errors would cause the incorrect selection of the closest two stars when the distance to these stars were similar. Although Leibe's feature extraction process now took O(f lg p)-time to compute, his database could be reduced to O(n), and subsequently, his database search could be performed much faster, though still linear-time. Liebe makes full use of the available angular independent pattern features, neglecting the stellar magnitudes. Liebe later implements the optional recursive direct-match mode, which could identify the remaining stars up to 20 times faster than the Lost-In-Space algorithm.

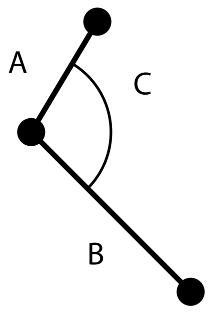

Figure 4 Liebe's parameters, 2 inter-star angles and 1 interior angle (from [13])

Then in 1993, Baldini [14] proposed a multi-step star-ID method. Baldini's method identifies the brightest *p* stars in a given image, requiring O(*f* lg *p*) time. He then measures the angular separation of the sequence of five stars, O(*p*) time. Baldini then proceeds through a linear search of the catalog search for every star in the catalog which falls within an acceptable tolerance range of the observed stars, requiring O(*p n*)-time, neglecting the time for dynamic list creation, although this step was later improved by other algorithms in the future. Using the symbol Δm to represent the fraction of stars in the catalog that fall within the acceptable range, Baldini obtained an intermediate result of O(Δm *n*) stars in each of *p* lists. He then compares the distance of each star in adjacent lists determining if any star cannot have an angular separation within the tolerance of the observed angular separation, requiring O ($p(\Delta m\ n)^2$)-time, although as he points out, as items are eliminated the number of comparison at each iteration is reduced. Baldini is then left with *p* lists containing stars that meet their neighboring distance criteria; he then forms all combinations of the stars, discarding combinations whose sequence of angular separations does not match the observed stars. The running time of this step is represented analytically by O($a\ p$), assuming there are approximately a stars in each of p lists. It is certain that the rejection of certain combinations in the second or third step is sure to reduce the total number of comparisons, and Baldini is certain to conclude with only one or two possible combinations of stars. Baldini has used five stars, inherently containing twelve independent features, but uses only nine when performing his identification process, suggesting the required field of view may be larger for Baldini's method when compared to other methods to be sure that there will be enough visible stars. Although the process will weed-out non-stars, the addition of non-stars to the algorithms increases most of the steps linearly or quadratically.

In 1995 Ketchum [15] proposed a second sequential filtering algorithm, this time using the brightness of the brightest star to attempt to determine the likelihood of pointing in any particular direction. She then filters the list of possible stars using the brightness of the second brightest star. Although she admits the algorithm would need to search as much as 43% of the catalog for the appropriate stars, she notes that the storage space required by her algorithm is much less than required by Van Bezooijen's method. Furthermore, Ketchum is one of few to give direct empirical data regarding the running-time of her algorithm, reporting it requires up to 63 seconds to run on a 50 MHz processor.

Later in 1995, Scholl [16] published a more straightforward method. The inter-star angles were to be ordered by their relative brightness, eliminating the permutations that arise when considering the possible orders of three stars. Unfortunately, Scholl retains the O($n\ f^2$)-sized catalogs, and does not specify the search technique used. While it is true that his method uses less time to search the database when compared to Baldini, it is nonetheless still O($n f^2$), since faster techniques were not proposed until the following year.

2.3 Search Time Dramatically Reduced

In 1996, Quine [17] was the first to attack the database search problem head on, realizing that a binary search tree (see Figure 5) could be used to search the database in O(lg n)-time. He retains Liebe's use of the two closest stars to a given star to form a pattern, resulting in a database of size O(n), instead of previous catalogs that used all observable combinations of stars. While this increases his feature extraction time to O(f lg b), the trade off is quite advantageous for large values of n.

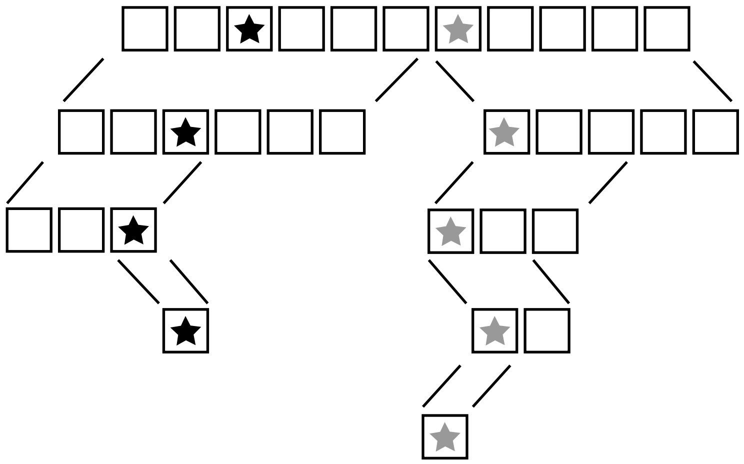

Figure 5 Quine's binary tree (from [17])

2.4 Novel Grid Algorithm

In 1997, Padgett [18] published perhaps the first novel star pattern recognition algorithm, which actually used a star "pattern." Padgett's grid algorithm cast the locations of neighboring stars as items on a loose grid (see Figure 6). Padgett solved the roll ambiguity by rotating the observed stars about a given star until the nearest star to the given star was aligned with the *x*-axis. The cells in the grid were then considered to be "on" if there was a star located inside it, and "off" if there were not. The locations of the "on cells" then become the features, the indexes of the "on cells" were listed as items in a vector. His feature extraction time became O(*f*). Unfortunately, Padgett was unable to improve the database search time for his features beyond linear, and the resulting database search remained O(*n*). Further analysis indicated that Padgett's method was quite robust to the presence of non-stars owing to its large grid cell size when compared to the angular error in star position.

2.5 Search Time Reduced Much Further

Later in 1997, Mortari [19, 20] proposed an even faster database search technique, the *k*-vector, constituting the core of the "Search-Less Algorithm," (SLA) star-ID. Mortari retained the approach of selecting any pair of stars in the field of view, O(*p*)-time, but he proposed using a "*k*-vector" to search the database in an amount of time <u>independent of the size of the database</u> [21].   shows the *k*-vector construction for a 10-element database. The small horizontal lines are equally spaced and they give the *k*-vector values: 0, 2, 2, 3, 3, 5, 6, 8, 9, 10.

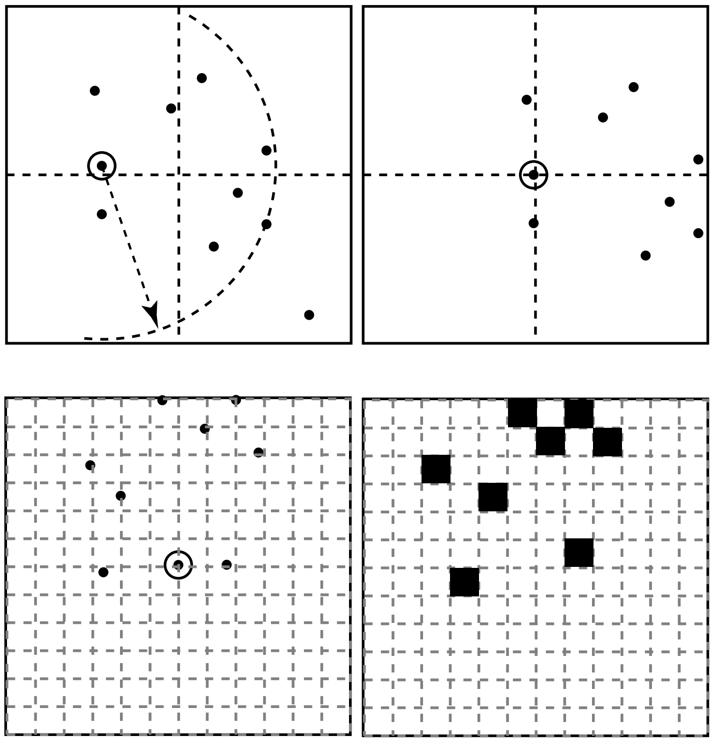

Figure 6 Illustration of Padgett's grid algorithm (from [18])

The search time for a single star-pair would be O(*k*), where *k* is the number of possible star pairs with inter-star angles within the measurement tolerance. Unfortunately, the dominant time of the algorithm came when comparing multiple lists of stars returned for each inter-star angle. Since multiple stars had to have all their inter-star angles confirmed to be a match, the running time of the comparison would be O($p\ k^2$), and *p* is the number of stars in the pattern required to guarantee uniqueness. Even though the resulting value of *k* would be a number based on the uncertainty associated with the inter-star angle measurement and the number of observable star pairs, Mortari had made the first important step in breaking the dependence of the database search time on the size of the database. Mortari's method could also reject a single non-star from a set of selected stars, without losing the progress made in identifying the others. The resulting *Search-Less Algorithm* (SLA) was then successfully tested on orbit on an Indian satellite [22].

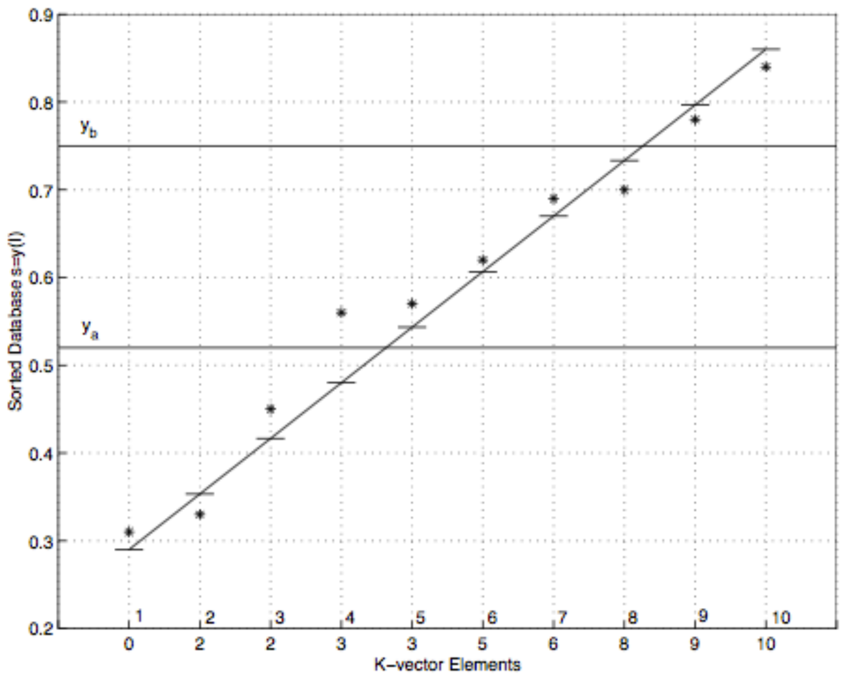

Figure 7 Example of k-vector construction (from [21])

A few years later, realizing that the robustness to non-star "spikes" was essential towards reducing the number of iterations of his algorithm, Mortari developed the "*Pyramid*" algorithm [1] which uses an optimal permutation algorithm to exploit the ability of his algorithm to select which stars to match. This permutation is written to minimize the time spent considering stars that don't match, fearing them to be non-star spikes. The code has been tested to reject non-stars in an image containing only five real star but with 63 non-stars thrown in. The *Pyramid* algorithm has been successfully tested on Draper's "Inertial Stellar Compass" star tracker [23] and on MIT's satellites HETE and HETE-2 [24]. This algorithm is presently under exclusive license to StarVision Technologies.

Neural networks, have been proposed for use in Star ID as early as 1989, [25]. In 2000, Hong [26], proposed using a neural network and fuzzy logic to identify the stars, as illustrated in Figure 8. Hong used the popular ordered triple, based on star brightness, and fed the resulting angular separations into a neural network. While his feature extraction process runs very fast, O(1), he is forced to use a massively parallel architecture to implement the neural network. Though such techniques may be used with much success on ground-based systems, it is uncertain if this technique is the best for use in a system with limited electrical power, or that requires expensive radiation-tolerant hardware. Hong notes quite accurately that his algorithm performs much faster than some other of the mentioned algorithms, referencing Van Bezooijen, Quine and Ketchum, but failed to make a comparison with Mortari's more efficient method. Hong readily admits that his technique requires more than a quarter-million multiplications.

2.6 A Number of Dubious Proposals

Then in 2007, Guangjun [27] proposed a feature extraction technique, similar to Liebe [13], using the inter-star angles and the angle made by two stars relative to a central star. Though his feature extraction time is O($f \lg p$), he uses a linear database search, performing bit-by-bit comparisons, running in O(n) time. While Guangjun's claim [27] is true that his algorithm runs faster than Padget's grid algorithm [18], similarly to Hong [26], he fails to compare his algorithm to more-recent faster algorithms.

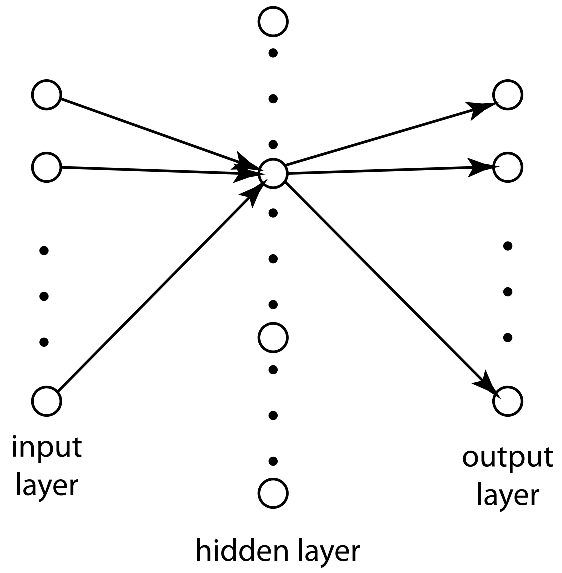

Figure 8 Hong's neutral network (from [26])

In 2008, Kolomenkin [28] proposed a modification of the SLA algorithm to reduce the time spent cross-checking the results of the k-vector. In the original SLA algorithm, Mortari selects any four stars in the image and performs six k-vector searches to find six lists of approximately $k = 100$ candidate star pairs. The cross checks take O(k^2)-time, and this cross-check step is the bulk of the time used by the SLA algorithm. Kolomenkin suggests, however, finding all lists of all possible stars in the image (which is O(f^2)-searches) and maintaining a list of the number of times each visible star is listed in a particular search result, O($f\,k$) items). Once all the inter-star angles have been determined and the stars voted, the visible-catalog star match with the largest number of votes is considered to be the correct answer, and the results are filtered from there. This latest step would have running time O($f\,k$). While the algorithm does perform the cross check O(k/f) faster than the SLA, it calculates O(f^2) more inter-star angles, and k-vector searches, each of which takes O(k)-time, contributing an increase of O($k\,f^2$)-time. For the purpose of this analysis, the time for list insertion for keeping track of voting is assumed to be O(1), though in practice it is difficult to perform this step in less than O(lg k)-time. So Kolomenkin's modification would run asymptotically faster in systems for which $f^2 < k$. Since, in a given system, f tends to be on the order of 10 to 40 and k on the order of 100, it seems dubious the algorithm achieves any real decrease in asymptotic running time, most likely if any improvement is achieved, it is by a constant factor. In the paper, Kolomenkin did not provide any direct performance comparison to the unmodified SLA.

For the reader's convenience, the major advances in asymptotic performance of star-ID are listed in Table 1.

2.7 Non-dimensional Algorithms

In 2003 Samaan, along with Junkins and Mortari [4], presented a new star-ID technique that was robust to calibration errors. For flight systems in which temperature variations would cause cyclic variations in the accuracy of the calibration, the new technique would promise to eliminate the ambiguity in matching star patterns. Instead of using the inter-star angles between stars in a triangle, Samaan used the triangle's interior angles, the angle between two stars, with a third star as a vertex. While the inter-star angles respond linearly to changes in star-camera temperature, the triangle interior angles are invariant in the first order of the distortion, as illustrated in Figure 9. Samaan's technique uses the smallest and largest of the interior angles to place stars in a catalog, so the feature extraction time is O(lg p). The database is subsequently searched with Mortari's k-vector searching technique, taking O(k) time. Samaan's numerical tests found that at least five stars must be matched before the technique produces, which introduces a cross-checking routine, using O($p\ k^2$)-time. Samaan concludes the paper by using star-ID to re-calibrate the camera, and thereafter use a more efficient Star-ID method.

Table 1 Major Advances in Star-ID Algorithms

Author	Year	Feature Extraction	Database Size	Database Search	Validation	Used measurements /Available
Junkins	1981	O(b)	O($n\,f^3$)	O(f^3)	N/A	3/3
Liebe	1992	O(f lg b)	O(n)	O(n)	N/A	3/3
Baldini	1993	O(f lg b)	O($n\ b$)	O($b(\Delta mn)^2$)	O(a^b)	9/12
Scholl	1995	O(b lg b)	O($n\,f^2$)	O($n\,f^2$)	O(k)	6/6
Quine	1996	O(f lg b)	O(n)	O(lg n)	N/A	6/6
Padgett	1997	O(f)	O(n)	O(n)	N/A	$2f/2f$
Mortari	1997	O(b)	O($n\,f$)	O(k)	O($b\ k^2$)	6/6
Kolomenkin	2008	O(b)	O($n\,f$)	O(k)	O($k\,f^2$)	6/6

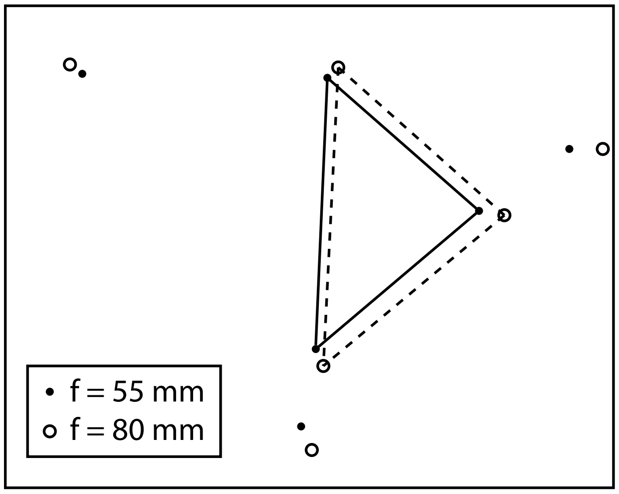

Figure 9 Image-plane distortion from calibration variations (from [4])

Rousseau also published a method in 2005 [29], which he billed as being robust to errors introduced by new CMOS Active Pixel Sensors (APS). His metric is the sine of star-triangle interior angles, but instead of using any combination of stars, he used only the closest two stars, and used only one of the three (two independent) interior angles as a parameter. His pattern selection pattern means there is only one entry in the catalog for each star; so his catalog size is O(*n*). It also follows that the feature extraction time is O(*f* lg 2) = O(*f*). Furthermore, Rousseau does not specify a method for selecting star triangles from the catalog, but according to his published parameter distribution, the fastest method available would be a binary-tree search, taking O(*k* lg *n*)-time. Rousseau then actually computes the attitude for each star triangle, and finds all the stars from the catalog that should be visible, which should take no less than O(*f*), and more likely O(*f* lg *n*). Each observation is then transformed into the reference frame. The observed stars are then matched up with catalog stars, and the inter-star angles compared. The process by which this is done is not described, but likely takes O($f \lg f$)-time. The best of the matches of all the triangles is then selected. The final analytic time of Rousseau's algorithm is then O($k f \lg f \lg n$). It is unclear whether Rousseau's performance data is on his original 45,000-star catalog, or another mentioned, reduced 1,300-star catalog, but his timed results are disappointing; all of his averages are longer than a second on a 650 MHz processor, far longer than most of the other available methods. Although the tests are performed in MATLAB, which unnecessarily increases the computation time, it is unclear why Rousseau claims the algorithm is fast from his reported data, and without any performance comparison to any other algorithm. Furthermore, he does not describe why his validation phase, which uses inter-star angles to reject incorrect matches, is more robust to APS-induced measurement errors, when the same inter-star angles are used by previous methods, like SLA. It seems likely that he simply used the smaller star catalog, in which larger measurement errors result in fewer incorrect matches. Rousseau's parameters, however, have the benefit that there is no ambiguity as to which star in the triangle is the listed star, as long as the star triangle does not contain nearly identical angles.

2.8 Nearest Neighbors

Samaan made other advances for recursive star-ID in 2005 [30], as had others [31]. His key to reducing the recursive mode time was to speed the selection of stars that ought to be visible given some other visible stars. He presented two methods, one which used the Mortari's Spherical Polygon-Search (SP-Search) [32, 33], which in turn used his *k*-vector, and the second which used a pre-built catalog of stars that should be visible if another star is visible, the Star Neighborhood Approach (SNA). The SP-Search uses a *k*-vector 3 times to find the stars within calculated x, y, and z ranges in inertial space. By finding stars in common in the three resulting lists, Mortari produces a list of stars that should be visible. Each of the three database searches takes O(k) time, while the cross-comparison takes O(k^3)-time. Samaan then uses the attitude estimate to search for the presence of these predicted stars in the set of the camera's observed stars. Samaan's other method, the SNA, constructs a table ahead of time, of the six closest stars to any given star, presuming these stars to be the most likely to be visible if the first star is found. Samaan's method takes O(p)-time to find candidate stars, if p stars are identified by the LISA. It is uncertain how many successive iterations would be necessary to ensure that all the stars in the given field of view have been found, other than it is most likely bounded by O($f\,p$).

Similar to Samaan's star neighborhood table, the Delaunay triangulation lists the nearby stars to a given star. Unlike the Samaan's data structure, the Delaunay structure ensures that all points are reachable from any other point. The Delaunay triangulation provides a powerful method for finding the nearest neighbor to a given data point, and also guarantees the uniqueness of the triangulation. The uniqueness property is used by Ziliang in 2003 [34], to reduce star triangles to a unique data set. Ziliang constructs the Delaunay triangulation of the observed stars, and then uses triangle measurements to match to triangles in the Delaunay triangulation of the star catalog. While both the database construction and image processing are then log-linear, O(n lg n) and O(f lg f), respectively, the Delaunay triangulation is extremely sensitive in many cases to perturbations in the directions to stars, and especially sensitive to the absence of stars that are in the catalog. Ziliang proposes to account for these deviations from uniqueness by removing stars from the observed vector set alternately upon failure to alter the triangle arrangements to attempt better ID. Ziliang does not discuss the method he used for searching the match of triangle properties, but mentions only that it “is arranged under specific rules, making the velocity of searching the database more rapid.” The best time achieved by this method is logarithmic, O(lg *n*). By using the Delaunay triangulation, Ziliang is able to drastically reduce the number of triangles that must be matched, thereby reducing the catalog size and searching time, but has in fact incurred serious robustness performance issues by relying on the easily perturbable unique data structure, as exemplified by his published results showing a rapid decrease in identification probability with increasing direction uncertainty. The Delaunay triangulation, however, presents a powerful tool, and its use is further examined in Chapter III.

2.9 A Historical Observation

Through the past three decades, steady advancements have been made by various authors, employing many techniques for autonomous star-ID. Although many complex arrangements have found to be successful, the trend for faster and more reliable methods saw rapid advancement in the 1990’s. Star-ID techniques tend to fall in one of four categories separated by two decisions: whether or not to use star brightness information, and whether to select any stars for a given star pattern or use ordinal information from brightness or distance. Once the goal of solving the Lost-In-Space case from a single image in real-time became possible without *a priori* information, advancements in the star-ID field turned towards improving the robustness of star-ID to errors in detecting stars and measuring their features.

Generalized pattern classification algorithms, while successful in identification, tend not to decrease the execution time of the star-ID process. Many such algorithms are based on identifying classes of objects, instead of individual objects, incurring inefficiencies due to the extremely large number of stars. Such algorithms also suppose that not every object can be measured before classification begins, which unnecessarily enlarges uncertainty bounds whereas high-precision star catalogs are publically available. Furthermore, generalized pattern classification and identification algorithms are based on identifying multiple observations from a single object, whereas star-ID concerns the relationships between otherwise indistinguishable stars. Lastly, generalized pattern classification algorithms do not sufficiently exploit the geometry of a star pattern or its inherent probability distributions. To decrease execution-time it is therefore reasonable to tailor an algorithm directly to the star-ID problem.

All of the algorithms, however, have apparently used solely one-dimensional search algorithms. Authors resort to complex error checking functions that act as intermediate filters to try to reduce the items retrieved from the searches. While many of the authors seem to ignore the step of database searching entirely, the most important advances were devised by authors who realized that the problem of database searching and star pattern feature extraction are inextricably linked. Yet none of them have detailed the use of an inherently multi-dimensional search, or the conditions required to make such a search technique effective.

Early in this research, it was hypothesized that a multi-dimensional search would drastically decrease the execution-time of a star-ID algorithm because it would both reduce the search time, and more importantly, reduce the filter stage, which is currently the dominant step in the star-ID process. The next chapter presents a comparative analysis of several multi-dimensional range-searching algorithms to find a candidate for the fastest search algorithm. Tailoring the feature extraction step to meet the demands of the fastest database search gives a new technique the best chance to dramatically reduce the computational complexity of star-ID.

CHAPTER III

DATABASE RANGE SEARCHING ALGORITHMS

A "range-search" algorithm finds entries in a database with a value in a given range [35-40]. In one dimension, this consists of two inequalities, min $\leq x \leq$ max. In multiple dimensions, the concept of a "range" can become much more complex considering not just rectilinear, but regions of circular, spherical or arbitrary shape. For the context of this dissertation, a multi-dimensional range-search algorithm will be one using an *orthogonal* set of ranges, each specified with a minimum and maximum in each dimension.

3.1 One-Dimensional Search Algorithms

The simplest computer-search algorithm is a "linear search" which is simply a loop over all n items in a database, checking whether each entry lies between the given range. If no additional information about the data entries is available, this may be the fastest method for obtaining the result. Such an algorithm performs in O(n)-time.

However, if the items are sorted, immediately the computer programmer is able to optimize which entries are examined. The search may begin with the item in the middle of the database. If this entry is too large, the algorithm then searches in the lower half of the space; if it is too small, the algorithm looks in the top half. The halving of the dataset continues recursively. The binary nature of the split results in the name "binary search." The run-time of such a technique is O(k lg n). Unfortunately, a minimum complexity of O(*n* lg *n*) precedes the use of the binary search, because the data set must be sorted. If the distribution of data entries is non-uniform, the binary search can usually only be surpassed by a hash table, though performance there is dependent on finding a useful hash function. Hash tables operate on the memory bits of the data directly, ignoring the direct relation to data space. The complexity of hash functions and hash tables are beyond the scope of this dissertation. Moreover "index-based" searches *can be* faster than a hash-table, especially for range-searches.

If the dataset has an approximately uniform density of entries per unit of data space, an "index" of where each data value resides in memory can be constructed. Indices of the data entries are placed in "bins" which each represent an equal amount of data-space, and an approximately constant number of data entries. While the bins are generally ordered sequentially in computer memory, they do not contain the exact same number of items, which necessitates an "index" of the location of each bin given a truly linear index, formed by a calculated offset and scale for the given data set. To obtain the index of a data item near a given value, the minimum value of the data set is subtracted; then the result is multiplied by ($n/\Delta r$), where Δr is the full range in data space. The result is the index that contains the index to the bin containing the requested value. This is the principle behind the "*k*-vector" [20-21]. Such an algorithm can perform in an amount of time independent of the number of items in the database; dependent only on the number of correct answers, O(*k*), assuming the math hardware of the computer can perform the subtraction and multiplication operations each in O(1)-time. For the *k*-vector as presented, the database must be sorted, also requiring O(*n* lg *n*)-time. In general, it is possible to create non-linear index-based data structures, but the non-linear function *must be invertible* and the evaluation time unnecessarily extends the performance of the search.

The above 1-dimensional methods have been used extensively in star-ID algorithms, despite the fact that more than one dimension of data must be matched in order to guarantee a match for any given set of observed star vectors. One might anticipate that moving to a multi-dimensional search could dramatically reduce the computational complexity of both the search and error checking stages of star-ID.

3.2 Multi-Dimensional Range-Search Algorithms

Because the binary search can be seen as a decision tree, algorithms derived from the binary search are commonly referred to as "Tree" based algorithms, such as the kd-tree, quad-tree and oct-tree [41-44]. For a range search, the kd-tree delivers promised O(k lg n)-run-time performance. Just like a binary tree in one dimension, the kd-tree divides the data set into two groups divided by a median value at each branch of the tree. The dimensions are selected cyclically, so for a 2-D data set, the root item in the tree would be the median value in the x dimension. The branches of this tree would be the median values in the y dimension in each of the halves of the data set. Tree-based algorithms are dependent on the "depth" or "height" of the tree for their performance, which is notoriously dependent on the "balancing" of the tree; i.e. the equal-partitioning of entries into the branches of the tree at each level. While the kd-tree arrangement conveniently guarantees proper balancing of the tree, it fails to achieve greater division of the data that could be obtained at each level.

3.2.1 Quad and Oct Trees

In 2 dimensions, the quad-tree divides data into four quadrants at each break of the tree. While it is not possible in practice for the selected node to be the median in both the x and y dimensions, it is possible to select the node closest to a point that does divide the space into equally dense quadrants. In other balancing schemes, the point selected is nearest the centroid of the data. In 3 dimensions, the oct-tree divides the data at each tree level into 8 octants. In general, the binary trees achieve $O(k + \log_{2d} n)$-time searches. The central problem with quad trees, and all tree-based searches, is that their performance is dependent on the height of the tree, which is dependent on the size of the database. As databases grow in size, performance decreases. Although many data structures have been proposed to solve various distribution problems, discussions of these structures ignores improvements that can be made if a data set is well distributed. For large databases that require faster searching, an algorithm that performs in less asymptotic time than a tree is desired.

3.2.1 The K-Vector ND

The K-Vector ND [45, 46], an index-based algorithm and associated data structure, can perform an orthogonal range search in $O(d + k)$-time, independent of the size of the database. The K-Vector ND has two primary data structures *sortedPointers* and *kVector* both of size *n*, along with three supporting arrays of dimension *d*. The *sortedPointers* array contains a list of pointers to the data entries the user wishes to retrieve (or a list of indices in an array). The entries are grouped by their bin number and placed in the *sortedPointers* array in order of increasing bin number. Since the entries contain different data values in each dimension, a desire to sort the entries further becomes a moot concept, so entries with identical bin numbers are not sorted further.

The *k*-vector provides a mapping between bin numbers and a location in the *sortedPointers* array. A bin number acts as an index into the *k*-vector, which yields either a pointer or an index into the *sortedPointers* array to the first item in the corresponding bin.

3.2.2 The Bin Function

In order to calculate the bin number of a particular data value, three other pieces of information are required: 1) the minimum value on each axis, 2) the number of bins per unit in data space, and 3) the number of bins along each dimension. These are each arrays of length *d*. The minimum value on each axis is referred to as the “offset” and is an array of real values. The number of bins per unit in data space is referred to as the “scale” and is an array of real values. The number of bins in each dimension is called “divisions” and is an array of integers. The bin function, Eq. (1) uses these very simple arrays to compute the bin number of each data entry.

$$binFunction(values[d]) = \sum_{i=1}^{d} \lfloor scale[i](values[i] - offset[i]) \rfloor \prod_{j=1}^{i} divisions[j] \qquad (1)$$

Since the product of divisions remains identical through the lifetime of the K-Vector ND, it is far more efficient to compute these products once and store them in another array. Therefore "products" is defined to be an array of length $d + 1$, with $products[1] = 1$ and $products[i] = divisions[i]*products[i - 1]$ for $i > 1$. The bin function is modified to fit Eq. (2).

$$binFunction(values[d]) = \sum_{i=1}^{d} \lfloor scale[i](values[i] - offset[i]) \rfloor products[i] \tag{2}$$

In the pseudo-code given in Table 2, the values for *d*, *products*, *scales*, and *offsets* are taken to be properties of an implicit class instance KVectorND.

The bin function loops *d* times as it adds a value along each dimension. Thus, the bin function completes in $\Theta(d)$-time. It uses only two data values as temporary storage, one for the accumulating bin value, one for the iterator.

Table 2 Bin Function Psuedo Code

```
int BinFunction(real values[d])
    int i
    int bin = 0
    for (i = 1, i ≤ d, i++)
        bin += products[i] * floor(scales[i] * (values[i] – offsets[i]))
    return bin
```

3.2.3 Building the *kVector*

For the average case in which $N = \lceil \sqrt[d]{n} \rceil^d$, the *kVector* is built in three stages. In the first stage, the items to be included in the *kVector* have their bin numbers computed and cached, through an iterative application of the "insertItem" function, listed in Table 3. The number of items in each bin is tallied. The remainder of the steps are listed in

Table 4. Second, the number of items in each bin is iteratively summed. The array of summations is the *kVector*. Third, pointers to items are placed in the *sortedPointers* array at the appropriate location. In the following pseudo-code, cachedBins, cachedIndexes, and inserted are taken to be defined in the appropriate scope.

Table 3 Insert Item Psuedo Code

```
Insert_Item(int item, real values[d])
cachedPointers[inserted] = item
cachedBins[inserted] = BinFunction(values)
inserted++;
```

Table 4 Finalize Psuedo Code

```
Finalize
int N = products[d]
int kVector[N+1] = 0
int i, bin
for(i=1, i≤inserted, i++)
   kVector[cachedBins[i]]++
int sum = 0, temp
int temp
for(i=1, i ≤N, i++
   temp = kVector[i]
   kVector[i] = sum
   sum += temp
kVector[N+1] = sum
for(i=1, i ≤ inserted, i++)
   bin = cachedBins[i]
   sortedIndexes[kVector[bin] + positions[bin]] = cachedIndexes[i]
   positions[bin]++
```

There are three loops. The first loop performs the bin function on each of n items, for a total time of $\Theta(d\ n)$. It creates a list of cachedPointers of size n and a list of *cachedBin* values, of size n; for a total data storage of size $2(1 + n)$ integers. The second loop creates a new array of size N integers and populates it, which takes $\Theta(N)$ time.

Since $N \approx n$, $N < dn$, $\forall\ d > 1$, then the $d\ n$ term remains the dominant term. The final loop occurs n times and takes constant time on each iteration, a time of $\Theta(n)$. The total run time for the build-time is $\Theta(d\ n)$.

3.2.4 K-Vector-ND Range Search

The range search algorithm has three stages. The first stage computes the bins that comprise the boundaries of the search space in each dimension. There is one starting bin, and d ending bins. Although it would appear that computing d bins each in $\Theta(d)$ time would take $\Theta(d^2)$, only one dimension value changes between each *endBin* value, meaning that the use of memoization [5] can reduce the computation of the d values to $\Theta(d)$ time. This optimization and the use of uniform scales across multiple rows of data are the primary difference between a traditional index-based search and the K-Vector-ND. The optimization simultaneously increases the K-Vector-ND's sensitivity to non-uniform data, as it cannot adapt the scales for each row of data.

The second stage copies pointers or indices from the *sortedPointers*. It is repeated for each row in the search range. While actual star-tracker implementation of this algorithm would probably use a pre-allocated array in which to place results, the frequent changes in k from rapidly altering test data sets necessitated a more robust solution to avoid buffer over-run. Thus, doubly-linked-list structures were used in the C source-code.

The third stage advances the bin boundaries as the search proceeds from one row to another. The second and third stages are then repeated until the search range has been covered. The code is listed in Table 5.

Table 5 Psuedo Code for K-Vector ND Range Search

```
Range Search(real mins[d], real maxs[d])
int minBins[d], maxBins[d], endBins[d+1], increments[d], i
endBins[1] = 0
for(i=1, i≤d, i++)
   minBins[i] = floor(scales[i] * (mins[i] – offsets[i]))
   maxBins[i] = floor(scales[i] * (maxs[i] – offsets[i]))
   endBins[0] += minBins[i] * products[i]
for(i=1, i≤d, i++)
   endBins[i+1]  = endBins[i] + ((maxBins[i] – minBins[i]) * products[i])
   if(i== 2)
      endBins[2] += 1
   increments[i] = products[i+1] – (endBins[i+1] – endBins[i]) – products[i]
int results[], beginIndex, endIndex
do
   beginIndex = kVector[endBins[2] – difference]
   endIndex = kVector[endBins[2]]
   for (i=beginIndex, i < endIndex, i++)
      push(results, sortedIndexes[i])
   i = 1
   do
      endBins[i] += increments[i]
      endBins[i+1] += products[i+1]
      i++
      if (i == d)
         return results
   while(endBins[i] > endBins[i+1])
```

The range search has three loops; the last loop contains two interior loops. The first loop runs *d* times, as does the second. The third loop runs until the search has completed. The first interior loop has its internals executed for each search result, in other words *k* times. The second loop runs at most *k* times, and most of the time it executes once. Sometimes it executes more often, up to *d* times. Since the effect of the loop is to increment the search through the dimensions of the bins, it will execute a finite number of times. If the search range were one bin, it would fire d times. If the search range is a column of bins, the loop fires up to k times. Thus the total number of times it fires cannot be more than $k + d$. The dominant terms are k and d, and the final running time is $O(k + d)$.

In the worst search case, where there are bins searched which contain no entries, the algorithm essentially "spins its wheels." Its performance cannot be written in terms of k since the bins being searched do not include items. Instead, the performance must be written in terms of the number of bins, N, or since $N \approx n$, in terms of n. Note that the time to search an empty row is $O(1)$ since the third loop's first interior loop will not execute. Thus the worst-case time is based on the number of empty rows searched. This can be approximated by using variables Δx for the range of the data in the database and Δr for the search ranges. The expression $\frac{\Delta r}{\Delta x} n^{\frac{d-1}{d}}$ better suits the performance of the K-Vector ND when the k-vector is empty. It is for this reason that to achieve the quoted performance, the database must be well-distributed so that the overall assumption that $N \approx n$ is valid for all search-range-sized subsets of the database.

3.2.5 Delaunay Triangulation

Another data structure used in multi-dimensional searches is the Delaunay Triangulation [47]. The Delaunay triangulation is especially useful in applications involving physical data; i.e. data entries for which a distance metric can be applied. In the case of 2-D data, the distance is simply the Euclidean distance. For a set of unit vectors on the surface of a sphere, one metric is the inverse cosine of the dot product of any two vectors (i.e., the angle). For any given triangulation, the set of data points are connected to each other by line segment edges such that all points are reachable from the graph, no adjacent three edges (which form a triangle) contain an interior point, and no line edges intersect, except at points. As one might expect, there are frequently many different valid triangulations for any given point set.

Delaunay imposed an additional criterion: a circle circumscribed about a triangle may not contain any points in its interior. Delaunay's criterion causes the triangulation to become unique: the Delaunay triangulation. This particular unique triangulation has the property that the nearest neighbor to any point is always connected by an edge. Combined with a physical concept of distance, it also has the advantage of a very fast "nearest neighbor search." Starting from any point in the graph, a step in the nearest neighbor algorithm is to examine the neighbors connected to the initial point for their distance from a target point. The algorithm then selects the neighbor nearest to the desired target point and iterates until no neighbors are closer than the selected point. Delaunay triangulations also have a relatively fast build-time, about O(n lg n).

The drawback to the Delaunay nearest neighbor algorithm in general is that the execution time depends on the number of triangles between the starting point and the target point, which are potentially on the opposite side of the unit sphere from each other! However, if a starting point near the target point is already known, for instance, from star-ID, then the algorithm has many fewer triangles to run through.

3.3 Comparative Performance of 2D Range Searching Algorithms

To confirm the analysis of the algorithms, the performance of the K-Vector ND is measured and compared to the performance of a quad-tree over seven database sizes and 10,000 search ranges. The quad-trees are balanced by using the node closest to the centroid of a given set as the root of that set.

The test is performed in Mac OS X running an Intel Core 2 Duo at a clock speed of 2.4 GHz. The unix system call "getrusage()" reports both the "user" process time as well as the "system" process time in increments of microseconds without including the time used by other threads. Each algorithm is directed to perform each search 50 times, and the total time is averaged over the 50 searches. Although the 10,000 search ranges are generated at random, there is a skew towards smaller search ranges since the minimum and maximum bounds are randomly generated separately. The root database of visible stars is filtered by star brightness to generate 7 databases of different sizes, each with identical data ranges. The same search ranges are applied to each database.

To improve performance of both algorithms, a "queue" is implemented to avoid repeatedly allocating and deallocating the small data structures used in the linked lists. When a linked list node is no longer needed, it is placed in a queue to wait for one to be needed. If there are no items in the queue, a new node is created. The Quad-Tree algorithm frequently merges lists as well, so list structures are also queued. It is assumed that a similar technique would be used in the final implementation of these algorithms; in many situations these search algorithms are called at a high frequency.

The database entries, shown in Figure 10, include up to 7,122 visible stars using

Geocentric declination and right ascension as the data values. Seven sub-databases of varying size are selected based on the brightness of the stars.

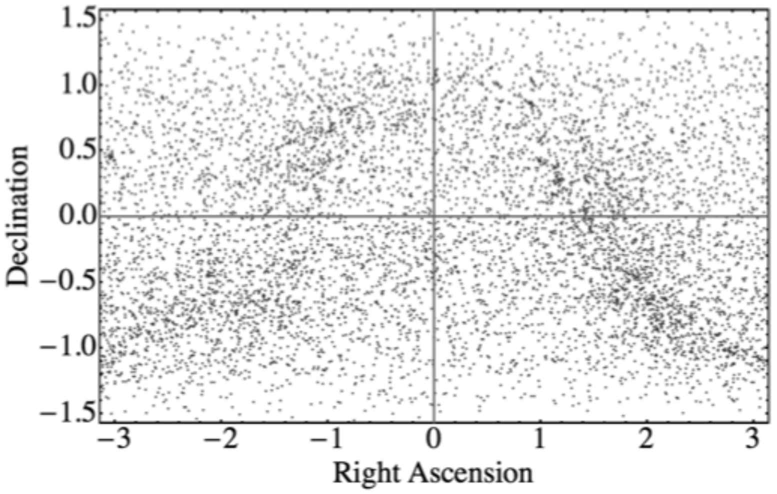

Figure 10 Database of stars for quad-tree k-vector-2D test

To best characterize the results, the ratio of execution times of the Quad-Tree to the K-Vector ND are presented. For values above dotted line, the K-Vector ND performs faster by the factor indicated on the vertical axis. For values below the dotted line, the Quad-Tree performs faster. A box-whisker plot is generated for each set of search results with matching database size, *n*, and the number of found results, *k*. The box contains half of the results, while the whiskers each contain a quartile, and a black line in the box marks the median.

A typical test result set for $k = 2$ is presented in Figure 11, which demonstrates that the advantage of using the K-Vector ND increases with increasing database size. It is clear from all the plots in Figure 12, that the Quad-Tree only performs faster in fewer than one quarter of the search cases.

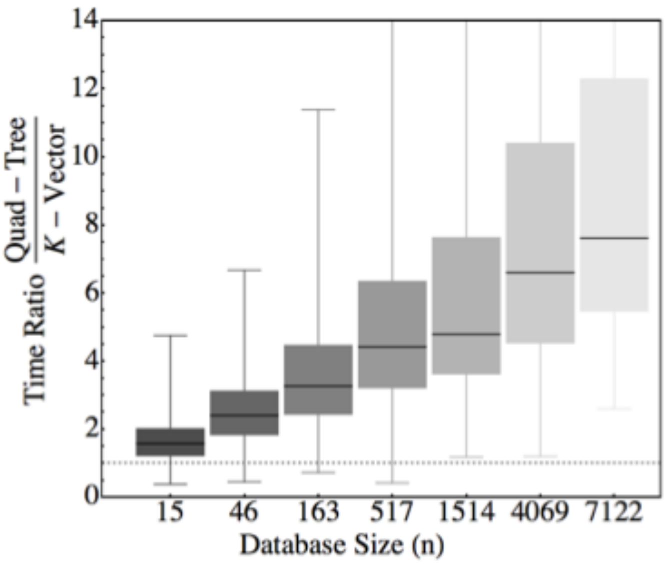

Figure 11 Typical test results for k=2

The graphs clearly show that the performance advantage of the K-Vector ND increases as the size of the database increases. This is due to the fact that the performance of the quad-tree decays by $\log_4 n$ as the database increases. This same logarithmic form is shown in the above plot, reflecting the nature of the quad-tree search performance. Note that even for a database as small as 15 entries, the K-Vector ND still has an advantage over the quad-tree. The size and distribution of the search ranges is not important because the search results have been collected by the number of found results, *k*.

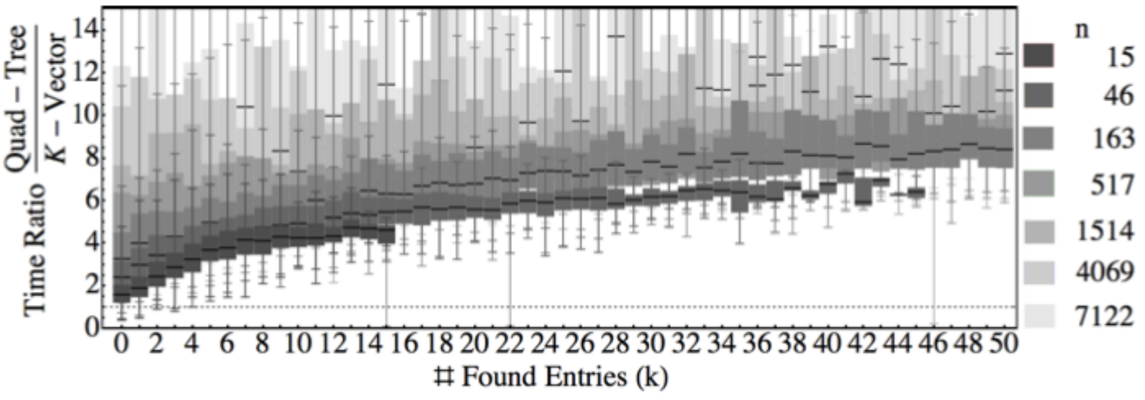

Figure 12 All test results k ≤ 50

There are outliers in the data, which may be related to the need of both algorithms to allocate linked list nodes when they reach a number of found items that is larger than previous searches.

3.3.1 Well Distributed

The K-Vector ND described herein performs range search very fast: O(d + k) for a well-behaved data set. This data set is constant, bounded, and data values are evenly distributed throughout the possible data ranges. Under these conditions, the K-Vector ND is found to perform linearly with the number of found items and independently of the size of the database. The performance is also found to be much faster than comparative tree based algorithms. Furthermore, the quad tree is specifically geared towards a 2-dimensional search, while the K-Vector ND code used in this test is still able to handle a data set with any number of dimensions, and could actually increase its performance by being refined for only 2 dimensions.

Unfortunately, the K-Vector ND only maintains these performance measures when the data is well-distributed. The experimental approach used in this paper uses a database with a distribution as poor as inverse sine, which is shown to outperform a quad-tree on average. The best case for the K-Vector ND would be a data set that has 1) no regions in which data cannot lie, and 2) approximately uniform distribution of entries.

The K-Vector ND also assumes that the database is static. While a quad-tree may have entries inserted at a logarithmic cost, the K-Vector ND unfortunately requires $O(d\ n)$ to be rebuilt if an entry is added. For many applications, this is acceptable because the database does not change often.

Despite the restrictions placed on the database, the K-Vector ND still gives an edge to many applications that can fit within the noted constraints.

3.4 Comparative Performance of 3D Range Searching Algorithms

To test these algorithms, the Hipparccos star catalog is filtered into subcatalogs each containing the stars brighter than a particular visual magnitude limit, ranging from 3.0 to 9.0 in increments of 0.5. Because a real search application would generally look for stars that should be listed, the test consisted of performing a range search centered around each star in the database. All algorithms are written in C, and compiled with GCC 4.2, with optimization set to "-O3", the most optimized setting.

3.4.1 Test: Oct-Tree vs. K-Vector ND

The first test compares the performance of the oct-tree with the K-Vector ND.

Each is given the same database and search ranges. The performance is measured with the UNIX system call “getrusage()” which measures the amount of a time the current process spends in the processor, measuring all operations of the process, and no operations of other threads. The time is measured in µs, which is about 2530 clock-cycles on the hardware. In order to increase the precision of the measurment since the K-Vector-ND typically completed in less than a single µs, the search is repeated 100 times, and the search time divided by 100. For the oct-tree, however, the significantly slower execution time caused the repetitions became a nuisance, so the number of iterations was reduced to 10.

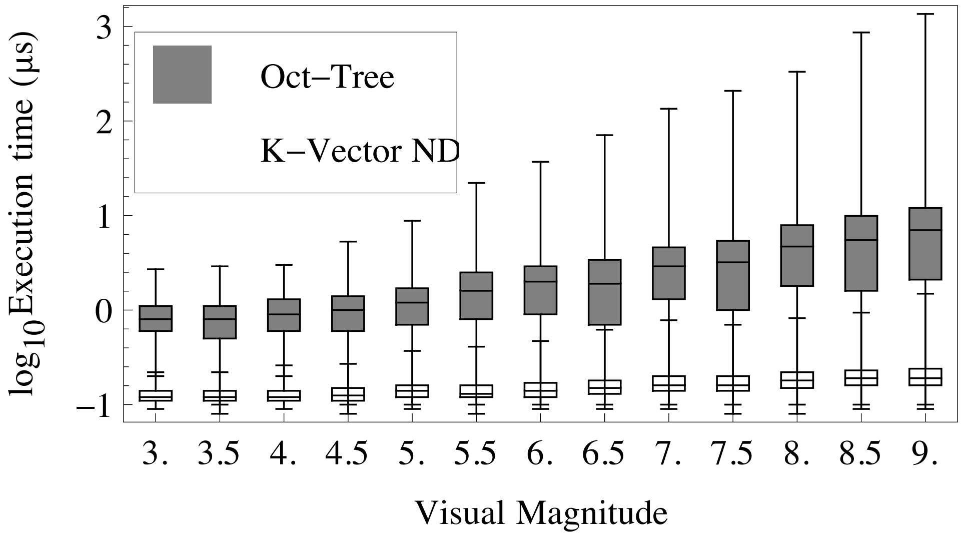

Figure 13 Relative performance of oct-tree and K-Vector ND

In Figure 13, the gray boxes represent the oct-tree performance while the white boxes represent the K-Vector-ND performance. It is clear from the plot that the oct-tree performs significantly more slowly than the K-Vector-ND across the range of catalog sizes; in general, about one order of magnitude slower. One may safely supplant the spherical polygon search algorithm with the K-Vector ND, using the same range formulas in Samaan's paper.

The plot also clearly shows an increase in the execution-time of the K-Vector-ND as the database size increases. This result is precisely what one expects: as the number of data points increases, the number of bins increases, but most new bins remain empty, as the number of bins on the sphere's surface containing all data points grows with the 2/3rds power of the number of data entries, causing the number of entries in each non-empty bin to increase, resulting in an increase in the execution time.

3.4.2 Test: Delaunay Triangulation Nearest Neighbor vs. K-Vector ND

The concept for the procedure is this: using the known identities of a few stars in an image and the estimated attitude therefrom, the identities of unidentified stars are sought by searching the catalog for a vector closest to the calculated inertial direction of an observed star. For the K-Vector ND, this means formulating a spherical-polygon-search type bounds—an identical test to the one already performed. For the Delaunay triangulation, this means seeding a nearest neighbor search with the identity of one of the stars in the image. For the Delaunay case, it is guaranteed that the starting star is within the FOV from the target point, and that the starting star is not the target star. To simulate this condition, a random transformation is generated which rotates the target star by no more than the size of a general FOV. The nearest point already in the triangulation to this random point is selected as the starting point, provided it is not the target point. The rotation direction and angle are generated separately. It should be noted that the rotation angle is uniformly random, meaning there is a bias towards selecting closer starting stars compared to the relationship between the number of stars and their inter-star angles.

The same measurement techniques as described above are used for this test. The times for the Delaunay nearest-neighbor search compared with the same K-Vector-ND search results repeated are shown in Figure 14.

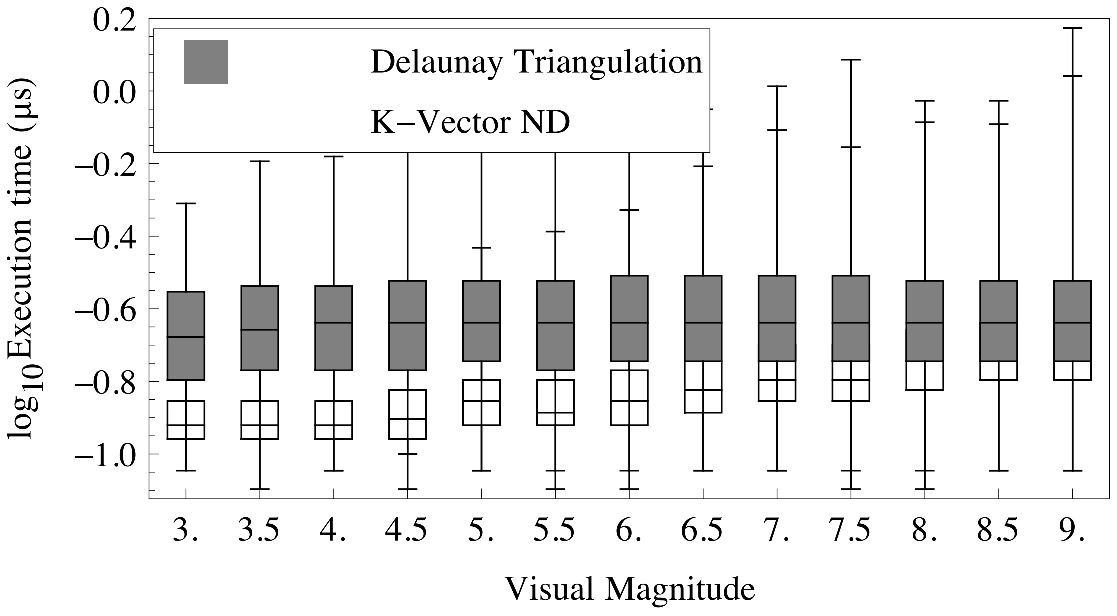

Figure 14 K-Vector ND vs. Delaunay search

It is clear from the plot that the Delaunay search rivals the K-Vector ND, especially for the larger database sizes. I suspect that when optimized, the Delaunay triangulation could run as much as a factor of 2 or 3 faster than the implementation tested herein, perhaps pushing it as much faster than the K-Vector ND as the K-Vector ND is faster in the above plot. When the execution-times of two algorithms are as close as they are in this analysis, the memory access characteristics & floating-point operations of a target platform can make the real difference as to which algorithm is faster.

It is informative to plot the performance of the Delaunay nearest-neighbor search in terms of the original random angle from the target point, shown in Figure 15.

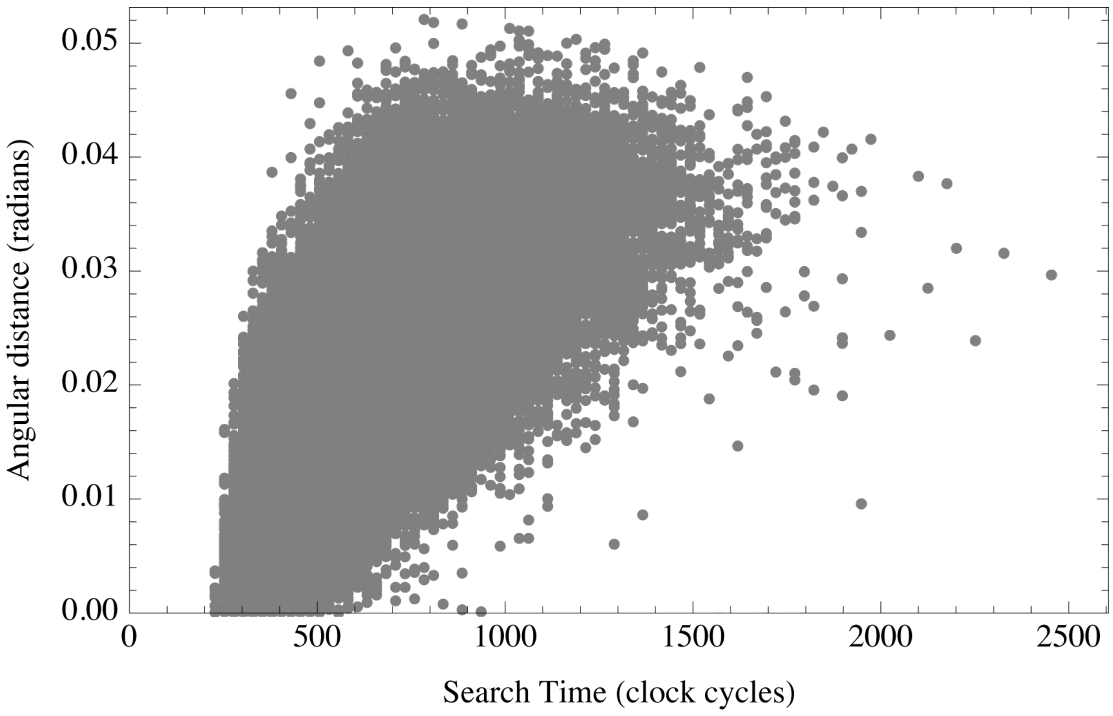

Figure 15 Delaunay angular separation vs. search time

The plot suggests the dominant factor in the Delaunay search-time is not the total angular distance from the starting point to the target point, but the number of triangles from the known starting point to the unknown point, as is expected for that algorithm. In cases where a known nearby point is not available, the search times are actually significantly higher, and approach the limit of the tree-based algorithms.

3.4.3 Test: Delaunay Triangulation Range Search vs. K-Vector ND

Unfortunately for the developer, the Delaunay nearest-neighbor algorithm is not a range search algorithm, and the need to determine if there are two stars near the target point dictates the use of a range-search-type algorithm. The nearest-neighbor algorithm can seed the range search algorithm, however. The algorithm would then consider neighbor points that are within the specified range, ignoring points that are outside the search range.

Unfortunately, again, the need to not consider points that have already been searched means an additional data structure is needed. A simple RAM-machine model, in which the nodes themselves are marked, which takes constant time to perform, check, and linear time to erase is generally not suitable for a star tracker processor, as the database may be stored in slow, energy-demanding, read-only, or extremely limited memory. The comparative results of the Delaunay range-search algorithm, seeded by the nearest neighbor algorithm, are presented in Figure 16.

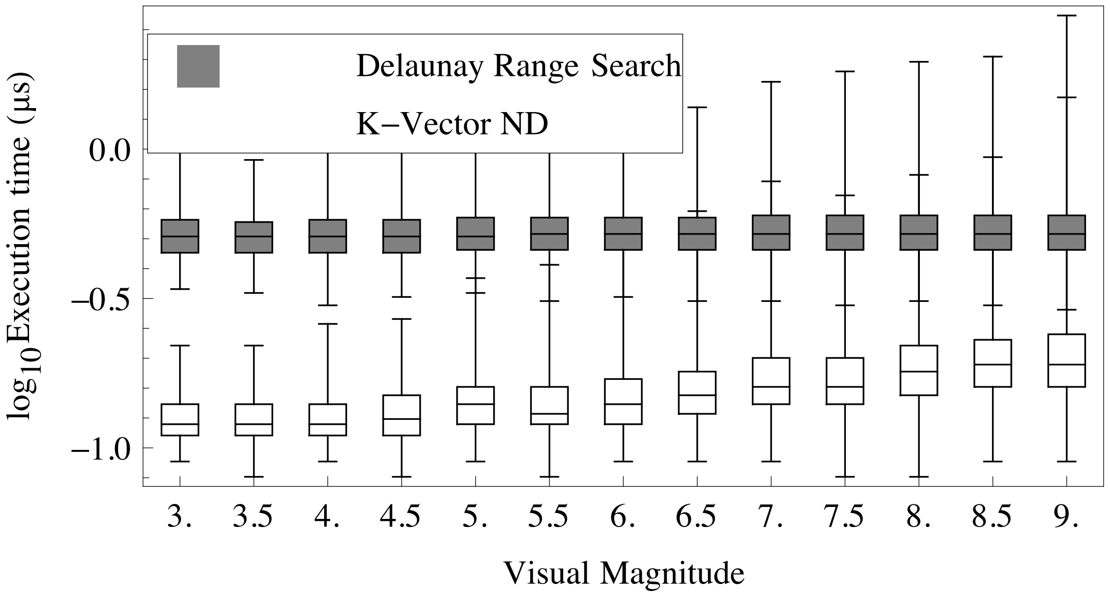

Figure 16 Delaunay range search vs. K-Vector ND time

The Delaunay Range-Search algorithm, which is seeded by the identical nearest-neighbor algorithm as above, is shown in gray, while the K-Vector-ND remains in white. Identical ranges are input to both algorithms. The results show the K-Vector-ND performs significantly faster across the range of tests. Note that a factor-of-three improvement from an optimized implementation of the nearest-neighbor search would correspond to a $\log_{10}$ of 0.5, which generally does not result in the Delaunay range-search performing faster. However, it is also clear from the figure that the Delaunay range search maintains its search time, while the K-Vector ND tends to increase in size with increasing catalog size. Again, this is due to the poor distribution of data throughout the volume of possible values. Because the Delaunay triangulation does not consider the unused volume of the sphere, its algorithm will eventually out-perform the

K-Vector ND if the database is allowed to grow large enough. However, for star catalogs with up to 80,000 entries (sufficient for current and anticipated spacecraft), the K-Vector ND remains faster.

3.5 Build-Times

All of the attention up to this point has been on the execution-time of the run-time algorithm, arguably the most frequently used component in the star-ID process, but some regard to the build-time of the databases that are searched should be given. The K-Vector-ND, as described previously, can be built in only O(d n)-time, while both the oct-tree and Delaunay triangulation require O(n lg n)-time. For the tests above, the build-time of each database was measured using the same technique, though repeating to increase measurement precision was not performed. The results are ploted in .

clearly shows the K-Vector-ND build-time is a few orders of magnitude faster than either the oct-tree or the Delaunay Triangulation. Through careful optimization, the build times for the oct-tree and Delaunay Triangulation could possibly be optimized by as much as an order of magnitude. The K-Vector-ND has the draw back that, as implemented, it is an immutable data structure, meaning that when a data entry is added to the already-built database, it must be entirely rebuilt. Though a mutable K-Vector ND exists, it was not tested for this dissertation.

Finally, in terms of memory size, the oct-tree uses eight pointers per data entry whether in canonical form or mutable form, the Delaunay triangulation can be optimized to use an average of 6 pointers per data entry (if discarding the triangle data that allows quick insertions), while the K-Vector uses about 2 pointers per data entry and 9 floating point values. Samaan's original Star-Neighborhood algorithm used 9 pointers per entry. So the K-Vector-ND uses between 3 to 4 times less memory than the other algorithms.

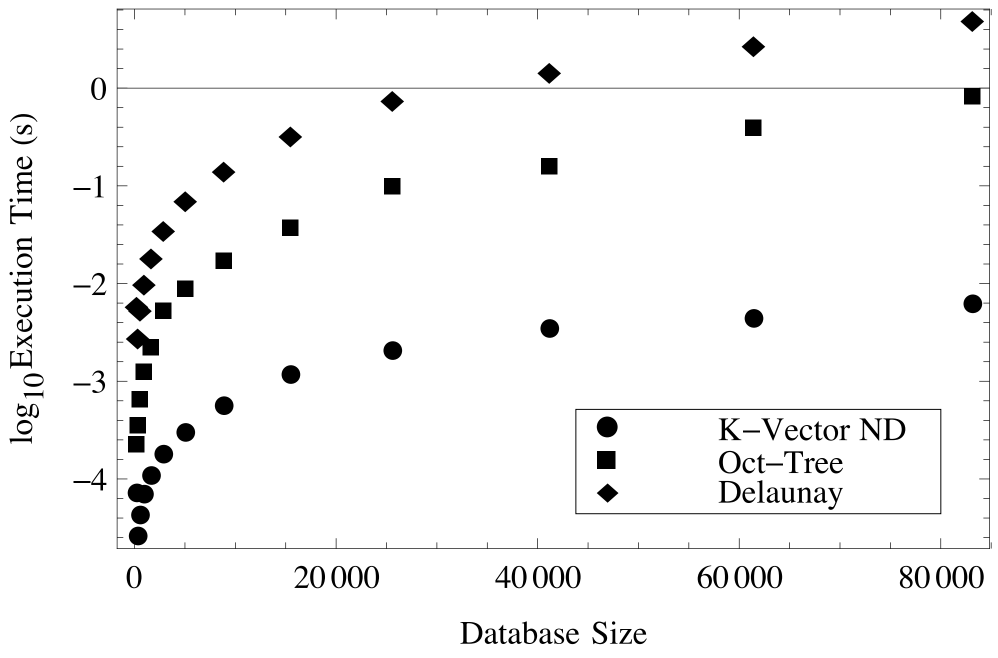

Figure 17 Search auxilliary data structures build-times

3.6 A Candidate Multi-Dimensional Search Algorithm

Given the results of the above tests, the K-Vector ND is able to consistently out-perform the other algorithms under the following condition: the data set must be nearly-uniformly distributed. Perhaps defining a set of star pattern parameters that are well distributed would create the fastest star-ID algorithm. Chapter IV introduces such a set of star pattern parameters.

CHAPTER IV

NEW STAR PATTERN PARAMETERS

4.1 Star Pattern Parameter Functions

Given all background stars with brightness above a selected threshold, there is a star pattern for each star. That star pattern consists of a star ($\hat{s}_0$) and its nearest $p-1$ neighboring stars ($\hat{s}_1, \hat{s}_2, \cdots, \hat{s}_{p-2}, \hat{s}_{p-1}$), sorted by increasing angular distance from $\hat{s}_0$. These stars may be thought of as directions to stars represented by unit-vectors on a unit-sphere. This choice for a star pattern is an extension of Liebe's pattern. Note that the selection of the nearest $p-1$ stars is the "wrong answer" itself, and will introduce a number of issues for the star-ID process to overcome. These issues will be mitigated by inserting additional "wrong answers" into the database.

The equations for the new star-pattern parameters [48] are:

$$P_a = \frac{1 - \hat{s}_i \cdot \hat{s}_{i+1}}{2 - \hat{s}_0 \cdot \hat{s}_i - \hat{s}_0 \cdot \hat{s}_{i+1}}, \quad 0 < i < p-1 \tag{3}$$

$$P_b = \frac{\hat{s}_0 \cdot \hat{s}_i - \hat{s}_0 \cdot \hat{s}_{i-1}}{\hat{s}_0 \cdot \hat{s}_{i-1} - \hat{s}_0 \cdot \hat{s}_{i+1}}, \quad 1 < i < p-1 \tag{4}$$

First, it should be noted that Eq. (3) creates star patterns for $p > 2$, and it generates two parameters for a pattern of 4 stars. While Eq. (4) creates star patterns for $p > 3$, and it generates a third parameter for a pattern of 4 stars. The star structures involved by these two parameters are shown in Figure 18 and Figure 19 respectively.

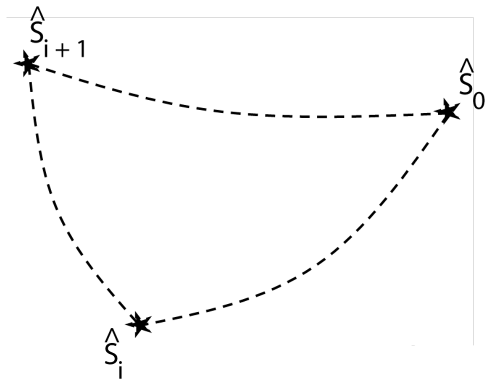

Figure 18 Example of star pattern for P_a

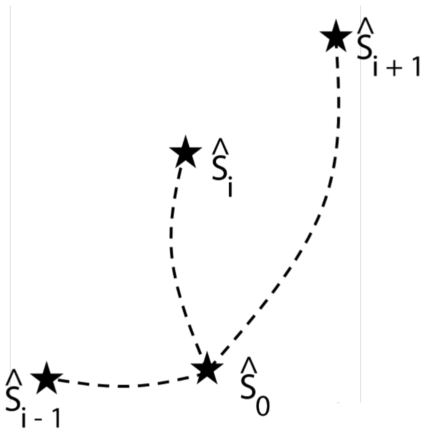

Figure 19 Example of star pattern for P_b

4.2 Tautological Bounds

Second, these equations are both tautologically bounded. For the first parameter (P_a), knowing $\hat{s}_0 \cdot \hat{s}_i \geq \hat{s}_0 \cdot \hat{s}_{i+1}$ implies $1 \geq \hat{s}_i \cdot \hat{s}_{i+1} \geq \cos\left(\cos^{-1}\left(\hat{s}_0 \cdot \hat{s}_i\right) + \cos^{-1}\left(\hat{s}_0 \cdot \hat{s}_{i+1}\right)\right)$. Both the minimum and maximum cases occur when $\hat{s}_0 \cdot \hat{s}_i = \hat{s}_0 \cdot \hat{s}_{i+1}$. For the minimum case, in which the numerator's dot product may approach 1, the pattern parameter approaches 0. For the maximum case, $\cos^{-1}\left(\hat{s}_i \cdot \hat{s}_{i+1}\right) = 2\cos^{-1}\left(\hat{s}_0 \cdot \hat{s}_i\right)$, and the resulting maximum parameter value is 2. While it is not practically possible for the minimum case to be achieved, since $\hat{s}_i$ and $\hat{s}_{i+1}$ would need to be in the same location, it is possible for the parameter to approach the minimum value. For the second parameter (P_b), $\hat{s}_0 \cdot \hat{s}_{i-1} \geq \hat{s}_0 \cdot \hat{s}_i \geq \hat{s}_0 \cdot \hat{s}_{i+1}$, which implies $\hat{s}_0 \cdot \hat{s}_{i+1} - \hat{s}_0 \cdot \hat{s}_{i-1} \leq \hat{s}_0 \cdot \hat{s}_i - \hat{s}_0 \cdot \hat{s}_{i-1} \leq 0$, so the bounds are 0 and 1. However, it is numerically possible for both of these bounds to be reached.

4.3 Well-Distributed

Furthermore, and perhaps most importantly, these new parameters are well-distributed between the bounds. The true measure of how well the K-Vector ND performs depends on its discretization of the data range into equally-sized and equally-spaced "bins." The K-Vector ND attempts to have the same number of bins as it has entries, making the average number of entries per bin 1. If the number of items in each bin is actually 1, then the K-Vector ND will search in O(d + k)-time, as advertised. However, if the number of items deviates significantly from 1, the search analysis will no longer be applicable, and the time for the search will increase.

In order to evaluate the distribution of entries into the bins, the number of items in each bin is counted for various database sizes, formed by varying the maximum visual magnitude (M_V) in each database from 3 to 9, for a $p = 4$ star pattern. Then the ratio of the number of bins containing a specific number of items to the number of total bins in the database is plotted against the number of items per bin, for each database size, in Figure 20.

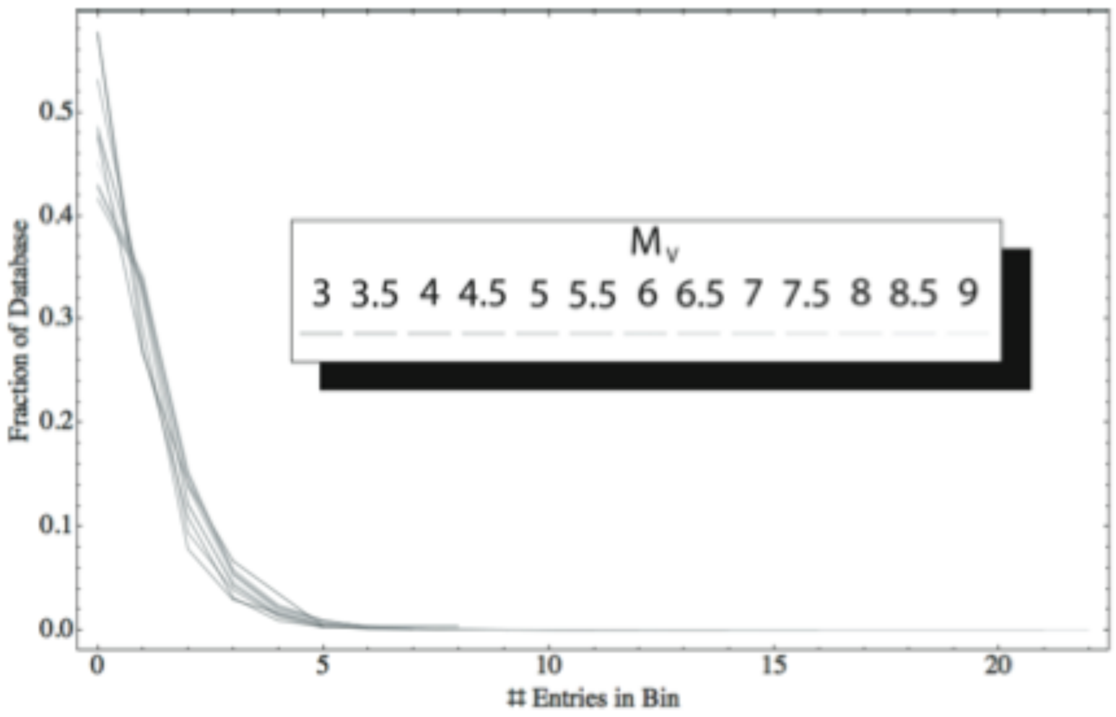

Figure 20 Distribution of star pattern parameters

Several properties of the distribution in Figure 20 are clear.

1. The distribution of entries remains well-distributed through all database sizes.

2. The maximum number of items in any one bin increases as the number of items in the database increases.

3. The fraction of bins containing no entries decreases as the number of database items increases.

Overall, the results show that the database remains well-distributed over many orders of visual magnitude. The larger databases might benefit from a $p = 5$ star pattern, to further separate the "clumping" of star patterns.

4.4 Error Bounds Formulation

The angular measurements made in a star camera unfortunately contain uncertainty, and thus the star-pattern parameters of the observed star patterns are not known exactly. To find the entry in the database with the corresponding star pattern parameters, a range of possible values for each parameter must be formulated at run time; the ranges are then fed into the multi-dimensional range search. The following are three approaches to estimate the error bounds with first-order techniques.

4.4.1 Principal Error Components

To understand what error bound formulations are "accurate," a graphical understanding of the pattern parameter is desired. To generalize the plot, consider that the sky field is rotated such that the selected central star is aligned with the *z*-axis, its closest star is aligned with the *x*-axis, and the remaining stars are plotted in the *x* and *y* axes. The resulting star pattern parameter may be read from the contour plot in Figure 21 by the position of $\hat{s}_{i+1}$.

Note that as the angle to the closest star changes, the contours in the plot will alter slightly. The large white circle at the center is an area where the second closest star may not be, since its position in this circle would make it the closest star. The smallest black circles surrounding the stars are "error bounds" of the central and closest star. In general, the relative size of these error bounds is much smaller than shown here. A star camera will blur star light over 3 to 5 pixels and achieve a centroiding accuracy of 1/20th of a pixel. In these cases, the stars will be at least 60-120 times further apart than the size of their error bounds.

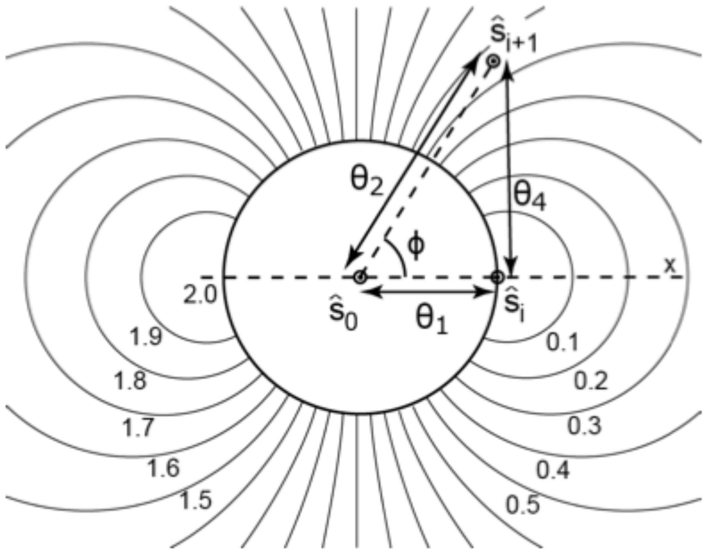

Figure 21 Star pattern parameter graphical interpretation

By using a plot like this, one could read minimum and maximum values for a particular star pattern. The error bounds, however, need to be transferred from the central and closest star to the second closest star. While such an action would not produce the exact error bounds for the parameter, it would be in keeping with the spirit of a first-order analysis.

Once the error has been transferred from the central and closest star to the second closest star, the measurements that could change the parameter value are only the angular separation of the second closest star with respect to the closest star and central star, while error in the separation of the central and closest star does not contribute directly to the error bounds (that particular error is added to the error in the location of the second closest star).

Instead of actually walking the enlarged perimeter of the second closest star's error bounds, the minimum and maximum values can be calculated by taking derivatives with respect to the principle components in a cylindrical coordinate system. Unfortunately, the angular component, φ (the angle from $\hat{s}_0$ between $\hat{s}_i$ and $\hat{s}_{i+1}$), is a value not directly measured in the available dot products. Thus, that angle, or rather its sine and cosine, must be calculated from the available measurements with Eq. (5).

$$\cos\phi = \frac{\cos\theta_1 - \cos\theta_2\cos\theta_4}{\sin\theta_2\sin\theta_4} \tag{5}$$

and its effect on the parameter derivative included, in Eq. (6):

$$\Delta P_a = 3\sigma\sqrt{\left(\frac{\partial P_a}{\partial\theta_4}\sin\phi\right)^2 + \left(\frac{\partial P_a}{\partial\theta_4}\cos\phi + \frac{\partial P_a}{\partial\theta_2}\right)^2} \tag{6}$$

where $\hat{s}_0 \cdot \hat{s}_i = \cos\theta_1$, $\hat{s}_0 \cdot \hat{s}_2 = \cos\theta_2$, $\hat{s}_1 \cdot \hat{s}_2 = \cos\theta_4$, and σ is the allowed error on each angle.

The parameter derivative, based on the closest and second closest angles, is plotted in Figure 22. Immediately, the unusually large derivative surrounding the origin becomes apparent.

As a first order approximation, the principle derivative component method cannot be beat in terms of accuracy. However, it is computationally complex, and a faster method would be more convenient.

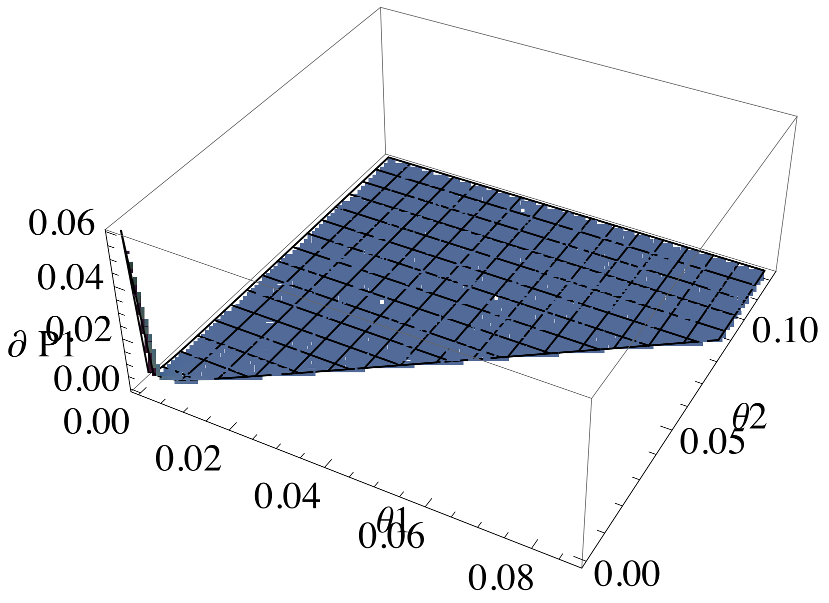

Figure 22 Principle error components

4.4.2 Partial First Derivatives

Perhaps the first tact in finding the error bounds of an equation is to take the first derivative with respect to the variables involved. Then, a first order approximation of the error may be found by simply multiplying each derivative by the uncertainty of the respective variable. Such a solution for the first parameter can be calculated by Eq. (7):

$$\Delta P_a = \sigma \frac{\sin\theta_4 - P_a(\sin\theta_1 + \sin\theta_2)}{2 - \cos\theta_1 - \cos\theta_2} \tag{7}$$

The trouble with this formulation is that it becomes zero under certain circumstances. Consider that $0 < \theta_4 \leq \theta_1 + \theta_2$, and by introducing the arbitrary scaling factor, $0 < h \leq 1$, then $\theta_4 = h(\theta_1 + \theta_2)$. The parameter derivative would become zero when $\sin\left(h\left(\theta_1+\theta_2\right)\right)=P_a\left(\sin\theta_1+\sin\theta_2\right)$. Using the small angle assumption, this is

$$h\left(\theta_1+\theta_2\right)=P_a\left(\theta_1+\theta_2\right) \qquad \text{or} \qquad h=P_a.$$

In practice, this equivalence occurs around $P_a = 0.5$, which is clearly incompatible with the graphical interpretation. Thus, a sum of partial derivatives is inadequate for calculating practical error bounds for the star pattern parameters. Another method is required.

4.4.3 Individual Mins & Maxs

A third approach for estimating the error bounds is to form two equations, each with the minimum and maximum possible values for the measurements inserted into the appropriate places in the equation to cause the parameter values to be minimum and maximum; Eq. (8).

$$\begin{cases} \min\left(P_a\right)=\dfrac{1-\left(\cos\theta_4+\varepsilon_4\right)}{2-\left(\cos\theta_1-\varepsilon_1\right)-\left(\cos\theta_2-\varepsilon_2\right)} \\ \max\left(P_a\right)=\dfrac{1-\left(\cos\theta_4-\varepsilon_4\right)}{2-\left(\cos\theta_1+\varepsilon_1\right)-\left(\cos\theta_2+\varepsilon_2\right)} \end{cases} \tag{8}$$

where $\varepsilon_k=\Delta\theta_k\sqrt{1-\cos^2\theta_k}$. The immediate concern is whether such a calculation would produce accurate error bounds, and second is whether those error bounds are bounded. Certainly, these values produce unbounded errors, and the minimum and maximum must sometimes be forced to lie within the (0, 2] tautological bounds.

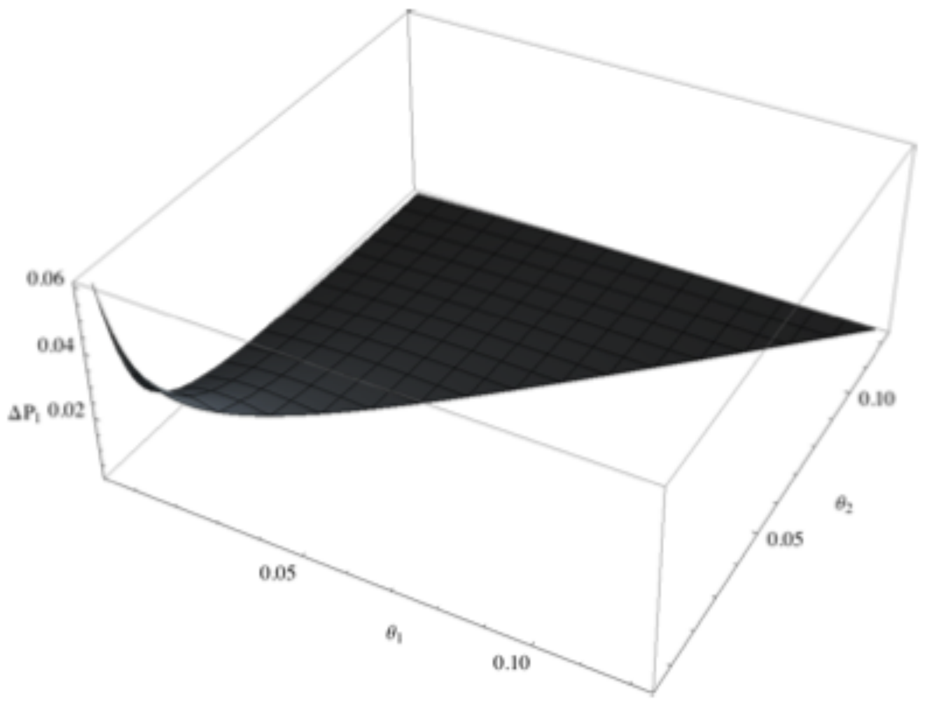

Figure 23 Individual derivatives error bounds

The surprise comes when this error formulation, shown in Figure 23, is compared to the principle error component analysis presented above; they are nearly identical. Thus, the individual mins & maxs method is as about as accurate as the graphical first order analysis, and far less computationally intensive than either the principle error components or the total first derivative. However, in general the error bounds calculated by this method grow larger as the pattern parameter increases, which is inconsistent with the graphical analysis. Nonetheless, they still provide sufficient range, and fast-enough computational speed to make star-ID much faster than existing algorithms.

4.4.3 Final Check

To provide statistical certainty that the proper star pattern has been identified, the interstar angles from the measured figure can be compared to those in the found entry in the catalog. An analytical approach to deriving the statistical confidence of a set of stars is given in [49], which describes the probabilities associated with a pattern of any number of stars selected in any manner across the celestial sphere. The root assumption is a uniform distribution of stars. Though that assumption is not true, it is nonetheless useful, and efforts have been made to filter the catalog to produce a more uniform catalog. [50]

4.5 The Core Star-ND Algorithm

Before considering all the conditions that can cause an error in the core algorithm, it is useful to describe the core algorithm at an intermediate stage. The basic approach to Star-ND is as follows:

• Preparation

1. Construct the star database of stars and their 3 closest neighbors.
2. Calculate the star-pattern parameters and build the K-Vector ND.

• Run-Time

1. Select an observed star closest to the center of the star image. $O(f)$
2. Find the p−1 closest stars to the selected star. $O(f \lg f)$
3. Compute the cosines and the parameter error bounds. $O(p)$
4. Use the K-Vector ND to search the database for entries within the error bounds.

O($k + d$)

5. Filter the results with the actual inter-star angle cosines. O(k)

The algorithm in this core form is not robust enough for a true application. Chapter V presents both the error scenarios and a technique for adding special “false” entries to the database to increase robustness and decrease the execution-time in case of an un-caught error scenario.

CHAPTER V

STAR-ND PROCESS

5.1 Limit Radii Angles

In order to proceed further, the reader should become familiar with four limit radii angles inherent in the star map. A "radius angle" is the angle made between a vector from the center of a sphere to the center of a small circle on the surface of the sphere and another vector from the center of the sphere to a point on the circle. These limit radii angles will allow the algorithms to make decisions about which stars are important for a particular process. Frequently, the computer algorithm can use the cosine of the angle in place of the angle, which is convenient since it is calculated with the very-fast scalar (inner) product.

5.1.1 Centroid Uncertainty

The first limit radius is the uncertainty of the direction to a star, σ. An image centroiding algorithm will determine the direction to an observed star with a precision related to this value. Formally, σ represents the standard deviation of the errors to the correct direction, and the sub-system designer may choose to use a higher multiple of σ, such as 3σ or 4σ as the limit for the accuracy, or may base the true precision on the brightness of the star. In this paper, σ will represent the maximum angular error that is statistically acceptable for a centroid.

5.1.2 Furthest 3rd Star

The second limit radius is the maximum angle to the $(p - 1)$th star in a star pattern from the central star, where p is the number of observed stars. Once the database of star-pattern parameters is available, it is a simple matter of an O(n) scan of the database to find the maximum angle of the last star in a pattern, where n is the size of the database. Alternatively, the value could be tracked as the database is created. For instance, for a pattern with $p = 4$ stars, the maximum angle to the 3rd closest star is known. After selecting a central star in a given image, any stars further away than this limit cannot be candidates for the nearest 3 stars.

5.1.3 Delaunay Radius

The third limit radius is the largest hole in the sky. Each star catalog has areas of the sky that are sparse and areas that are dense. The "largest empty hole" problem asks, "what is the largest radius of a circle containing no stars?" By answering this question, one can simply add the star direction uncertainty, and be guaranteed that any circle drawn in the sky of this radius will contain at least one star. Given any camera boresight direction, there is guaranteed to be a star within this radius, which suggests a starting point to solving the problem of finding the minimum required FOV. This value can be measured from a closed spherical Delaunay triangulation of the star map. "Spherical" means that the stars lie on the surface of a sphere, and thus the Delaunay triangulation is not the traditional 3-space triangulation, but a 2-space triangulation on the surface of the 3-D sphere. "Closed" is to say that every point on the unit sphere is interior to at least one spherical triangle. Algorithms exist to perform this triangulation in O(n lg n)-time, though it is often simpler to write the algorithm in O(n^2)-time. Like the maximum (p−1)th-star problem, it is only necessary to perform this calculation once for any given catalog, therefore the performance is not usually as important as the run-time algorithm.

5.1.4 Validity Limit

The fourth limit radius angle is that of the “validity limit,” the radius of the chosen camera’s FOV minus the maximum (p−1)th-star angle. Within this radius, all stars have all their nearby stars in the FOV, and they are called “valid” stars. Outside this radius, the nearest stars might be present, but they are not guaranteed, and those stars are called “invalid.” Later in the dissertation, a process to determine which stars are known to always be valid and which stars may be invalid is described.

5.2 Issues with Nearest Star Patterns

The use of the three nearest stars raises a number of concerns. First, if two of the near stars have approximately the same angle to the central star, a small error in their observed direction may cause them to be sorted incorrectly, causing the algorithm to calculate the incorrect parameter bounds and return no matching star patterns. Second, the presence of non-stars will cause the algorithm to compute the wrong star pattern parameters and search the wrong portion of the database. Third, sparse regions of the sky may cause one or more of the nearby stars to be outside the field of view, causing the incorrect selection of stars. The solution to these issues is to add carefully altered star-pattern entries to the database, essentially more "wrong answers." Then, at run-time, the star-ND algorithm will find the false entries and correctly determine the identity of the stars, instead of failing to identify them at all.

Other issues, such as the presence of false stars, require modifying the run-time algorithm to iterate through different selections of stars. As will be shown in the performance analysis, sorting the stars is the dominant term in the required time to execute the core star-ND algorithm. Avoiding unnecessary re-sorts will be the primary objective. The limit radii angles will be used to intelligently decide which stars should be used or sorted for a particular purpose to reduce the number of iterations.

5.2.1 The Wrong Order

Consider a star pattern in which the closest star and the second closest star, $\hat{s}_1$ and $\hat{s}_2$, are at nearly the same distance to the central star, $\hat{s}_0$. If the uncertainty in the directions to these stars obtained by the centroiding process is greater than the difference in the angle to these stars from the center, then the run-time algorithm could sort them in the wrong order, as illustrated in Figure 24. As will be shown later in this dissertation, it is computationally expensive to postulate a new permutation of the selection of the nearest three stars, so repeating the Star-ND process is undesirable. Instead, the solution is to create a false entry for the star database, with the equidistant stars swapped in order. By doing so, the run-time algorithm will find the false entry and correctly identify the stars, even when the stars have been sorted in an incorrect order. It is necessary to calculate the star pattern parameters with the new order, and in some cases, to enforce the values to be within the tautological parameter bounds. The solution applies not only to the closest two stars, but to any others in the star pattern, even the next star beyond the closest selected stars. It is even possible for three stars to be at nearly the same distance from the central selected star, generating a total of five wrong answers for a star pattern with 4 stars, $p = 4$.

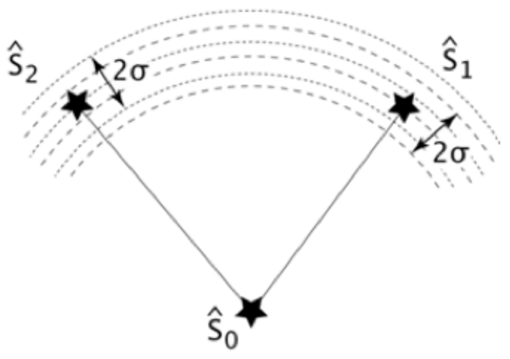

Figure 24 The order in which the stars are to be sorted could be calculated incorrectly given the uncertainly in their calculated centroids

5.2.2 The Wrong Stars

Consider now that non-stars appear in the star image. Examples of such problems include visible planets, orbiting spacecraft, nearby dust, etc. If they are near the center of the image, the non-stars may be selected during the identification process as a star. If this happens, the wrong star pattern parameters will be calculated, and no matching star pattern will be found. The run-time algorithm must be able to handle these situations by selecting different stars. To modify the Star-ND core algorithm to properly handle the presence of false stars, four important points should be noted:

1. The maximum angle (and therefore the minimum cosine) between a selected star and its (p−1) nearest star is known from the database,

2. The database already contains false entries for situations in which the orders of the nearest stars may be numerically poorly-determined,

3. The dominant term in the execution-time of the star-ND algorithm is sorting the (p−1) nearest stars, and

4. Fewer numbers of false stars are more probable than higher numbers.

Correspondingly, when star-ID fails, constructing the next permutation of the stars does not need to consider:

1. candidate neighbor stars that are too far away from the selected star,

2. altered orders of nearby stars,

3. different selected central stars until all possibilities have been exhausted, or

4. large numbers of false stars before small numbers.

5.3 Star-ND Run-Time Algorithm

Thus, the flow chart for the Star-ND is shown in Figure 25. As an example of the nearest star selection process, consider a star, “0”, and its nearest neighbors, “1”, “2”, “3”, “4”, and “5”, sorted by increasing angular distance from “0”. The first selection of stars to try is (0, 1, 2, 3), but if that does not succeed, then one of the nearby stars may not be a real star. The combinations (0, 1, 2, 4), (0, 1, 3, 4), and (0, 2, 3, 4) should be tried, and if none of those succeed, then there may be two false stars. Those combinations, in order, would be: (0, 1, 2, 5), (0, 1, 3, 5), (0, 1, 4, 5), (0, 2, 3, 5), (0, 2, 4, 5), and (0, 3, 4, 5). (These specific orders allow the pattern to be incremented in O(*p*) time.)

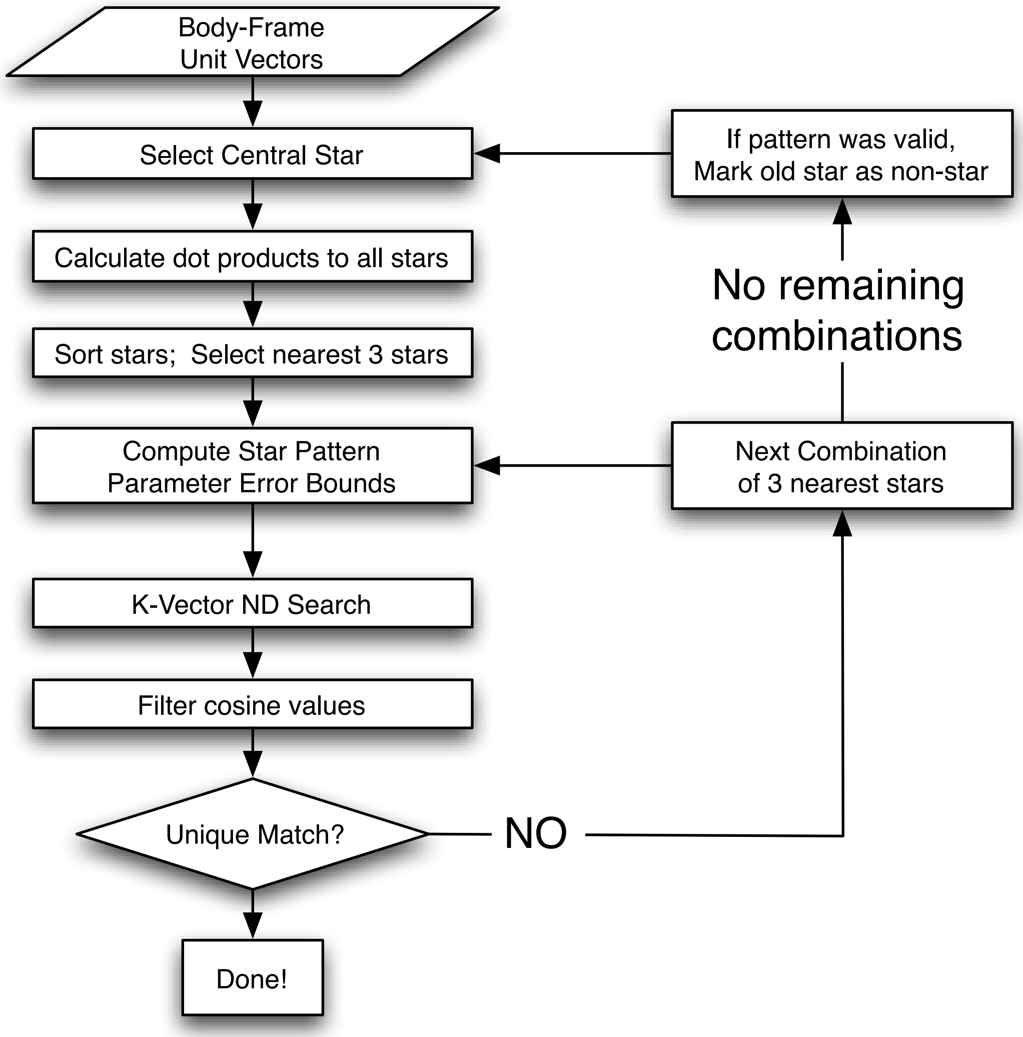
Body-Frame Unit Vectors
Select Central Star
Calculate dot products to all stars
Sort stars; Select nearest 3 stars
Compute Star Pattern Parameter Error Bounds
K-Vector ND Search
Filter cosine values
Unique Match?
Done!
NO
Next Combination of 3 nearest stars
No remaining combinations
If pattern was valid, Mark old star as non-star

Figure 25 The Star-ND algorithm flow chart

Note that for each new presumed non-star there are $(b + 1)(b + 2)/2$ new combinations to consider, where b is the number of "bad" or false stars. Thus, the total number of star combinations considered is $(b + 1)(b + 2)(b + 3)/6$. This is very large, but applies only to the $p = 4$ case. In general, the total number of cases to consider is $O(b^{(p-1)})$. Such an exponential growth suggests that the number of stars in the pattern should be kept low to reduce the time required to compute all of the cases.

With such a large amount of time being spent on false star combinations, why should the algorithm not consider a new selection of its central star until all the nearby stars have been considered?

First, it can. The algorithm coded for performance testing includes a cutoff for a maximum number of false stars. However, this feature has been disabled in order to evaluate the worst-case performance. Thus, a star tracker sub-system designer is free to tell the algorithm that when it thinks there may be 5 false stars (or whatever threshold is selected) that it should instead consider that the selected central star is false.

Second, the amount of time required to calculate the dot products between all of the stars with the selected central star and sort them to find the nearest 3 is large compared to the amount of time it takes to calculate the star pattern parameters and search with the K-Vector ND. Despite the fact that the measure and sort process takes $3f$ floating-point multiplies, $2f$ additions, and $O(f \lg f)$ floating-point compares, the constant factors are large. The time required to attempt star-ID for a single combination of stars, for $p = 4$, is 37 additions, 17-multiplies, 6 divides, 6 square roots, 9 compares, and $O(d+k)$ operations for the K-Vector ND to search, where $d = 2p - 5$, and k is generally between 1 and 50. The sorting process also has a large number of control-flow changes, which makes it even more expensive on systems with long processor pipelines (such as the platform on which these algorithms were tested). This analysis would be different if the memory access characteristics were different for the platform in which the algorithm is implemented. For instance, an FPGA or custom ASIC would allow a faster dot product and an O(*f*) sorting implementation; they might also have relatively slow memory access, which might change the choices presented in this dissertation. See Chapter VIII, "Directions for Future Research" for preliminary discussion of using parallel hardware with Star-ND.

Furthermore, if the central selected star is the false star, then all of the selections of nearby stars will fail, and when that happens the selected star can be rejected from the list of stars. There are two caveats to this rejection:

1. if the central star or one of its nearest stars is in fact a conjunction between a real star and a false star (thereby perturbing its direction beyond the centroiding accuracy), and

2. if the selected star is too close to the edge of the field of view, which leads to the third error case: sparse regions and invalid star patterns.

The subsystem designer might consider the above cases to be frequent enough to warrant not rejecting unidentifiable stars as false stars before recursive star-ID.

5.4 Sparse Regions and Invalid Star Patterns

Remembering that the maximum angle to the $(p-1)^{th}$ nearest star in the whole database is known, the run-time algorithm will know if a star is close enough to the center of the FOV to have all of its nearby stars in the FOV. If the star is too close to the edge of the FOV, then the nearby stars might be outside of the FOV. If this could be the case, the star pattern is “invalid.” Clearly, to avoid selecting stars without their nearest neighbors in the FOV, it is desirable to select a star near the center of the FOV. In highly-populated regions of the sky, this choice is sufficient to select a star with its nearest stars inside the FOV. However, the non-uniform distribution of the stars in the sky means sparse regions may not have enough stars if the FOV is too small.

To quantitatively measure the required size of the FOV, the number of potentially invalid star patterns is counted. To determine if a particular star has a potentially invalid pattern, a postulated optical axis is positioned away from the $(p-1)$th nearest star in the pattern by half the FOV size in the direction of the central star. If there is a star nearer to the optical axis than the selected star, with its $(p-1)$ nearest stars in the postulated FOV, then the selected star will probably not be selected for star-ID unless its $(p-1)$th nearest star is in the FOV. If there are no stars closer to the postulated optical axis with their $(p-1)$ nearest neighbors in the FOV, then the given star’s pattern may potentially be observed as invalid at run time. The validity limit is illustrated in Figure 26 along with the location of a given star pattern when searching for potentially invalid star patterns.

Potentially invalid star patterns can have wrong answers inserted into the database as well. For the given potentially invalid star, there may be stars located at an angle from the central star further than the $(p-1)$th nearest star but closer than the known maximum and closer to the postulated optical axis than the real $(p-1)$th star. Such stars can replace the actual $(p-1)$th nearest star in a new false entry into the catalog. For very sparse star patterns it is reasonable to use this procedure for some of the closer stars as well, for instance, the ($p - 2$)th star.

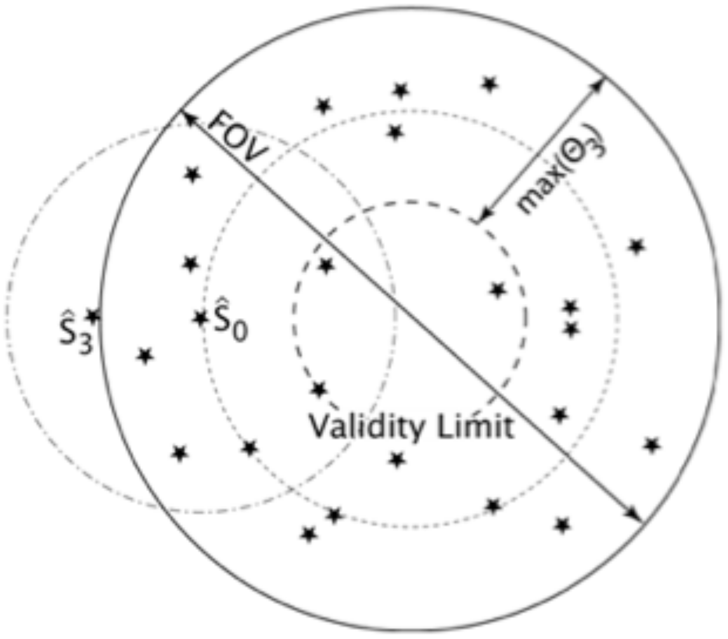

Figure 26 Illustration of the validity limit and checking the validity of a star pattern

There is, of course, one case in which such a stand-in will not be possible, and that is the one case in which the ($p - 1$)th nearest star is at a maximum distance - which, unfortunately, is generally in a very sparse region of the sky. The actual stars will determine if this case is potentially invalid or if there is another star that would be identified first. In tests, it is normal for this region of the sky to be so sparse that there are no alternate stars to select first. Fortunately, however, there are so few stars in this region that forming new combinations has almost nothing to run on, so switching to another central star happens very quickly. Also, the low number of stars allows the sorting to take place rapidly. Oddly enough, there are usually stars in “clusters” near these sparse stars that have their nearest stars very close, providing reliable identification after only 2 or 3 different selections of a selected star.

5.5 Database Growth

When adding false entries to the database, the bounds of the data range do not change, but the density of items does, effectively increasing the number of results returned by a range search, thereby increasing the search and filter times. Thus it is desireable to evaluate the growth rate of the database as items are added.

In contrast to other state-of-the-art star-ID algorithms that use pairs of observable stars, Star-ND begins with a single entry per star, then other “false” entries are added. Basically, the size of the database grows linearly with the number of stars, instead of quadratically as do algorithms containing every observable pair of stars in their database. Each entry in the Star-ND database contains the identities of the four stars in the pattern, and the dot products of the six star combinations. For 7,122 stars, the size of the database is listed for two choices of data types in Table 6.

Table 6 Memory used by Star-ND

	4-byte int 8-byte float	2-byte int 6-byte float
Catalog	167	126
Database	446	306
K-Vector ND	56	28

When contrasted with other star-ID algorithms that use all observable star pairs or triples, this new algorithm yields quadratic memory savings. But adding so many extra items to the database will make it grow larger. By varying the size of the database, and holding *p* constant, the size of the database also grows nearly linearly, as illustrated in Figure 27, which plots the number of entries in a database vs. the number of stars in the original sub catalog. The plot has been drawn with a logarithmic scale on both axes to exaggerate any non-linear behavior. The sub-catalogs were extracted from a larger catalog by visual magnitude, with limits ranging from 3.5 to 9.0 in increments of 0.5.

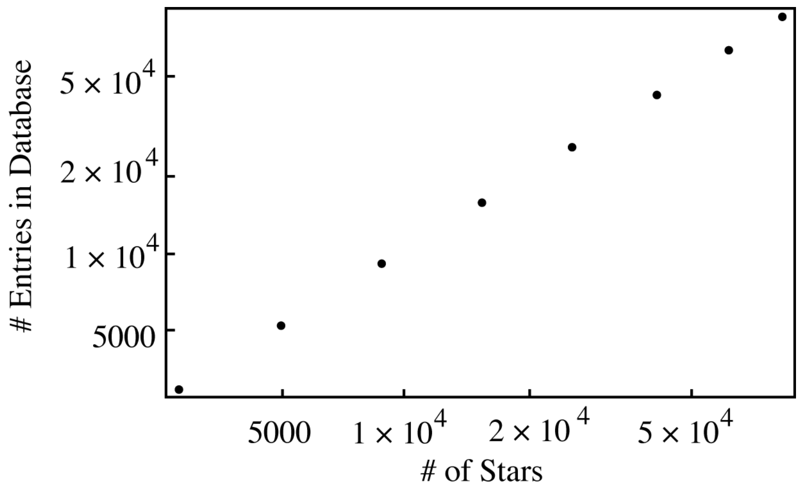

Figure 27 Log-log plot of the size of the database with false entries added vs. the number of stars

While using a star pattern consisting of the closest stars to a particular star has been considered to be the “wrong” answer to the star-ID problem from the viewpoint of robustness, the technique of adding additional false entries to the database can preemptively compensate for predictable issues and thereby reduce the run-time and failure rate of the resulting star-ID algorithm. Before presenting the results of a testing the performance and success of the Star-ND technique, a brief consideration of two recursive star-ID approaches are given.

5.6 Recursive Star-ID with the K-Vector ND

Recursive star-ID differs greatly from Lost-in-Space star-ID because *a priori* information regarding the identity of one or more stars, or an estimate of the attitude is available. In recursive star-ID, one approach is to use the estimated attitude to find all stars that should be in the image then match each star, another approach is to search for each observed star in the catalog. The K-Vector ND can be used in both approaches, but it has decided advantages in the latter.

In both approaches, an estimated attitude is required, to search for a particular star, and the precision of the estimate will contribute to the success of finding the correct identity. Other data structures, such as the Delaunay Triangulation and Star Neighborhod Table (see Chapter II), can offer recursive star ID without an estimated attitude. Also in both approaches, a catalog consisting of stars represented as unit vectors, and a K-Vector ND of the catalog are constructed. In the first approach, the estimated attitude and the FOV are used to generate a set of search ranges for the entire FOV. In the second, search ranges are generate for each star independently. While the second approach at first appears to spend more time searching due to the overhead associated with the K-Vcetor ND search, it is in fact much faster, as illustrated by analyzing the steps below:

5.6.1 Approach 1

First, the estimated attitude and FOV are used to generate orthogonal search ranges encompassing the entire FOV in the inertial reference frame. The camera's rectangular FOV is circumscribed with a circular region, and that region is extended to fill an orthogonal region. Next, the K-Vector ND is searched with these bounds. The stars are then transformed to the camera's reference frame to produce a set of predicted stars. Many of the stars will not lie within the bounds of the camera, because both the process of circumscribing the FOV and the process of finding the orthogonal ranges enlarge the search area and return stars that are not true results of the search. Stars outside the rectangular field of view are removed from the predicted star set. Lastly, another search commences to match the observed stars with the predicted stars. A variety of techniques can be used for the last step, including a linear search Delaunay triangulation, or one of the 2D multi-dimensional searches. For instance, the observed stars can be placed into a 2D K-Vector ND representing the image plane, as illustrated in Figure 28; then a search for each predicted star in the image plane can be performed using the estimated attitude uncertainty to create a search range. Given a particular uncertainty in the attitude estimate, there may more than one possible match for a given observed-predicted star pair, so additional techniques may be required to correctly resolve these multiple matches.

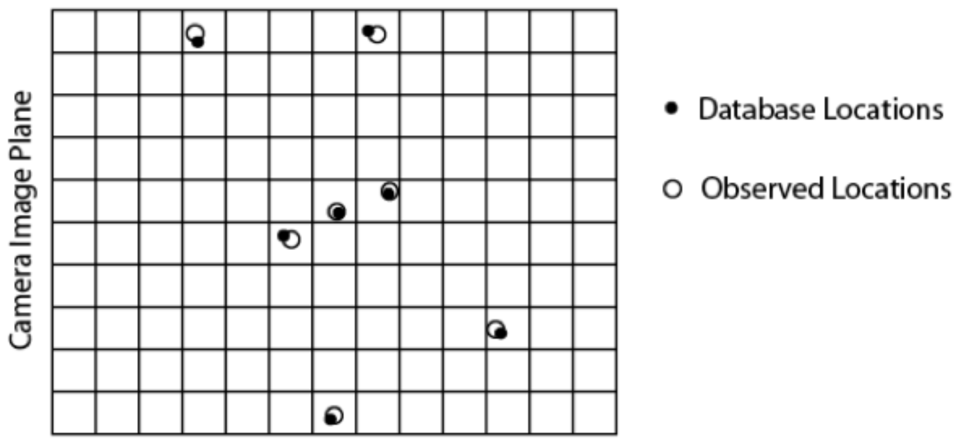

Figure 28 Converting the image plane into a K-Vector ND

As a side note there are two alternate ways to remove out-of-bounds stars. The first would be to take their dot products with various camera bounding unit vectors to avoid using the time to transform out-of-bounds stars. Five inertial unit vectors (the boresight, and vectors perpendicular to the four sides of the FOV directed generally towards the boresight axis) can be used to filter the stars in four zones. The first zone, as depicted in Figure 29 is the area less than half of the minimum FOV width from the boresight axis. Stars within this area can be accepted as "in-bounds" with one dot product to the boresight axis and one comparison. The largest zone consists of the stars further from the boresight than half of the diagonal FOV, but within the orthogonal search range. Such stars can be rejected with a second dot product and comparison. The last two zones are both outside of the smallest zone and inside the largest zone, but may lie inside or outside the actual FOV. The four unit vectors perpendicular to the FOV sides provide a quick check as to whether these stars are inside or outside the actual

FOV. Statistically, about half of the remaining stars are outside the FOV, it is not known ahead of time which of the 4 dot products is the best to use to rejecte them. The half of the stars that are within the FOV will require all 4, the other half stand an equal chance of being rejected by each side, meaning that O(4 * *f*/2 +2 * *f*/2)=O(3*f*) dot products are taken to perform this step. By acknowledging that an additional 3 dot products are needed to transform each in-bound star, O(3 * *f*/2), for a total of O(4.5 *f*), the analysis is clearly in favor of simply transforming each star into the inertial frame.

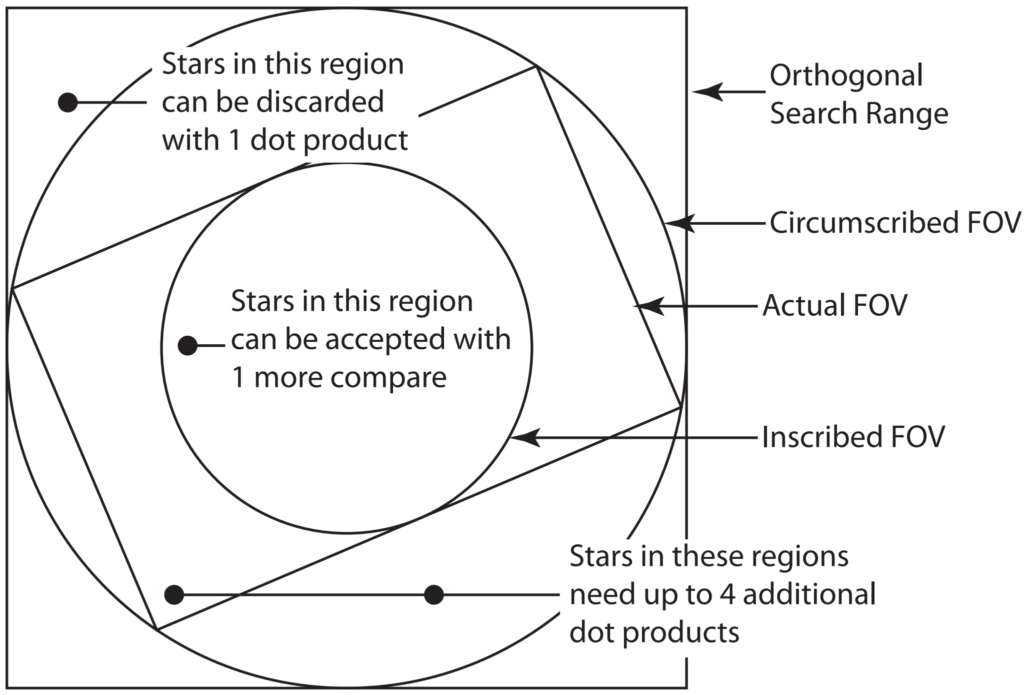
Stars in this region can be discarded with 1 dot product
Orthogonal Search Range
Circumscribed FOV
Stars in this region can be accepted with 1 more compare
Actual FOV
Inscribed FOV
Stars in these regions need up to 4 additional dot products

Figure 29 Zones for accepting and rejecting stars before transforming

However, the transform itself can be used in the filter transform process. Using the pinhole camera model, the coordinates in the image space of the star and the actual FOV bounds are directly related, as illustrated in Figure 30. The only additional operation beyond the coordinate transformation is the extension of the unit vector onto the projection plane, which is accomplished by scaling the body-frame unit vector by the inverse of its *z* value. Of course this requires transforming the *z* component first, and a check can be made to ensure the *z* value is large enough, similar checking the largest zone as above. Next, since a non-square FOV is typically arranged with its *y* dimensions as smaller than *x*, rejecting stars based on their *y* component would eliminate more stars than checking the *x* component. If the absolute value of the *y* component is less than the product of the actual FOV's limit in the *y* direction times the *z* value, then the vector cannot lie within the bounds, and the division by *z*, which usually takes longer than a multiply, need not be done, nor does the *x* component need to be calculated. Lastly, the *x* component can be transformed, and a similar check to the *y* component applied. Using this technique for a check performs the lowest required number of floating point operations to determine if a star is indeed within the bounds of the actual FOV. There is no guarantee that time will be saved, however, because different processors and math libraries use different amounts of time to perform the various math operations, from multiplication, to memory retrieval, and division. Possibly more significant is processor instruction pipline, which is quite sensitive to control flow changes, of which there are many in the check-reject technique.

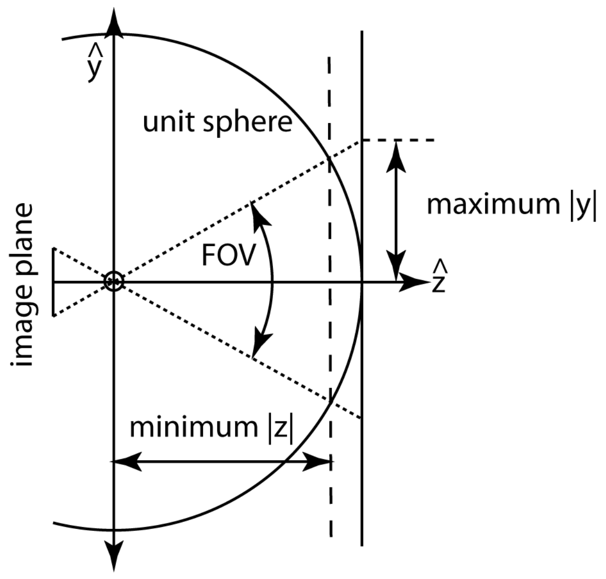

Figure 30 Scaling the pin hole model

5.6.2 Approach 2

A bigger reason not to use any of the above techniques for checking bounds is the fact that the above approach is asymptotically slower than the following. Using the estimated attitude, each observed star is transformed into the inertial reference frame. Using the estimated attitude uncertainty, each inertial direction is enlarged into a search range. The K-Vector ND then searches for each star individually in the catalog, returning only a small set of possibilities for each observed star. Of course, observed stars with no matches are likely not stars; observed stars with one match have been possitively identified; observed stars with more than one match need further reconciliation. The two approches are summarized in Table 7.

Table 7 Steps in Two Selected Recursive Star-ID Algorithms

Approach 1		Approach 2	
Calculate Search Range	O(1)	Transform	O(f)
K-Vector ND Search	O(d+2 f)	Calculate Search Ranges	O(f)
Filter	O(2 f)	K-Vector ND Search	O($f\,(d + k)$)
Transform	O(f)	Reconcile Multiple Matches	O(?)
Build Search Structure	O(f)	N/A	
Search for each star	O(f)	N/A	
Reconcile Multiple Matches	O(?)	N/A	

As a sample technique for resolving multiple matches, the reader may note that the initial attitude estimate is typically more accurate in boresight direction than in rotation about the boresight. The observed-predicted star matches which have a single possible match can have their error information coordinatized in both the x and y directions and as a rotation about the centroid of the stars used to perform the initial attitude estimate. Consistencies in these errors may be tracked and used to resolve the multiple match cases.

CHAPTER VI
PERFORMANCE TEST RESULTS

6.1 Simulated Star Directions

To determine the best-case performance of the Star-ND algorithm, the first test provides a set of unit vectors taken directly from the catalog. By using the same catalog to construct the database, there will be no dim stars fluctuating just over the brightness threshold. 10,000 random attitudes are generated, and the stars which would be visible in the FOV are selected and transformed into the camera frame. The star directions, however, are perturbed by a random direction and angle up to 17 arc-seconds to simulate the unavoidable errors due to centroiding. To mimic the effect of non-stars in the image, additional unit vectors are added at random locations both in the image plane and in the list of stars themselves.

The Star-ND algorithm is tested for performance under two circumstances: with and without false stars. In both cases, the same test data sets are used. For a database of 7,122 stars and a $p = 4$ star pattern, 10,000 random test attitudes of the star camera are generated, and the observed stars are provided to the ID algorithm as body-frame unit vectors. The test FOV was a 17º-diameter circle. Each star had its direction perturbed by as much as 17 arcseconds. The algorithm code was written in C, compiled by GCC with optimization set to -O3, the most optimized setting. The processor was an Intel Core 2 Duo running at 2.33 GHz, though the algorithm was single-threaded and used no more than a single core at any one time. The memory bus has a clock rate of 667 MHz.

The time is measured with the UNIX system call “getrusage(),” which reports the number of microseconds used by a process. To improve the precision with which the results are measured, each test case is repeated 100 times, and the total time divided by 100, placing the precision of the measurements in the 0.01μs range, or 24 clock cycles. 10, 000 tests were run, at random attitudes.

6.1.1 Without False Stars

False stars are not included in this test, and the results are shown as a histogram in Figure 31.

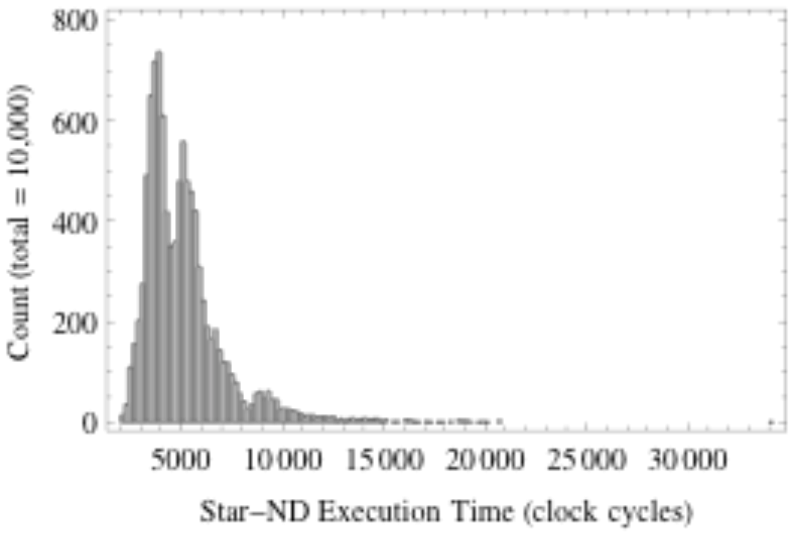

Figure 31 Histogram of Star-ND execution time for the ideal case

The results show that the mean execution time is 4,538 clock cycles, and the median is 4,170 clock cycles. There is one result at 31,362 clock cycles, which corresponds to a single case above 20,000. The minimum time was 1,724 cycles. The algorithm correctly identified four stars in all test cases.

An analysis of the algorithm shows that the run-time should be dependent on sorting the observed stars and the run-time of the K-Vector ND search, totaling O($d + k + f \lg f$). Plotting the execution times versus these factors in Figure 32 shows this to be the case, revealing the O($f \lg f$) sort time associated with finding the 3 closest stars using the well-known MergeSort algorithm. Additional variability in the run-time is caused by the filter process, which checks 6 inter-star angles. At each angle, it may reject a particular candidate and forego the further checks. It is clear from the plot that the sorting step is the dominant term in the performance of the Star-ND algorithm.

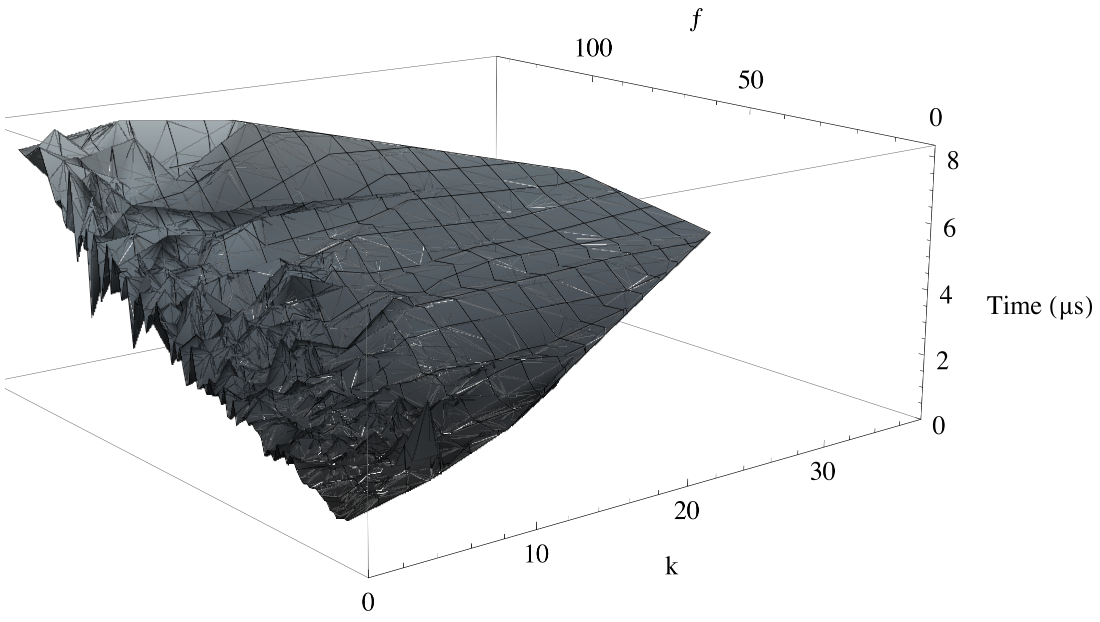

Figure 32 Star-ND execution time vs. *f* and *k*

6.1.2 With False Stars

Next, the performance test is repeated, but false stars are added to the list of observed stars. As the test is repeated with higher numbers of false stars, the old false stars from the previous iteration are retained, so each iteration only adds one false star to each test case. As mentioned earlier in the paper, in order to achieve a worst-case measurement, the run-time algorithm makes no assumptions about the maximum number of false stars, which means when it fails to identify stars, it considers all the possible combinations of nearby stars before it will reject the central star as a false star.

The results in Figure 33 show discordance amongst the minimum, median and maximum execution times. The reason is that non-stars affect the algorithm differently depending on their location in the FOV. If non-stars are closer to the edges of the FOV, the stars near the center of the FOV remain unaffected. If the non-stars are near the center of the FOV, then they may affect the selection of stars, causing an iteration of the Star-ND algorithm. There is, of course, an overall increase in execution time of the algorithm when false stars are present caused by increased time to compute inter-star cosines and sort them. The results show the algorithm execution time can significantly increase for the case of many false stars, but this is expected behavior for most star-ID algorithms.

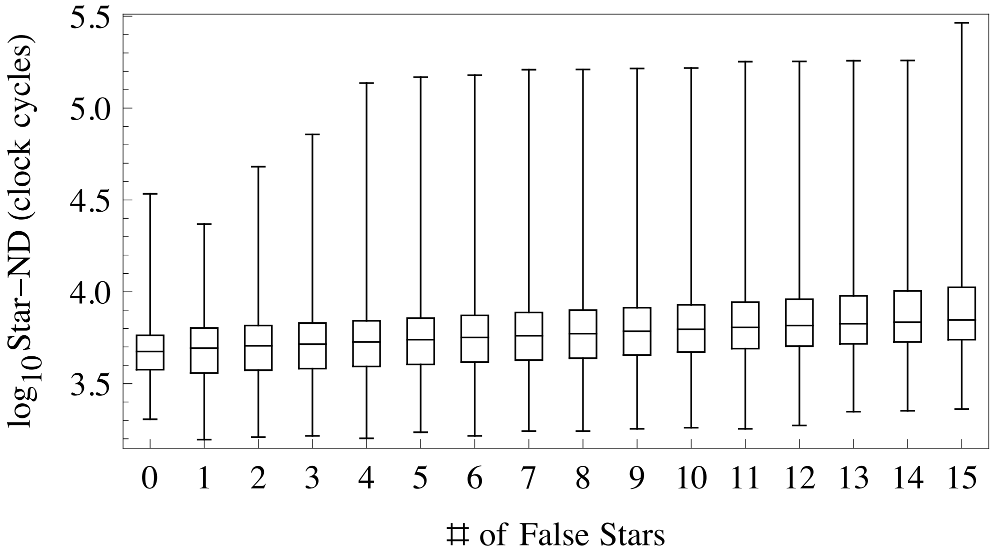

Figure 33 Logarithmic plot of the execution time for the false-star test

6.2 Simulated Images

To determine a more practical estimate of performance, the Star-ND algorithm is tested from data obtained from simulated star images. The images were simulated by Virtual Star Tracker (VST) (© Dr. Daniele Mortari, Texas A&M University), which is a highly-accurate star-camera image simulator. VST generated 1000 simulated images, which were imported to custom-written image processing and centroiding algorithms. The centroid directions were used as input to the Star-ND algorithm. The run-time of the Star-ND algorithm was measured in repeated loops to increase precision.

6.2.1 Simulator

VST was created for the purpose of testing Star-Tracker algorithms. Over the years it has become increasingly physically accurate. Notably, my work on VST has included improving both the direction (stellar aberration, proper motion, light-time delay) and brightness (instrument magnitude, Milky-Way background). Other faculty, students and staff have enhanced VST to import Satellite Tool Kit (STK) propagation results for orbital motions, produce standard astronomical FITS files as output and simulate exact camera noise histograms. VST is extremely useful for testing the performance and success of image processing, centroiding, star-ID, and attitude determination algorithms, because it can perform in a few minutes what would take hours of experiment, months of planning and many thousands to millions of dollars to do in real-life. In most cases, the physical effects simulated by VST are at least one conceptual level more accurate than required by the algorithms under test.

For the VST simulation, the simulated star camera was placed on a geosynchronous satellite, controlled to nadir pointing, with the star camera pointing away from Earth. This gives a slight rotation to the satellite equal to the rate of the earth's rotation, causing the star camera's boresight axis to sweep through the field of stars. While it does not accomplish a sweep near all stars, through the several-hour duration of the simulated experiment, the boresight did move from a sparse area of the sky to a Milky-Way-dense portion of the sky.

VST can access the Bright Star Catalog and the Hipparcos Catalog [51]. For the purpose of this simulation, the Hipparccos catalog was used. The Hipparcos Catalog includes a detail listing of stars through visual magnitude 10, and a few stars dimmer through magnitude 12. For the purpose of these simulations, all stars through visual magnitude 11 were included in the images. The Hipparcos Catalog includes proper motion information for many of the stars, and that information was included to move the stars to their estimated position as of Julian Date: 2455030.927025463, which was the beginning of the simulation.

In addition to the proper motion, the stars' directions were also altered by the special relativistic effect known as stellar aberration, also known as the "headlight" effect [52]. In short, stellar aberration is the apparent shift in a star's direction due to the motion of the observer. Though the effect is small, 20" (10^{-4} radians) maximum, and gradual through the star field, high-sensitivity star cameras have the capacity to observe a discrepancy in large inter-star angles across wide fields of view.

As illustrated in Figure 34, the light "ray" with the velocity $\vec{u}$ such that $|\vec{u}| = c$ is observed by the telescope with velocity $\vec{v}$. To simplify the equations, the observer's velocity is aligned with the *x*-axis.

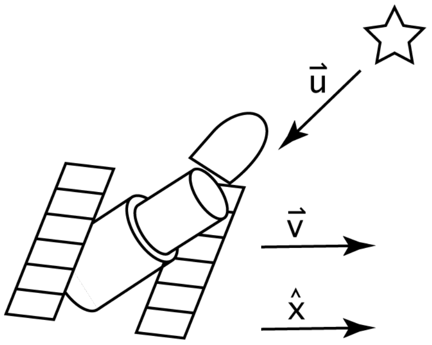

Figure 34 A spacecraft with velocity *v* observing a star

Relativistically, the observed velocity, $\vec{u}'$, is calculated by Eq. (9).

$$u'_x = \frac{u_x - v}{1 - \frac{v u_x}{c^2}}$$
$$u'_{yz} = \frac{u_{yz}}{\gamma\left(1 - \frac{v u_x}{c^2}\right)}, \quad \gamma = 1 \Big/ \sqrt{1 - \frac{v^2}{c^2}} \tag{9}$$

To speed the calculation, the formal relativistic equations are simplified by assuming that the observer's velocity is small compared to the speed of light, $v/c \approx 0$. The formula reduces to the well-known transport theorem [53] in Eq. (10): the direction to each star (represented as a unit vector in sun-centered inertial space) is multiplied times the speed of light. The velocity of the observer (satellite velocity in ECI is added to the Earth's orbital velocity) is then added to the light-vector. (Note that the direction to the star is opposite of the light velocity.)

$$u'_x = \frac{u_x - v}{1 - \underbrace{\frac{v u_x}{c^2}}_{0}} \Rightarrow u_x - v, \quad \gamma = 1 \Big/ \sqrt{\underbrace{1 - \frac{v^2}{c^2}}_{1}}, \quad u'_{yz} = \frac{u_{yz}}{\underbrace{\gamma\left(1 - \frac{v u_x}{c^2}\right)}_{1}} \Rightarrow u_{yz} \tag{10}$$

The sum is re-normalized and used as the apparent direction to the star. Analysis shows that the discrepancy between the special-relativistic stellar aberration and the transport-theorem stellar aberration is less than 2x10^{-9} radians or less, five orders of magnitude smaller than the effect, as shown in Figure 35.

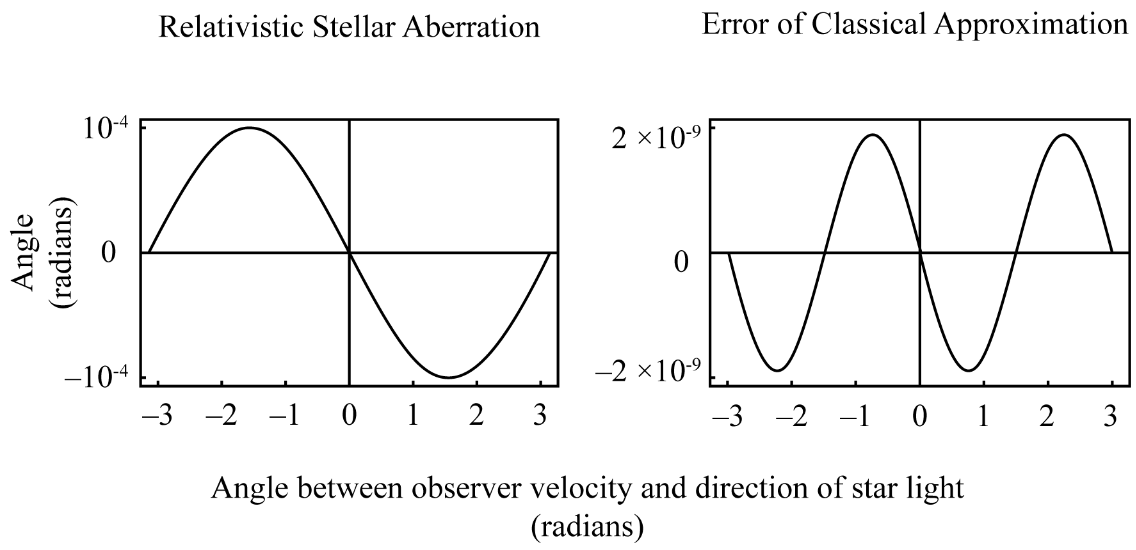

Figure 35 Stellar aberration & classical approximation error from Earth

6.2.2 Instrument Magnitude Correction

One of the enhancements I contributed to VST is the instrument magnitude correction, yielding a more accurate brightness for each star. For this correction, the spectral profiles of the stars are convolved with the quantum-efficiency curve of the sensor's CCD to provide a more accurate prediction for the expectation of the number of electrons from the star camera.

The hipparccos catalog provides spectral type information for each star. The temperature portion of this information is used to associate each star with a spectral profile, as calculated by Kurucz [54]. A sample plot of Kurucz data for a star with surface temperature of 7000 K is plotted in Figure 36. To simplify the model, all stars are assumed to have the same metallicity as Earth's sun. Each star is scaled to the sun, using its known energy output, spectral curve, and visual magnitude to calibrate the scale. The spectral curve for each temperature (from the corresponding spectral type) is convolved with the Johnson V curve to determine how much of the curve's energy is measured in the visual spectrum (and accounted for in the visual magnitude). Then, using the visual magnitude for each star, the star's total power flux is calculated and stored.

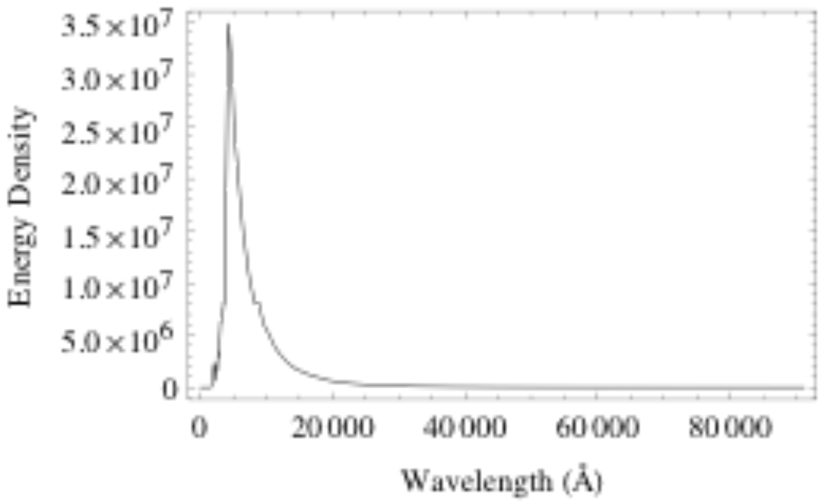

Figure 36 Sample Kurucz distribution for T = 7000 K

Once a specific camera is selected, its quantum efficiency curve is convolved with each spectral curve and the photon-wavelength energy relation, yielding a coefficient that determines the fraction of that curve's energy that is detected by the camera. The expectation value for the number of electrons produced by the CCD is then easily scaled by the star's power flux, the camera aperture and the integration time.

6.2.3 Non Stars (Planets)

Of all visible non-star objects, only planets were included in the simulation. VST's simulation of planets is orbitally accurate, but not as photometrically accurate as the procedure describe for stars above. Planets' apparent directions are adjusted both for the stellar aberration (as described above for stars) and for light-transit time to the 1st order. It is interesting to note that for the special case of Venus and Earth at closest approach, these effects approximately cancel each other.

6.2.4 Camera Parameters

VST assumes a pin-hole model of a camera, yielding a simple linear relationship between the direction of incoming light and the resulting point-spread center on the sensor. It is assumed that any real star camera would have its non-linearities mapped ahead of any mission use of the camera. For the simulation, a 1082 x 1312 pixel CCD was used, with a 0.008 mm pixel pitch. The sensor was set at a focal length of 35 mm, and the camera aperture was set at 3848 mm^2. The simulated camera was set for a half second exposure time, and a 15-second image interval. Over 1000 images, the duration of the experiment was just over 4 hours.

6.2.5 Image Noise

For this simulation, the histogram-replicating feature of VST was not used; VST's default is to simulate the background thermal radiation noise with a Gaussian distribution of random pixel values. The peak and standard deviation of the distribution are set to coincide with some available real night-sky images, to ensure the noise level was realistic. Some of the noise effects not simulated include: vignetting, proton streaks, stuck pixels, and resident space objects.

6.2.6 Point-Spread-Function

Real star cameras intentionally blur starlight over a few pixels to enhance the centroiding accuracy. VST uses a pure Gaussian distribution to simulate the point-spread function of the camera. For this test, the standard deviation of the blur was set to 1.5 pixels. The Gaussian is not rendered beyond the 4σ radius. A sample simulated image is shown in Figure 37; the brightness of the image has been adjusted to reveal the detail in the image.

6.2.7 File Output

By default, VST exports images in FITS [55] format, using 16-bit unsigned values. Since the FITS standard is signed values, the camera parameters were selected to avoid saturation beyond a 15-bit maximum value.

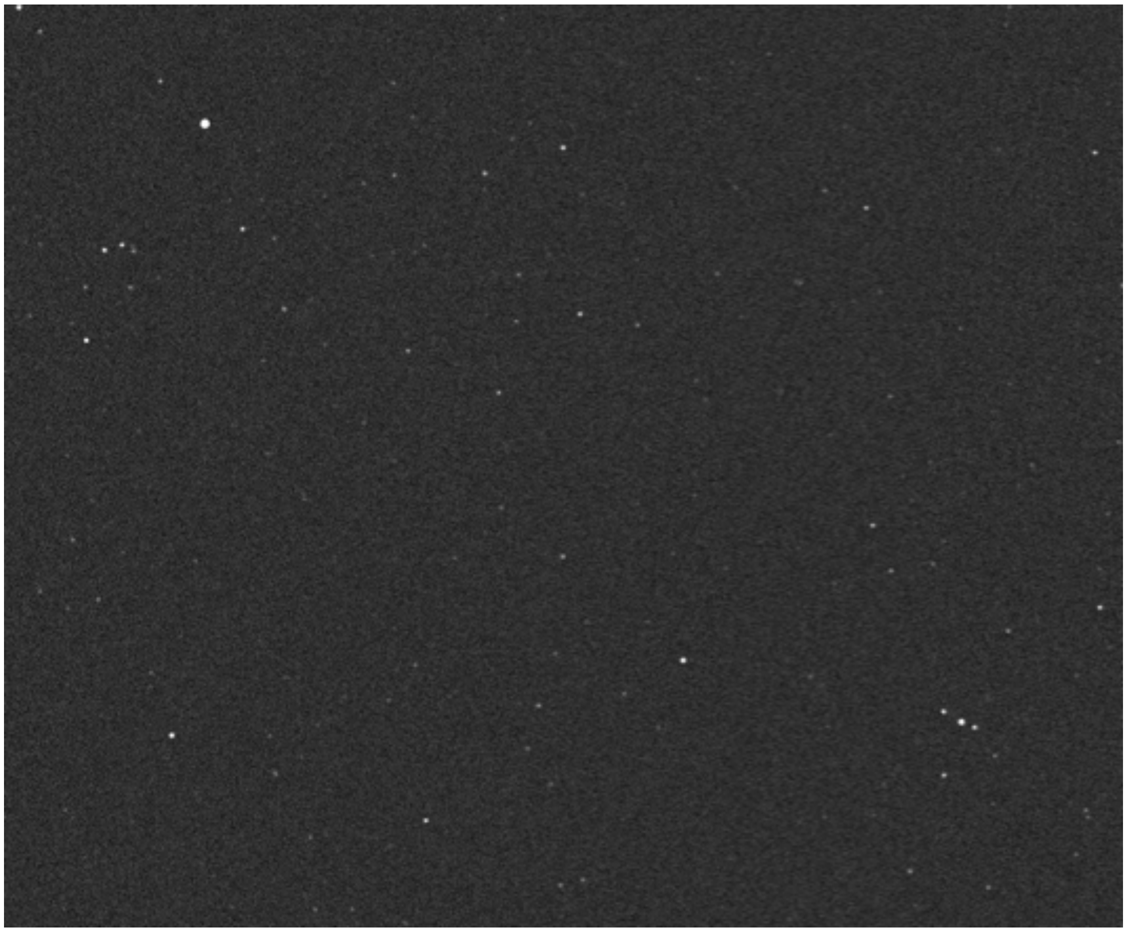

Figure 37 Sample simulated image

6.2.8 Image Processing

Before centroiding, a 5x5 Gaussian filter with a 1.5-pixel standard deviation was passed over the image to reduce error due to random thermal noise. To improve filter speed, the filter was carried out as successive column and row filters, each with a 1 x 5 gaussian signal. Mathematically, these two processes are identical to a single Gaussian block, but require approximately 1/5 the computational complexity.

After the Gaussian filter, pixels brighter than a specified threshold were selected and adjacent pixels were collected into pixel groups. A centroid was calculated for each pixel group, based on the brightness and location of each pixel.

To increase the brightness precision further, a region of interest occupying an annulus extending from twice to thrice the radius of each centroid was defined [56]. Inside this region, the pixel values were summed to generate an average background brightness, which is then subtracted from each centroid to produce a more accurate electron count for each star. As the algorithms were written quickly and automated, no attempt was made to correct for cases in which other stars appeared inside the annulus.

A plot showing the brightness of each star in each image is given in Figure 38. For brighter, easily identifiable stars, the plot shows the characteristic variation of measured brightness. Once the more accurate electron count was calculated, the centroids were rejected if they fell below a slightly higher level, to improve consistency of the brightness level threshold.

No attempt was made to detect or separate double stars in image processing or centroiding.

Based on the observed variation in the measured brightness of the stars, a new set of false star patterns were added to the star database. The minimum brightness of each star pattern was checked, and a new false pattern entry was constructed if one of the three nearest stars had a brightness within 5% of the minimum threshold. The result was to dramatically increase the catalog size by about 100 %.

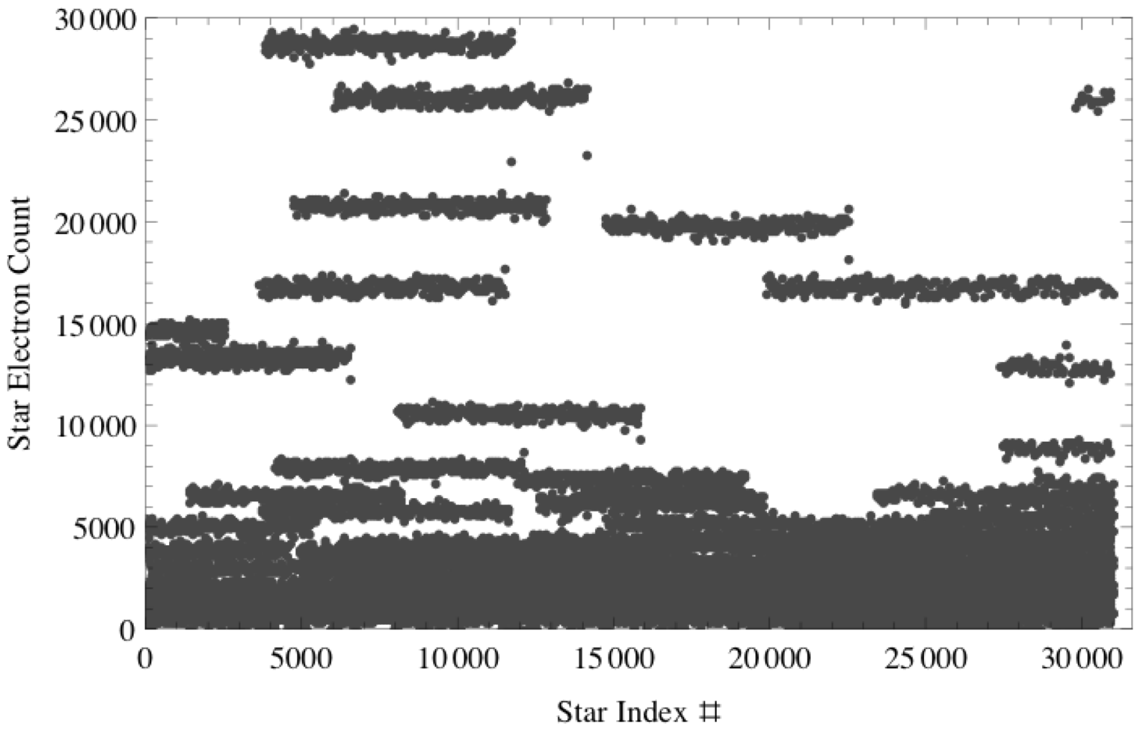

Figure 38 # of electrons per star, showing variation in star brightness

6.2.9 Catalog Preparation

To give Star-ND the best chance at success, the most physically-accurate star catalog was desired, and the process to produce such a catalog is identical to the process used by VST to simulate the stars, so the VST catalog was used directly, because this preparation already included the proper-motion and instrument-magnitude corrections, both highly desirable for the target precision of Star-ID. Planets were not included in the star catalog used to generate the Star-ND database. The results of the instrument magnitude correction are shown in Figure 39, which clearly shows streaks of stars, each with the same spectral type, and therefore a similar bolometric correction.

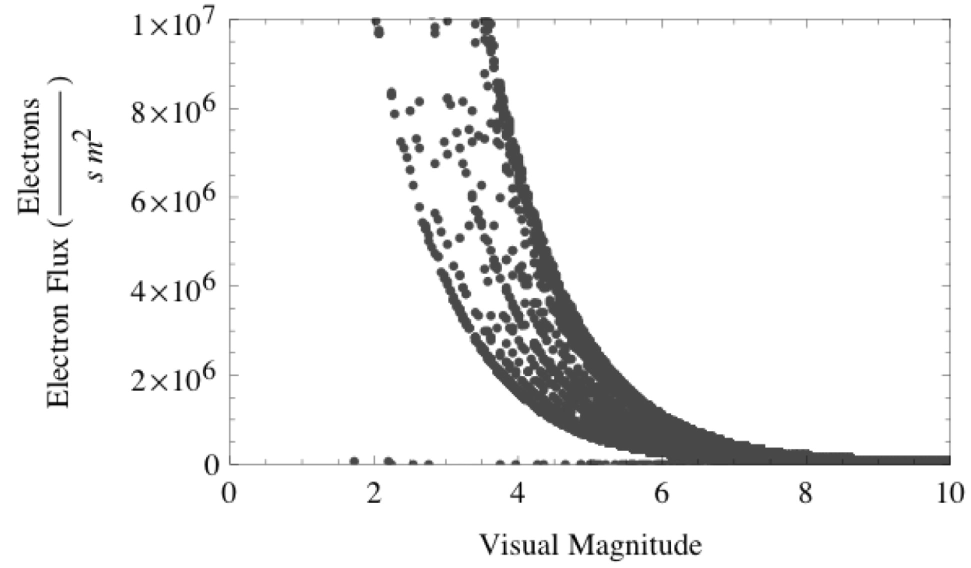

Figure 39 Instrument electron fluxes vs. visual magnitude

6.2.10 Machine Setup & Software

These performance tests were carried out on an Apple MacBook Pro (2009), 13" with a dual 2.53 GHz processor. The system's RAM memory bus operational frequency is 1067 MHz. The operating system is Mac OS X, 10.6.2. The Star-ND C source code was compiled with GCC 4.2 through XCode 3.2.1, Apple Inc.'s Integrated Development Environment. The code was written single-threaded, and the processor usage was monitored as the code executed to ensure that no more than 1 processor core was used during the code's execution. The compiler was set to "-O3" which means "most optimized." Under this setting, the compiler optimized the code for speed, even at the cost of increasing code size, meaning there are likely to be more in-lined functions.

Though the Star-ND code was written in pure C to optimize both performance and portability, the test application which loaded the image centroiding data and performed measurement and logging of performance results was primarily written in Objective-C in order to decrease development time and increase test setup flexibility. The test setup code is not included in this dissertation.

The UNIX system call "getrusage()" was used to obtain the process resource usage information from which user time and system time were obtained both before and after each test run. On the test platform, the time precision of the getrusage() call is 1 μs, which corresponds to about 2530 clock cycles. Since the algorithm's run-time can be as low as this value, each test case was repeated 10 times, and the total time divided by 10, resulting in a 0.1μs precision, or 253 clock cycles.

6.2.11 Simulate Image Results By Catalog Brightness Threshold

Because the Star-ND technique assumes that the stars that will be observed are above a given threshold, but the photometry in real star tracker images is not exact, the robustness (i.e. success rate) will depend on correctly matching the catalog brightness threshold with the camera noise sources and image processing settings. If the threshold used to select the subcatalog is off by several tenths of a magnitude, the success rate diminishes significantly. The simulated image tests are intended to show the sensitivity of the Star-ND algorithm to the brightness measurement of the stars in an image. To test the exact nature of this fall-off, the success rate of the Star-ND algorithm is measured across forty-five different catalog thresholds, while keeping the image processing settings the same. In Figure 40, the catalog threshold is decreased by a twentieth of a magnitude between each test case.

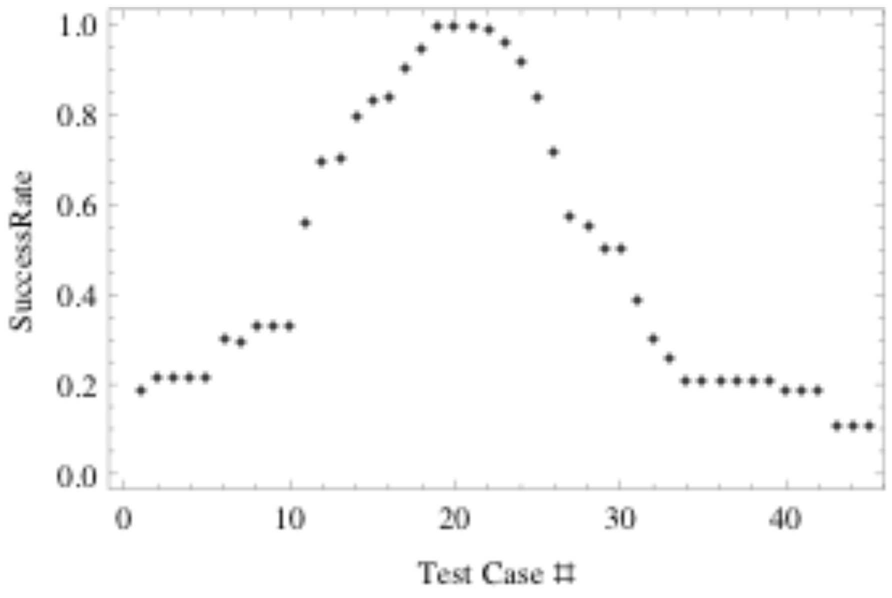

Figure 40 Star-ND success rate vs. catalog brightness threshold

The results show Star-ND achieves 100% success rate in test case 20. This result is consistent with the expectation that the brightness threshold must be fairly accurate for the star pattern to be selected correctly.

6.2.12 Selected Simulated Image Results

It is assumed that in the case in which 100% success rate was achieved, the catalog brightness threshold matched the image brightness threshold. A more detailed view of these test cases is presented in Figure 41. These results show that the execution time generally clumps in the 4,000 to 15,000 clock cycle region, while outliers range up to 65,000 clock cycles. This discrepancy between average and maximum execution time is expected based on the false-star test above.

Further analysis shows a heavy dependence of the execution time on the number of iterations performed, shown in Figure 42, which is the expected behavior. The test results show up to 6 iterations of central star selection, which means in many of the test cases there were many dim stars peeking above the threshold. To determine if the K-Vector ND has actually reduced significantly the number of items returned by the database search, the number of found database entries (*k*) is plotted in Figure 43. The plot shows the maximum number of found entries in the K-Vector ND to be 6, a very low number, considering there are about 5000 stars in the database. The Star-ND parameters have narrowed the identity of the stars to fewer than 0.1%. In contrast to the SLA, in which each database search may find as many as 1.5% of the entries.

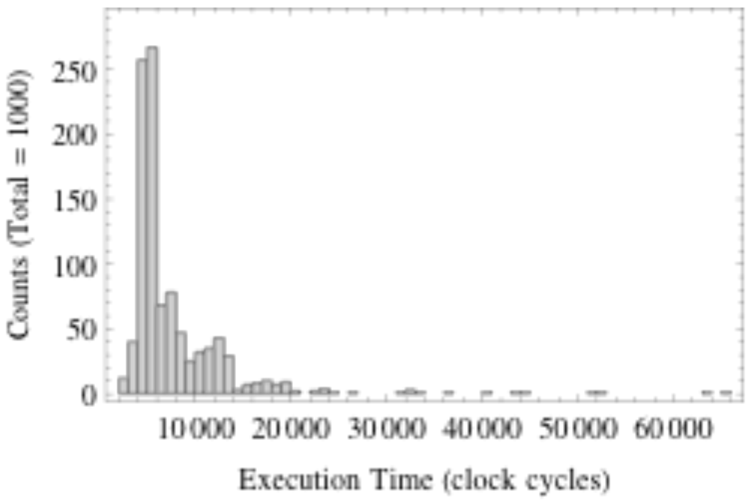

Figure 41 Selected simulated image test results histogram

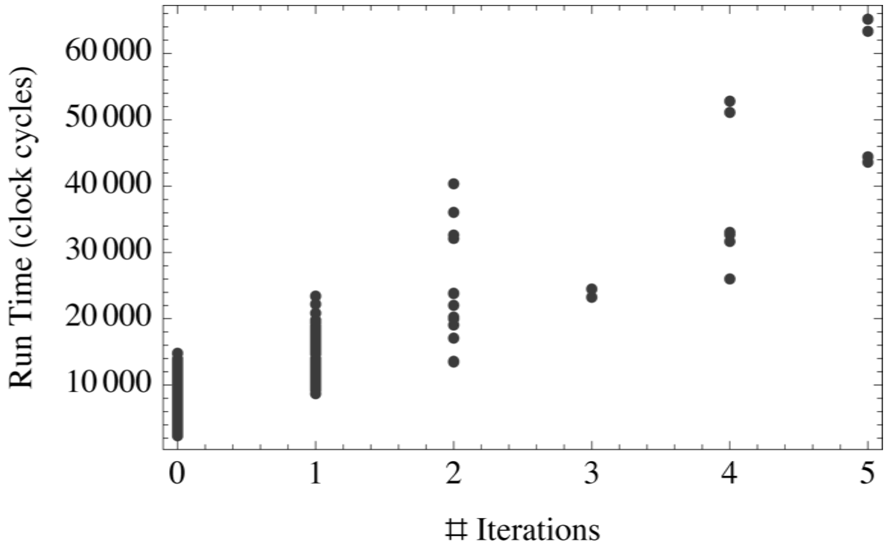

Figure 42 Star-ND execution time vs. # of iterations

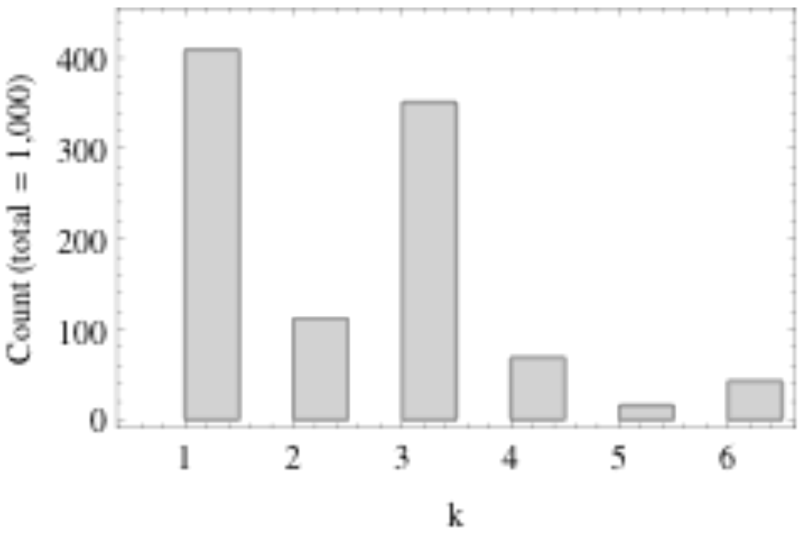

Figure 43 Number of found entries in Star-ND database

6.2.13 Comparison to Pyramid

Pyramid and its variants are the current state-of-the-art algorithm for both robustness and performance [1]. As a benchmark to Star-ND, the results of the Star-ND tests are compared to Pyramid in Figure 44. The test uses the same camera and star catalogs, which is not necessarily the best performance settings for PYRAMID. The results show that the Star-ND algorithm varies in its execution time from 1/100th to equal to the Pyramid execution time, for an average of a factor of 25 times improvement over Pyramid.

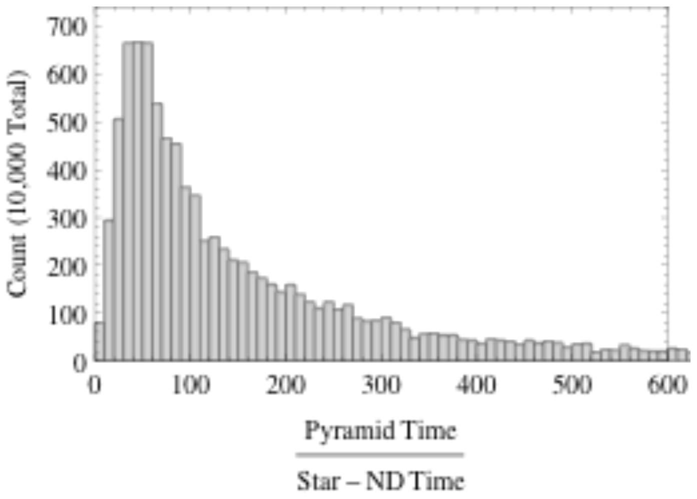

Figure 44 Pyramid execution time divided by Star-ND execution time

Furthermore, the performance of the two algorithsm in the presence of many non-stars is compared for identical test cases as above. The results of this false star trial are presented in Figure 45.

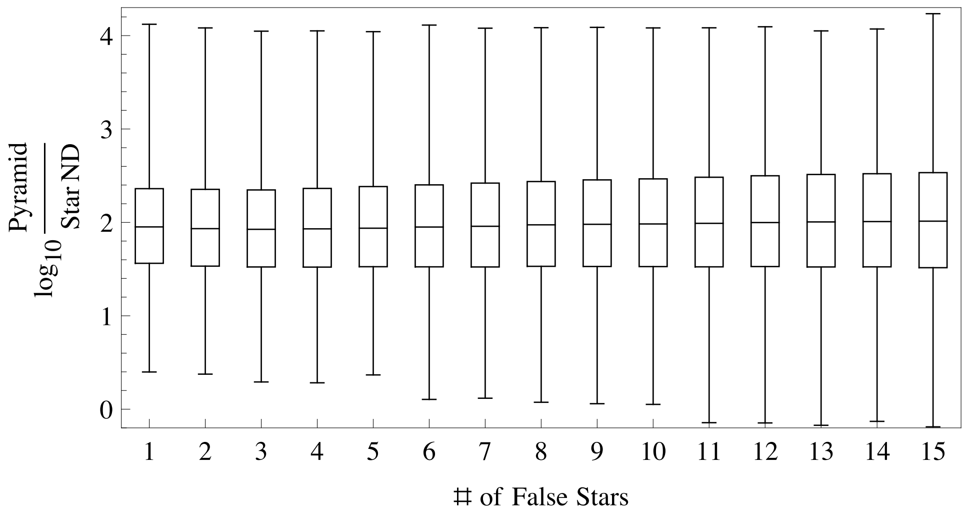

Figure 45 Comparison of pyramid and Star-ND execution times when including false stars

CHAPTER VII

GUIDELINES FOR STAR-TRACKER SUB-SYSTEM DESIGNERS

One of the issues associated with needing the 3 nearest stars is its tendency to increase the size of the required field of view when compared to methods that can use any combination of stars. There are two approaches to determine the required FOV size: theoretical and experimental.

7.1 Theoretically

Theoretically, a uniform distribution of stars in the sky would associate an equal area of the sky (which we may represent as a circle on a unit sphere) with each star. Finding 4 stars would then fill a circle with four times that spherical area. Finding the spherical angle associated with this area is shown in Eq. (11):

$$Area = 2\pi r^2 \int_0^\theta \sin\phi \, d\phi = 2\pi r^2 (1 - \cos\theta) \tag{11}$$

If this area is $1/n$ of the total $4\pi r^2$ area, the angle θ can be solved for simply:

$$\theta = \cos^{-1}\left(1 - \frac{2a}{n}\right) \tag{12}$$

where a is the number of stars in the circle and n is the number of stars in the sky. Doubling this θ values would yield a FOV size. Note in the integral the appearance of the $1 - \cos\theta$ expression that is the important expression in the star pattern parameter functions. Practically, it is impossible to construct such a uniform distribution with more than 12 stars, so the actual distribution of stars must be non-uniform in some manner.

7.2 Experimentally

Experimentally, the distribution of stars in the sky is not uniform due to the galactic plane, so the star patterns are formed and invalid star patterns found as described in section VII for a given FOV size. The number of invalid star patterns is then compared to the number of stars in the database, and plotted in Figure 46. Also, the theoretical curve, multiplied by a few constant factors (1.25, 2.5, and 4.1), has been plotted on the same graph, along with results from the Delaunay triangulation, which are represented as white dots.

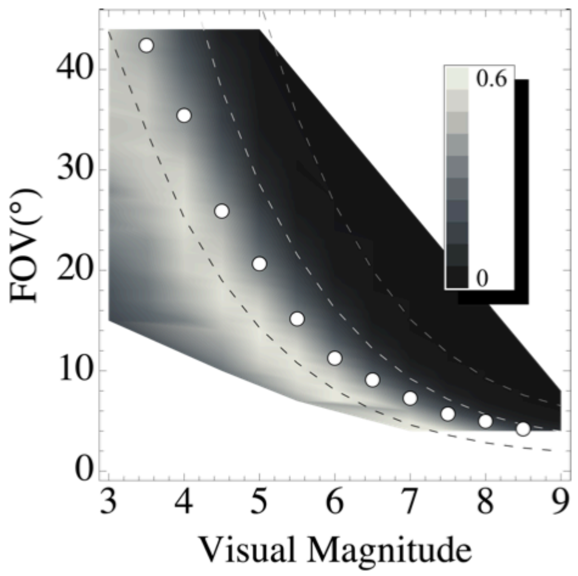

Figure 46 Various calculations for FOV size

There are four interesting results from this plot:

1. For a particular database size, the ratio of invalid star patterns increases as the FOV decreases, reaches a peak, and then decreases.

2. That maximum ratio of invalid star patterns for each database size is the same (0.6).

3. By multiplying the theoretical curve by a constant factor, it may be placed directly on the maximum ratio for all database sizes.

4. By multiplying the theoretical curve by a higher constant factor, it may be placed in a region with very few invalid star patterns, or none at all.

Why should the number of invalid star patterns decrease as the FOV shrinks? The code used to calculate these numbers checks to see if there are three nearby stars within a FOV's size of the selected star. If there are not, the star is not counted at all. As the FOV decreases, there are fewer and fewer cases in which the star is able to have three nearby stars within the FOV. Thus, designing a star tracker with a FOV smaller than the peak would mean frequently there are not four stars in the FOV, no matter the algorithm used to identify them.

7.3 Minimizing the FOV

A subsystem designer could compute the curves as given above to set the FOV for a particular star tracker, but the values for the 4.1× curve are unusually large. Statistically, it is impossible to guarantee that simply having four stars in the FOV guarantees that they are a star and its nearest three, so algorithms that consider any pair of stars operate correctly with smaller fields of view than the new Star-ND algorithm. The number of stars which need to be measured and sorted increases with $f = O(1 - \cos(\mathrm{FOV}/2))$ which, for small FOV is $O(\mathrm{FOV}^2)$. Sorting is $O(f \lg f)$. Shrinking the FOV thus produces a O(FOV^2 lg FOV) improvement in execution time, and a linear improvement in angular accuracy, so it is desirable to attempt to modify the algorithm to reduce the required FOV.

The way to do so is to modify the rules in the previous sections such that 1) a threshold, t, for the maximum number of false stars is set, and 2) if there are fewer than ($p + t$) stars inside the max(θ(p−1)) limit, more stars are sorted to find the nearest stars. By doing so, it is implied that the potentially invalid patterns have also been duplicated with further-than-max(θ(p−1))-stars added. Only a few outliers push the boundary of the max(θ(p−1)) limit, meaning that by inserting only a few extra cases to account for these sparse stars can significantly reduce the max(θ(p−1)) limit itself, since the original patterns can be excluded from the calculation of the max(θ(p−1)) limit and considered only once the new lower limit has been reached unsuccessfully. Note also that we have been required to add a threshold for the maximum number of false stars in order to begin this process, and that may be undesirable in some cases. It is up to the sub-system designer to choose how much of a decrease in FOV and increase of database size are needed. By using this expansion, it is not the case that we loose all of the performance advantage of only sorting the stars within the max(θ(p−1)) radius limit. A new maximum radius for the known replacements can be generated. Only the stars within this new outer limit need to be sorted for the expanded case.

Practically, the smallest FOV size may be found by iterating the database construction until the smallest FOV size with no potentially invalid star patterns which have no substitutions is found. An upper bound for the FOV size is the Delaunay largest empty circle diameter plus twice the max(θ(p−1)) radius. Clearly the FOV could not be effective if it were smaller than the Delaunay diameter. For instance, a binary iteration between these sizes to shrink the FOV to whatever precision I request, usually around a tenth of a degree. For each iteration, the FOV is increased if there are potentially invalid star patterns thathave no substitutes and shrunk if there are none. The amount by which the FOV is changed is half of the previous step, and the initial size is the prediction of the theoretical curve, and the initial step size is half of the difference of the maximum and minimum sizes. The number of iteration steps is lg[(max FOV − min FOV)/(2 precision)].

CHAPTER VIII

CONCLUSIONS

8.1 Summary

Using an inherently less-robust star pattern, a new set of star pattern parameters allows calculating directly, through the K-Vector ND, the identity of a star from angles-only measurements. To increase the robustness and decrease the execution-time, expected errors are used to add "false" entries to the star database. The result is an asymptotically faster algorithm, which, in one case, is a factor of 25 faster than current cutting-edge SLA-based algorithms. Furthermore, its database grows linearly with the number of stars in the catalog, asymptotically smaller than all-combination star patterns.

8.2 Directions for Future Research

There are some interesting extensions to this research, which are nearly all performance enhancements, judged to be slightly outside the scope of this dissertation. Given sufficient time & funding, they would quite certainly have been implemented.

8.2.1 Improving Performance with Parallelization

The dominant term in the run-time of the Star-ND algorithm is sorting. In order to select the nearest stars, a sort on the stars must be performed. It is well-known that sorting using a comparison-based sort takes O(*n* lg *n*)-time, and the various linear-time sort algorithms are inappropriate. However, if a parallel platform were developed to perform Star-ND, sorting could be performed in O(n)-time, given sufficient hardware. Though it is possible to sort in O(lg n)-time in a hardware implementation, the centroid information is unlikely to be provided in constant time, rather in linear time. Thus, an efficient parallelized sort would perform like an insertion sort, with direct links between memory cells, each large enough to hold the appropriate vector information. Thus, instead of O(n)-copies of data, O(1) copy-times would allow the total algorithm to run in O(n). This implementation requires O(n)-hardware, however, though no more memory than would be required by the comparison-sort.

Also, taking dot products from a given star to the others must be performed at least twice for each star-ID. If each vector memory location were augmented with a dot product circuit, taking the dot products of n items would take O(1)-time, instead of O(n)-time. (It is assumed that simple math operations use O(1)-time, though in a hardware implementation this is no longer the "correct" assumption.) The subsequent sort operation could run in O(lg n)-time, since all of the data would become available simultaneously.

Finally, each star pattern parameter may be calculated at the same time. Only the four star vectors are required, from which the appropriate dot products may be calculated simultaneously in parallel hardware. Next, the five (1-sqrt()) calculations could be performed simultaneously, and then the six limits could be calculated at the same time. From the processor's point of view, this would significantly reduce the time required for the star pattern parameter bound calculation, but it should be noted that these operations currently require no code pipeline flow changes, an expensive operation that currently dominates the sorting algorithm time.

The K-Vector ND is much harder to parallelize, since the memory access is assumed not to be parallel. The increments that are applied to the row locations could be updated while data from the current row is being accessed.

The filter step which checks all six inter-star angles could also be parallelized, though it is assumed that this data is entered in a serial fashion-which reduces the effectiveness of the parallel filter. Instead it is likely that this data is checked as it is copied in from memory, and that once a failure on one value has happened, the remaining values do not need to be copied in from memory.

8.2.2 Resolving Double-Stars in Star-ID

A "double-star" is actually two stars whose point-spread functions have met such that the centroiding algorithm is unable to distinguish them. The traditional step in the attitude determination process to resolve these double-stars is typically image processing. Expensive image-processing algorithms can effectively resolve a double-star illuminated area into two separate stars. [57]

A straightforward practice for handling double-stars is to find these cases when building the star-database and combine them into one catalog entry, or to remove the star from the catalog entirely [58]. However, there are cases in which the rotation of the camera about its boresight can alternately join stars cataloged as separated, and separate stars cataloged as joined. This effect is caused by the rectangular pixels, which have a different size depending on whether they are measured transversely or diagonally. Thus there is a gray area in star separation in which stars may be observed both as connected and separated depending on the unknown orientation of the camera.

Consider two stars are near each other, each with its own Gaussian light distribution. The CCD will produce a summation of their light. There is an assumed noise level for the CCD, and higher still a cutoff for pixels deemed to be bright enough to be used in centroiding. There are two limits that determine if this gap is seen in the resulting pixels.

For an algorithm that is built on identifying any possible pair of stars, the solution to these unusual cases is simple: ignore them and identify others. [59-68] They happen rarely enough that they are neglected in the simulated image tests presented in this dissertation. However, as photometry precision decreases and catalog sizes increase, the number of these cases is non-zero, and they could potentially cause a possible failure of the star-ID algorithm, or at least an extra iteration. Avoiding unnecessary failures and iterations is desirable.

The concept of inserting false catalog entries is an attractive option for identifying these stars. Since mathematical models can predict which star pairs may be seen both as connected and separate, the separate cases may remain in the regular section of the star catalog, but an additional "possible double star" section can be added to the end of the catalog. Corresponding false star pattern entries would be added to the database. Identification of these stars would happen normally, and at the end, if one of these cases were identified, the run-time algorithm would know that it is a double-star by the location of the star in the catalog. If so desired, an image-re-processing algorithm could then attempt to distinguish the stars.

In fact, the resolution can occur during the recursive algorithm, as well. Both the separated stars and the double star are in the catalog, and can be separated using the initial attitude estimate. In general, the accuracy of a given centroid is 1/20th of a pixel or better. The angular separation between the stars is typically close to twice the Gaussian blur size, between 3 to 10 pixels. Therefore the stars are usually separated by 60-200 times the error in estimation, and should be easily distinguished once an attitude estimate is available.

This technique can be applied to any generalized Star-ID algorithm (with one exception), since it does not rely on the math or flow control used in identifying the original star pattern, only that an additional false entry be added to the catalog in a controlled manner. The exception is any star-ID technique based on a unique graph of inter-star relationships, as the addition of false entries would create a duaility in the graph, though such a dual entry could theoretically be grafted into the graph along an invariant perimeter.

8.2.3 Planets, Satellites, and Variable Stars

The presence of variable stars and non-stars impacts the Star-ND technique in a powerful way. When a new light source appears, or the brightness of an existing light source dips below the brightness threshold, the star patterns change and for the Star-ND technique to be useful, the database must be altered. While it is a simple matter to store the new catalog entry at some new location in memory, the K-Vector ND must be rebuilt to accommodate the change. This is unfortunately an O(*d n*)-time procedure, because the “bins” are stored consecutively. This arrangement allows the K-Vector ND to search rows very quickly (and to instantly skip over empty bins), and is therefore the algorithm tested in this dissertation. However, if the bins were not located in consecutive memory locations (by the inclusion of bin length data, or pointers to the end of each bin) such a mutable K-Vector ND could easily have items removed or added. The required data size would increase and the range-search time would increase, however, the process of adding or removing a few entries to the database would be asymptotically faster. One such arrangement is depicted in Figure 47.

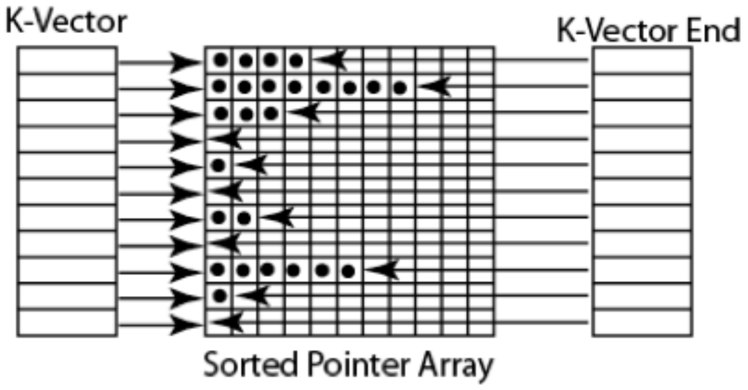

Figure 47 Sample implementation of a mutable K-Vector ND

8.2.4 Non-Linear Search Bounds & Pre-searched K-Vector ND

The rectangular search bounds presented in this dissertation make an important but inaccurate assumption about the information flow of the highly-non-linear star pattern parameters, which is noted as the “not entirely consistent, but nonetheless useful” star pattern error bounds formulation of the error bounds. Since the K-Vector ND is designed to work as fast as possible by making the orthogonal range assumption, it is useless to try to calculate the more accurate non-linear error bounds. However, non-linear error bounds calculation would specify fewer entries in the database. A more accurate error bounds analysis would calculate directly the non-linear error bounds, reduce the number of items retrieved, and reduce the time spent in the filter step.

Furthermore, these error bounds are actually “backwards.” The values calculated are “what range of values might be observed from this value?” whereas the true question sought by the search algorithm is “what original values might have been observed as this value?” The difference is actually quite stark when looking at the way the linearized error bounds are calculated as the star pattern parameter increases. From a perspective based on the principal error components, it is clear that variations in error bounds should increase as the parameter increases from 0 to 1, but then decrease from 1 to 2. However, the variation from the min/max technique continues increasing. Thus, star patterns near a value of 2 receive much wider error bounds than they should, resulting in more found entries from the database than they should.

A potential solution for both problems is the same: the pre-searched K-Vector ND, illustrated in Figure 48. The K-Vector ND is like the K-Vector ND in that the data range is evenly divided into bins, however, instead of placing an entry in the bin that intersects its value, the pre-searched K-Vector ND places the entry in every bin in which it might be observed. This means that not only are the non-linear error formulations generating smaller search ranges, but the flow is the correct direction. To search this "pre-searched" K-Vector ND, one does not need to compute error bounds; that has already been done. One simply needs to look in the bin corresponding to the measured values. All of the entries that might take on the measured values are already listed in that one location. The draw back of this method is a much larger K-Vector ND database size, and significantly larger database build-time.

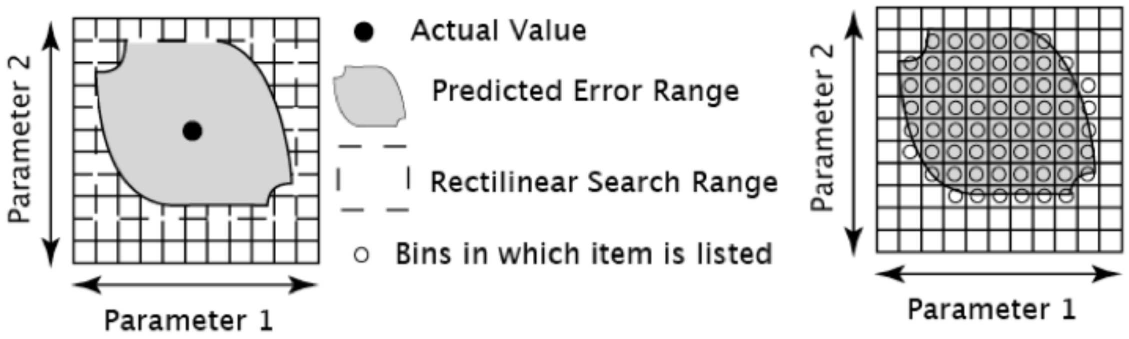

Figure 48 Illustration of non-linear search region for K-Vector ND

8.2.5 The Search for a Fully-independent Parameter Set

The number of independent parameters for a give star triangle is 3, and for two adjacent triangles, 5. This dissertation introduces only 3 parameters for two adjacent triangles, which means there may be a different set of functions that fully harnesses the independent information in the star pattern. As independent functions, such functions would theoretically be well distributed, but the data set is constrained by the existing stellar data.

REFERENCES

1. MORTARI, D., SAMAAN, M.A., and BRUCCOLERI, C. "The Pyramid Star Identification Technique," *Navigation,* Vol. 51*,* 2004, 171-183.

2. GOTTLIEB, D. M. *Spacecraft Attitude Determination and Control*. Wertz, J.R. Ed. Microcosm, Inc., El Segundo, CA, 1978.

3. BEZOOIJEN, R. W.H. V. *Automated Star Pattern Recognition*. Ph.D. Thesis, Stanford University, Palo Alto, CA, 1989.

4. SAMAAN, M.A., MORTARI, D., and JUNKINS, J.L. "Nondimensional Star Identification for Uncalibrated Star Cameras," *Journal of the Astronautical Sciences*, Vol. 54, January-March 2006, 95-111.

5. CORMEN, T. H., LEISSERSON, C. E., RIVET, R. L., and STEIN, C. *Introduction to Algorithms*. 2nd Ed., McGrw-Hill, Boston, MA, 2003.

6. SALOMON, P.H. "A Microprocessor Controlled CCD Star Tracker," AIAA 14th Aerospace Sciences Meeting. AIAA, Washington, D.C., 1976.

7. JUNKINS, J.L., WHITE, C., and TURNER, J. "Star Pattern Recognition for Real-time Attitude Determination," *Journal of the Astronautical Sciences,* Vol. 25, July-September 1977, 251-270.

8. JUNKINS, J.L. and STRIKWERDA, T.E. "Autonomous Star Sensing and Attitude Estimation," *Proceedings of the Annual Rocky Mountain Guidance and Control Conference*, February 1979, number 79-013.

9. STRIKWERDA, T. E. and JUNKINS, J. L. "Star Pattern Recognition and Spacecraft Attitude Determination," Technical Report ETL-0260, U.S. Army Engineer Topographical Laboratories, Fort Belvoir, VA, 1981.

10. GROTH, E.J. "A Pattern Matching Algorithm for Two-dimensional Coordinates Lists," *Astronomical Journal*, Vol. 91, 1986, 1244-1248.

11. SASKI, T. "A Star Identification Method for Satellite Attitude Determination Using Star Sensors," *Proceedings of the 15th International Symposisum on Space Technology and Sciences*, Tokyo, Japan, May 1986, 1125-1130.

12. ANDERSON, D. *Autonomous Star Sensing and Pattern Recognition for Spacecraft Attitude Determination*. Ph.D. Thesis, Texas A&M University, College Station, May 1991.

13. LIEBE, C.C. "Pattern Recognition of Star Constellations for Spacecraft Applications," *IEEE Aeronautics and Electronic Systems Magazine,* Vol. 10, June 1992, 2-12.

14. BALDINI, D., BARNI, M., FOGGI, A., BENELLI, G., and MECOCCI, A. "A New Star-Constellation Matching Algorithm for Satellite Attitude Determination," *ESA Journal*, Vol. 17, 1993, 185-198.

15. KETCHUM, E. A. and TOLSON, R. H. "Onboard Star Identification Without a Priori Attitude Information," *Journal of Guidance, Control and Dynamics*, Vol. 18, March-April 1995, 242-246.

16. SCHOLL, M.S. "Star-field Identification for Autonomous Attitude Determination." *Journal of Guidance, Control and Dynamics*, Vol. 18, January-February 1995, 61-65.

17. QUINE, B.M. and WHYTE, H.F.D. "A Fast Autonomous Star-Acquisition Algorithm for Spacecraft." *Control Engineering Practice*, Vol. 4, May 1996, 1735-1740.

18. PADGETT, C. and DELGADO, K.K. "A Grid Algorithm for Autonomous Star Identification." *IEEE Transactions on Aerospace and Electronic System,* Vol. 33, January 1997, 202-213.

19. MORTARI, D. "A Fast On-board Autonomous Attitude Determination System Based on a New Star-ID Technique for a Wide FOV Star Tracker," *Advances in the Astronautical Sciences*, Vol. 93, 1996, 893-903.

20. MORTARI, D. "Search-less Algorithm for Star Pattern Recognition," *Journal of the Astronautical Sciences,* Vol. 45, April-June 1997, 179-194.

21. MORTARI, D. “K-vector Range Searching Techniques.” *Advances in the Astronautical Sciences*, Vol. 105, 2000, 449-464.

22. SOLAIAPPAN, A., PANDIYAN, R., RAMACHANDRAN, M., and VIGHHNESAM, N. “Attitude Determination Using an Experimental Fast Recovery Star Sensor (FRSS) for a Geostationary Spacecraft,” 2nd International Astronautical Congress. Flight Dynamics Division, ISRO Satellite Centre, Bangalore, India, October 2001.

23. BRADY, T., TILLIER, C., BROWN, R., JIMENEZ, A., and KOUREPENIS, A. “The Inertial Stellar Compass: A New Direction in Spacecraft Attitude Determination,” 16th Annual USU Conference on Small Satellites, Logan UT, 2002.

24. CREW, G. B., VANDERSPEK, R., and DOTY, J. “Hete Experience with the Pyramid Algorithm,” Technical Report 02139, MIT Center for Space Research, Cambridge, MA, 2002.

25. ALVEDA, P. and SAN MARTIN, A.M. “Neural Network Star Pattern Recognition of Spacecraft Attitude Determination and Control,” *Advances in Neural Information Processing System I.* Denver, CO, 1989, 213-322.

26. HONG, J. and DICKERSON, J.A. “Neural-network-based Autonomous Star Identification Algorithm,” *Journal of Guidance, Control and Dynamics*, Vol. 23, August 2000, 728-735.

27. GUANGJUN, Z., WEI, X., and JIANG, J. “Full-sky Autonomous Star Identification based on Radial and Cyclic Features of Star Pattern,” *Image and Vision Computing*, Vol. 26, 2008, 891-897.

28. KOLOMENKIN, M., POLLAK, S., SHIMSHONI, I., and LINDENBAUM, M. “Geometric Voting Algorithm for Star Trackers,” *IEEE Transactions on Aerospace and Electronic Systems,* Vol. 44*,* April 2008, 441-456.

29. AU ROUSSEAU, G.L., BOSTEL, J., and MARZARI, B. “Star Recognition Algorithm for APS Star Tracker: Oriented Triangles,” *IEEE Aerospace and Electronic Systems Magazine,* February 2005, 27-31.

30. SAMAAN, M.A., MORTARI, D., and JUNKINS, J.L. “Recursive Mode Star Identification Algorithms.” *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 41, October 2005, 1246-1245.

31. UDOMKESMALEE, S., ALEXANDER, J.W., and TOLIVAR, A.F. “Stochastic Star Identification,” *Journal of Guidance, Control and Dynamics*, Vol. 17, November-December 1994, 1283-1286.

32. PARISH, J.J., PARISH, A.S., SWANZY, M., WOODBURY, D., MORTARI D., and JUNKINS J.L. “Stellar Positioning System (Part I): Applying Ancient Theory to a Modern World,” AIAA/AAS Astrodynamics Specialist Conference, Honolulu, HI, August 2008.

33. WOODBURY, D., PARISH, J.J., PARISH A.S., SWANZY, M., MORTARI, D., and JUNKINS, J.L. “Stellar Positioning System (Part II): Overcoming Error During Implementation,” AIAA/AAS Astrodynamics Specialist Conference, Honolulu, HI, August 2008.

34. ZILIANG, W. and QUAN W. “An All-Sky Autonomous Star Map Identification Algorithm,” *IEEE A&E Systems Magazine*, March 2004, 10-14.

35. BENTLEY, J.L. and FRIEDMAN, J.H. “Data Structures for Range Searching,” *Computing Surveys*, Vol. 111, No. 4, 1979, 254-275.

36. ALSTRUP, S., BRODAL, G. S., and RAUHE, T., “New Data Structures for Orthogonal Range Searching,” *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, 2000, 198-207.

37. SUBRAMANIAN, S. and RAMASWANNY, S. “The P-Range Tree: A New Data Structure for Range Searching in Secondary Memory,” *Proceedings of the 6th Annual Symposium on Discrete Algorithms*, ACM Press, 1995, 378-387.

38. CHAZELLE, B. “Lower Bounds of Orthogonal Range Searching: I. The Reporting Case,” *Journal of the ACM*, Vol. 37, April 1990, 200-212.

39. CHAZELLE, B. “Lower Bounds of Orthgonal Range Searching: II. The Arithmetic Model,” *Journal of the ACM*, Vol. 37, June 1990, 439-463.

40. LUEKER, G.S. "A Data Structure for Orthogonal Range Queries," *Proceedings of the 19th Annual IEEE Symposium on the Foundations of Computer Sciences*, Ann Arbor, MI, October, 1978, 28-34.

41. MEHLHORN, K. "Data Structure and Algorithms 3," Vol. 3 of *Monographs in Theoretical Compuer Science*. An EATCS Series, Springer, New York, NY, 1984.

42. BENTLEY, J.L. "Multidimensional Binary Search Trees Used for Associative Searching," *Journal of the ACM*, Vol. 18, No. 9, 1975, 509-517.

43. AGARWAL, P. K. *Handbook of Discrete and Computational Geometry*, CRC Press, Boca Raton, FL, 1997, 575-598.

44. PREPARATA, F. P. and SHAMOS, M. I. *Computational Geometry*, Springer-Verlag, New York, 1985.

45. SPRATLING, B. B. and MORTARI, D., "The K-Vector ND and Its Application to Building a Non-dimensional Star-ID Catalog," presented as paper AAS 09-126 at the 19th AAS/AIAA Space Flight Mechanics Meeting Conference, Savannah, GA, February 8-12, 2009.

46. MORTARI, D. and NETA, B. "K-vector Range Searching Techniques," *Advances in the Astronautical Sciences*, Vol. 105, 2000, 449-464.

47. DELAUNAY, B. "Sur la Sphère Vide," Bull. Acad. Science USSR VI: Class. Sci. Math., 1934, 793-800.

48. SPRATLING, B. B. and MORTARI, D. "Star-ND Multidimensional Star Identification," Submitted to *AIAA Journal of Guidance, Control and Dynamics.*

49. KUMAR, M., MORTARI, D., and JUNKINS, J.L. "An Analytical Approach to Star Identification Reliability," *Acta Astronautica* Vol. 66, 2010, 508-515.

50. SAMAAN, M.A., BRUCCOLERI, C., MORTARI, D., and JUNKINS, J.L. "Novel Techniques for the Creation of a Uniform Star Catalog," presented as paper 03-609 at the AAS/AIAA Astrodynamics Specialist Conference. Big Sky, MT, August 2003.

51. PERRYMAN, M.A.C., LINDEGREN, L., KOVALEVSKY, J., HOG, E., BASTIA, U., BERNACCA, P.L., CREZE, M., DONATI, F., GRENON, M., GREWING, M., VAN LEEUWEN, F., VAN DER MAREL, H., MIGNARD, F., MURRAY, C.A., LE POOLE, R.S., SCHRIJVER, H., TURON, C., ARENOU, F., FROESCHLE, M., and PETERSON, C.S., "The Hipparcos Catalogue," *Astronomy and Astrophyics*, Vol. 323, July 1997, 49-52.

52. TIPLER, P. A. and LLEWELLYN, R. A. *Modern Physics*. 3rd Ed. W. H. Freeman and Company, New York, 1999.

53. SCHAUB, H. AND JUNKINS, J. L. *Analytical Mechanics of Space Systems*. AIAA, Reston, VA, 2003.

54. KURUCZ, R. L. The Kurucz (1992) Model Atmospheres. URL http://www.stsci.edu/science/starburst/Kurucz.html. Accessed on June, 2009.

55. PENCE, W. D. "FITS Documentation Page" URL http://fits.gsfc.nasa.gov/fits_documentation.html, Accessed on June, 2009.

56. BERRY, R. and BURNELL, J. *The Handbook of Astronomical Image Processing*. 2nd Ed. Willmann-Bell, Inc., Richmond, VA, 2009.

57. MORTARI, D., POLLOCK, T.C., and JUNKINS, J.L. "Towards the Most Accurate Attitude Determination System Using Star Trackers," *Advances in the Astronautical Sciences*, Vol. 99*,* 1998, 839-850.

58. SMITH, N., BAE, S., and SCHUTZ, B. "Forty-Nine Biased Star Positions from ICESat Flight Data," presented as paper AAS 10-205 at the 20th AAS/AIAA Space Flight Mechanics Meeting, 14-17 February 2010, San Diego, CA.

59. SAMAAN, M.A. *Research on Multiple FOVs Star Sensor Data Processing*. Ph.D. Thesis, Texas A&M University, College Station, June 2003.

60. KATAKE, A.B., OCHOA, J.O., ZBRANEK, J., DAY, B., BRUCCOLERI, C., and GOODSELL, D. "Development and Testing of the Starcam SG100: A Stellar Gyroscope," presented as paper 2008-6650 at the AIAA Guidance and Control Conference Exhibit. AIAA, Honolulu, HI, August 2008.

61. JUNKINS, J.L., MORTARI, D., POLLOCK, T.C., BOYLE, D., CARRON, I., ABDELKHALIK, O.O., ETTOUATI, I., HILL, C., and CANTRELL, J. "Feasibility Study and System Concept Development for the Space Situational Awareness Cameras System." Contract SC-03A-22-08, Shafer, October 2005.

62. ETTOUATI, I., MORTARI, D., and POLLOCK, T.C. "Space Surveillance with star trackers. Part I: Simulation," presented as paper AAS 06-232 at the AAS/AIAA Space Flight Mechanics Meeting,Tampa, FL, January 2006.

63. ABDELKHALIK, O. O., MORTARI, D., and JUNKINS, J.L. "Space Surveillance with Star Trackers. Part II: Orbit Estimation," presented as paper AAS 06-232 at the AAS/AIAA Space Flight Mechanics Meeting, Tampa, FL, January 2006.

64. MORTARI, D. "Planet and Time Estimation Using Star Trackers," presented as paper AAS/AIAA 06-218 at the AAS Space Flight Mechanics Meeting, Tampa, FL, January 2006.

65. KARIMI, R.R. and MORTARI, D. "Designing an Interplanetary Autonomous Navigation System (IANS) using Visible Planets," presented as paper AAS 09-160 at the AAS/AIAA Space Flight Mechanics Meeting, Savannah, GA, February 2009.

66. JUNKINS, J.L., POLLOCK, T.C., and MORTARI, D. Multiple Field of View Optical Imaging System and Method. U.S. Patent Pending No. 60/239,559, January 2001.

67. MORTARI, D. and ANGELUCCI, M. "Star Pattern Recognition and Mirror Assembly Misalignment for Digistar II and III: Multiple FOVs Star Sensors," *Advances in the Astronautical Sciences,* Vol. 102, February 1999, 1175-1184.

68. MORTARI, D. and ROMOLI, A. "Navstar III. A Three Fields of View Star Tracker," IEEE Aerospace Conference, Big Sky, MT, March 2002.

VITA

Benjamin Barnett Spratling IV received his Bachelor of Science degree in physics from Auburn University in 2004, having completed his Honors thesis in variable-density groundwater flow. He entered the Aerospace Engineering program at Texas A&M University in August 2005. His research interests include spacecraft system engineering, manned spaceflight, iPhone apps, and computer-based automated engineering techniques. Spratling and Dr. Mortari won the Best Paper Award for the AAS/AIAA 2009 San Diego conference presentation of the core Star-ND parameter equations. Spratling plans to co-author a book on stellar navigation with Dr. Mortari.

Mr. Spratling may be reached at his email address, me@benspratling.com, or by mail at 2320 Cahaba Rd., Birmingham, AL 35223.