1-1-2013

Analysis of Star Identification Algorithms due to Uncompensated Spatial Distortion

Steven Paul Brätt
Utah State University

ANALYSIS OF STAR IDENTIFICATION ALGORITHMS DUE TO UNCOMPENSATED SPATIAL DISTORTION

by

Steven P. Brätt

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Mechanical Engineering

Approved:

Dr. R. Rees Fullmer Major Professor	Dr. Charles M. Swenson Committee Member
Dr. David Geller Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2013

ABSTRACT

Analysis of Star Identification Algorithms

due to Uncompensated Spatial Distortion

by

Steven Paul Brätt, Master of Science

Utah State University, 2013

Major Professor: Dr. Rees Fullmer
Department: Mechanical and Aerospace Engineering

With the evolution of spacecraft systems, we see the growing need for smaller, more affordable, and robust spacecrafts that can be jettisoned with ease and sent to sites to perform a myriad of operations that a larger craft would prohibit, or that can be quickly manipulated from performing one task into another. The developing requirements have led to the creation of CubeSats. The question then remains, how to navigate the expanse of space with such a minute spacecraft? A solution to this is using the stars themselves as a means of navigation. This can be accomplished by measuring the angular separation between illuminated pixels in a camera image and associating the pixels with a corresponding star. Once identified, the spacecraft can obtain a quaternion solution to pinpoint its position and facing. A series of star identification algorithms called Lost in Space Algorithms (LISAs) are used to identify these pixels as stars in an image and assess the accuracy and probability of error associated with each algorithm. This is done by creating various images from a simulated camera program, using MATLAB as the program interface, along with images of actual stars in the night sky containing uncompensated error data taken with an Aptina camera. It is shown how suitable these algorithms are for use in space navigation, what constraints and impediments each have, and if low quality imagers can be used to determine attitude using these LISA's.

(221 pages)

PUBLIC ABSTRACT

Analysis of Star Identification Algorithms

due to Uncompensated Spatial Distortion

by

Steven Paul Brätt, Master of Science

Utah State University, 2013

Major Professor: Dr. Rees Fullmer
Department: Mechanical and Aerospace Engineering

With the evolution of spacecraft systems, we see the growing need for smaller, more affordable, and robust spacecrafts that can be jettisoned with ease and sent to sites to perform a myriad of operations that a larger craft would prohibit, or that can be quickly manipulated from performing one task into another. The developing requirements have led to the creation of Nano-Satellites, or CubeSats. The question then remains, how to navigate the expanse of space with such a minute spacecraft? A solution to this is using the stars themselves as a means of navigation. This can be accomplished by measuring the distance between stars in a camera image and determining the stars' identities. Once identified, the spacecraft can obtain its position and facing. A series of star identification algorithms called Lost in Space Algorithms (LISAs) are used to recognize the stars in an image and assess the accuracy and error associated with each algorithm. This is done by creating various images from a simulated camera, using a program called MATLAB, along with images of actual stars with uncompensated errors. It is shown how suitable these algorithms are for use in space navigation, what constraints and impediments each have, and if low quality cameras using these algorithms can solve the Lost in Space problem.

(221 pages)

ACKNOWLEDGMENTS

There have been many people who have made a great impact on my life and have helped me achieve my goals and pushed me to excel. To my wonderful professor, Dr. Rees Fullmer, I'd like to thank you for your constant support and input as I've worked to complete this thesis and finish my master's program. You have helped me improve dramatically in my abilities as a researcher and engineer. I'd like to thank your wife who was kind enough to let you spend so much of your time helping me.

To David Fowler, a good friend and great co-worker; thank you for your help and friendship as we worked together on both our research theses. You've helped me understand much more than you know.

I would like to say a special word of gratitude to my marvelous parents, Glenn and Odile, and my family who were always there to encourage me to pursue my dreams and ambitions. Thank you for your love, your prayers, and for the person you have helped me to become.

Steven Paul Brätt

CONTENTS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Catalog	=	An ordered database comprised primarily of stars, positions in Earth Central Inertial space, and magnitude intensity.
Feature	=	A measurement between two spots or stars, such as angular distance, pixel distance, angular separation, pixel intensity, magnitude intensity, etc.
Feature List	=	An ordered database of stars with corresponding features based on a pattern
Image	=	A 2-dimensional array of pixels with intensity measurements per pixel returned by an imager.
LISA	=	Lost in Space Algorithm. An identification algorithm that requires no a-priori attitude information.
Match	=	A spot that has been correlated to a star.
Pattern	=	A group of features created from a selection of spots or stars.
Star Camera	=	A device that obtains images.
Star Tracker	=	A star camera primarily used to fix the orientation of a satellite to a single star.
Tolerance	=	The amount of variation given to a feature during an algorithms search through a database for identification.
Spot	=	A single or group of illuminated image pixels that has been centroided to a single point on an image.
Spot List	=	An array of spots and their 3-dimensional unit vector locations relative to the image.
Tags	=	The number of instances the same star has been used to identify a spot.
Votes	=	The number of instances a verification has proven a match.

CHAPTER 1

INTRODUCTION

I. Overview

Technology has made great strides in the past two decades with the creation of the first CubeSat class satellites [1]. These provide dramatically reduced carry weight at launch, the increased ease of jettisoning, and the opportunity to be used as verification and validation vehicles [2]. However, these small satellites have similar issues that affect larger spacecraft such as the physical allocation of on-board components [3], scientific instrumentation restrictions [4], the ability to navigate [5], and one of all satellites' primary requirements: the ability to determine its attitude in space [5].

Many attitude determination sensors exist such as: Magnetometers, Sun sensors, Gyroscopes, Earth Horizon sensors, Orbital Gyrocompasses, Star Trackers and Cameras, Solar panel sensors, and more [5]–[7]. These sensors range in accuracies from 0.1^{o} to 5^{o} for sun, magnetometers, and solar cells [8], [9], and may require upwards of 60 minutes for attitude convergence [9].

II. Star Cameras

Pioneered in the early 1970's [10], [11], star cameras are area array imaging sensors built from Charge Couple Devices (CCD) and are the most accurate instrument for spacecraft attitude determination, with accuracies ranging within 2.78 μdeg to 0.05 deg [12]–[14]. These star camera systems are extensively used in attitude determination and many other uses such as inertial platform supervision and correction, automatic tracking of artificial satellites, missile plume, and visible radiation [7]. These star cameras weigh 2.2 to 30 lbs with power consumption rates of approximately 5 to 120 Watts [12],[15]–[17].

Several commercially available star camera systems have been developed and tested on-orbit with various star identification programs and include: HAST [17], Terma HE-5AS [15], ST5000 [16], Ball CT633 [18] and many others. The design types range from Canopus to Gimbaled to Rocket/Missile to Fine Guidance [11], and can reach in price anywhere from $50,000 to half a million dollars [16],[19],[20]; yet despite the cost provide exceptional high quality images with low to moderate resolution, lens baffles, small Field of Views (FOV), and precision optics. Advancement in CubeSat specific star cameras have

yielded the Blue Canyon XACT [21], Berlin's BST ST-200 [22], and others with masses of 0.11 to 1.54 lbs, power consumption between 220 mW to 2 W, and volumes of 3.0x3.0x3.81 cm^3 (0.0343U) to 10x10x5 cm^3 (0.5U).

III. Star Fields and Identification

In the past 31 years the design direction of star camera sensors has turned towards star-field images using a wider FOV and away from single star images [23]. This improves the ability to obtain attitude determination by using multiple fixed points in the sky to acquire a quaternion solution.

Star-fields are easier to recognize due to their geometric relationships, such as angular separation between stars and spherical distances, making a star-field more unique than tracking a single star. Star magnitudes, or star intensities, alone are an undependable means of recognizing stars in an image from a star camera, though they can be used to aid in the reduction of stars in an image. Combined together, these geometric relationships and magnitudes create recognizable areas of interest in the sky that can be used to identify stars in an image and calculate an attitude solution.

As early as 1963 [7] various star identification algorithms have been developed to identify stars in an image, the techniques and methods of which range from the experimental to the space qualified. These algorithms interpret stars in an image as spots and translate these spots into Earth Central Inertial (ECI) coordinate positions, which are converted to a quaternion estimation of the satellite's position in ECI.

The ability to recognize stars autonomously and to determine spacecraft attitude with only a simple star camera is a great advantage, yet to correctly identify the stars in a satellite's view requires the correct identification algorithm for that star camera, proper tables and catalogs of stars, and well defined optic lens distortion calibration.

However, with the widening of the FOV, issues facing star identification arise. These include lens distortion in the star camera image, false objects in the image (i.e. planets, other satellites, radiation, etc.), and noise sensitivity. Highly capable star identification algorithms are needed to facilitate correct star identification and attitude determination for the spacecraft. This thesis will investigate a few of these algorithms and their behaviors as well as their limitations in identification, solution attainability, error prevention, and probabilities of error.

IV. Commercial Cameras

One of the most critical situations for a spacecraft is determining its attitude in space. This is called the Attitude Determination problem. A subset of this problem is for the case where star identification algorithms are given no a-priori attitude knowledge and must still achieve star identification. This is termed as the Lost in Space problem and the programs and methods that meet this need are called Lost in Space Algorithms (LISA's).

Within the last ten years, developments in cellular phone hardware and pixel imaging have produced commercially available digital cameras boasting light weight, lower power consumption, and higher resolution images [24]. Current cell phone cameras, ranging in weight of 3.95 to 6.34 ounces [24], improved resolutions of between 1 to 41 mega pixels [25],[26], pricing of $10 to $800 [26],[27], and volumes near the 0.01U to 0.08U, are ideal for CubeSats and provide opportunities for more affordable and low mass-cost missions. However, these cell phone cameras are not specifically designed as star cameras, having general characteristics comprised of large FOV optics, lower optic and lens quality, increased noise sensitivity, and spectral intensity (color) dependency. Such characteristics define cell phone cameras as being low quality star imagers.

With these commercial products being so readily available and more powerful every year, it is proposed to evaluate if Lost in Space Algorithms can be used on low quality star imagers. Given the difficulties in star identification and the reduced resolution of current small camera devices, questions arise such as:

- Can these identification methods be used on high resolution, but low quality cameras?
- How is the behavior of the identification process influenced by a low quality star camera?
- How well can the identification algorithms obtain attitude determination with distorted data?

With a typical cell phone, it is expected that the processing ability will be enough to not only use the identification algorithms and process attitude control, but have sufficient remaining processing power to control a CubeSat in flight. The desired goal is to test whether or not one or multiple star identification algorithms tested in this analysis could be potentially a viable Lost in Space solver and could be placed on a

camera phone. Experimentation will be conducted with a low quality test camera called an Aptina [28]. Multiple identification algorithms will be created and tested to prove whether star identification algorithms can be used on this camera and by extension other possible commercial products, by measuring the errors in the identification process of the algorithms to find a model of comparison between identification programs. Error bounds, reference catalog selections, ground based and spaced based solutions, simulation results, and judging criteria will be shown. It is anticipated that the quality of the Aptina imager will be more than sufficient to identify stars.

V. Thesis Statement and Objectives

A. Thesis Statement

The purpose of this research is to model and analyze the errors in Lost in Space Star Identification Algorithms to determine whether these algorithms can be used with a low quality camera to obtain attitude determination.

B. Objectives

There are four primary objectives in this research to aid in establishing the viability of using Lost in Space Algorithms on low quality cameras.

1. Identify which identification algorithms are most likely to succeed. This will be accomplished by an in-depth review of past methods and selection of which past identification algorithms have been of greatest success. Based on this, additional identification algorithms will be constructed to test methods of star processing and verification.
2. Identify boundaries for using star identification algorithms with a star camera. This will be achieved through examination of sources of error for the identification algorithms. These sources of error will be defined based on past research and camera parameters. The identification algorithms will be evaluated through simulation and experimentation to determine their range of suitability within these errors.

3. Define probability of errors for each algorithm. Achieved through measurement of the number of incorrect solutions from the images the algorithms solve. This will be compared to the number of solutions that are returned empty or successful.
4. Develop a suitable simulation model of the sky and camera for algorithm testing. Attained through construction of MATLAB programming. Will be used to externally input values to the identification methods and evaluate the solutions returned by the algorithms.

CHAPTER 2

STAR CATALOGS AND IDENTIFICATION METHODS EXAMINED

I. Star Catalogs

The basic building block for star identification is a *Star Catalog*. The first star catalog on record was created in 127 B.C by Hipparchus of Nicaea [29] containing approximately 1025 stars. Over time, and with the development of more sophisticated star instrumentation, star catalogs have been enhanced to contain additional information such as the associated constellations to a star, the color, brightness, carbon content, position in the ECI (Earth Centered Inertial) coordinate system, dwarf star content, etc.

The most fundamental star catalog that will be necessary in the use of star identification must contain an indexing of the stars in the sky and their positions. 25 star catalogs were reviewed by Thurmond [29] outlining the number of stars in each catalog and the method in which they were collected. Table 2.1 shows the star catalogs that were found and the year in which they were published.

Table 2.1 Summary of star catalogs

Catalog Name	Observer	# of Stars	Published Date
Rhodes	Hipparchus	1025	127 B.C.
Almagest	Ptolemy	1028	150
Zij-I Sultani	Ulugh Beg	992	1437
Astronomiae Instauratae	Tycho Brahe	777	1592
Rudolphine Tables	Kepler	1000	1627
Catalogus Stellarum Fixarum	Hevelius	1564	1690
British Catalog	Flamsteed	2866	1712
Coelum Australe Stelliferum	Lacaille	9766	1742
Praecipuarium Stellarum Inerrantium	Piazzi	6748	1803
FK		1535	1879
Bonner Durchmusterung	Argelander	457848	1886
Cordoba Durchmusterung	Thome	578802	1932
Carte du Ciel		1958	1887
Cape Photo Durchmusterung	Gill & Kapteyn	454875	1896
AGK		8468	1900
BSC-HR		9096	1908
PGC	Boss	6188	1910
Henry Draper	Pickering & Cannon	225300	1918

SAO	Smithsonian	258997	1966
Perth 70		24978	1970
Hubble GSC	NASA	15000000	1990
PPM	NASA	181731	1991
Hipparcos	ESA	118218	1997
Tycho	ESA	1058332	1997
2Mass Point Source		470992970	2003

A. Star Databases

Of the catalogs listed, few provide the direct information necessary for star identification as required in this analysis. From this list the most relevant for use of star identification were the Henry Draper [30], PPM [31], Hipparcos [32], and Tycho [32] catalogs.

1. Henry Draper Database

The Henry Draper star catalog used a prism in front of the telescoping lens to spread the light according to wavelength to obtain spectral information per star. This provides a highly specific means of identifying stars, as each star would emit a unique wavelength spectra. It is a whole sky catalog observing stars up to a magnitude of nine [33].

2. PPM Database

The PPM (Positions and Proper Motions) catalog covers nearly two hundred thousand stars north of the 2.5 degree southern declination for the epoch J2000. The main purpose of the catalog was to provide a dense and accurate net of astrometric reference stars on the northern celestial hemisphere [34].

3. Hipparcos-Tycho Databases

The Hipparcos and Tycho catalogs were developed in 1989 with the launching of the ESA's (Eurpean Space Agency's) funded satellite *Hipparcos* which flew from 1989 to 1993. This name comes from the acronym for High Precision Parallax Collecting Satellite. Its main function was to obtain accurate parallax measurements and star intensities. The Hipparcos database was published in 1997 and cataloged precise lightyear distances and ECI positioning of 118218 principal stars to a resolution of 1 milliarcsecond [29]; an updated version with re-processed data was published in 2007. The Hipparcos catalog was particularly

notable for its stellar parallax measurements, which were more accurate than those produced by ground-based observations [33]. The Tycho catalog contains nearly ten times more stars, each measured 130 times during the mission to an accuracy of 25 milliarcseconds.

B. Main Identification Catalog

The Hipparcos Database was chosen due to its high positioning knowledge to 1 milliarsecond and for its comprehensive database listing of stars. It contained a sufficient number of stars in both hemispheres for the ability to identify stars globally. Star intensity information allowed for variability in testing of camera parameters during simulation and experimental analyses.

A fully updated version of the database was published in 2007 [33] and was well known for its ease in star indexing and precession. Below in Table 2.2 is an example of the catalog information formatted from the Hipparcos database [35].

Table 2.2 Hipparcos database formatting

Hipparcos Cat. Field	Name	Description
H1	HIP	/Identifier (HIP number)
H2	Proxy	/Proximity flag
H3	RAhms	/RA in h m s, ICRS (J1991.25)
H4	DEdms	/Dec in deg ' ", ICRS (J1991.25)
H5	Vmag	/Magnitude in Johnson V
H6	VarFlag	/Coarse variability flag
H7	r_Vmag_Source	/Source of magnitude
H8	RAdeg	/RA in degrees (ICRS, Epoch-J1991.25)
H9	DEdeg	/Dec in degrees (ICRS, Epoch-J1991.25)
H10	AstroRef	/Reference flag for astrometry
H11	Parallax	/Trigonometric parallax
H12	pmRA	/Proper motion in RA
H13	pmDE	/Proper motion in Dec
H14	RA_Error	/Standard error in RA*cos(Dec_Deg)
H15	Dec_Error	/Standard error in Dec_Deg
⋮	⋮	⋮
H75	VI_Color_Reduct	/VI used for reductions
H76	Spect_Type	/Spectral type
H77	Spect_Type_Source	/Source of spectral type

II. Star Identification Algorithms Reviewed

Since the 1970's [36], several star identification algorithms were created to answer the Lost in Space problem, and are separated into two categories of identification analysis: 1) an instance of subgraph isomorphism, or 2) pattern recognition. Subgraph isomorphism deals with the angular separations between the stars and their adjacent neighbors; pattern recognition associates stars with a pre-defined image pattern, such as is used in facial recognition. The latter includes Grid algorithms, Neural networks, and Genetic algorithms [36]. The focus of this study will be on the first classification of star identification methods using subgraph isomorphisms. Brief descriptions of several of the algorithms developed under this classification are presented below based on the author who created them. The terminologies used in this work are set in braces {} next to the authors' original terminologies which are left intact to maintain the authors' meaning.

A. Gotlieb

Gottlieb [37] in 1978 developed the Polygon Matching method. From a set of measured stars {spots} two are arbitrarily selected as points {spots} 1 and 2, and the corresponding angular separation {feature} was computed and denoted as d_m^{12}. Then all pair of stars {spots} *(i,j)* in a finite region of the catalog are selected such that:

$$\left| d(i,j) - d_m^{12} \right| \le \varepsilon \qquad (2.1)$$

where ε is the uncertainty {tolerance} in the distance measurements of the star sensors {imager} and *d(i,j)* is the angular separation {feature} calculated for the pair of stars in the catalog. Gotlieb's method states that the number of star pairs {pattern} is not negligible. Even if only one pair {spot to star ID} of observable stars {spots} is obtained, two possible star identifications exist. In both instances, it would be necessary to select another measured star {spot}, point 3, from which two more separations {features} could be calculated, d_m^{13} and d_m^{23}. A third star {spot}, d_k, was then searched for in the catalog that could be combined with the previous pairs {patterns} such that,

$$\left|d(i,j)-d_m^{13}\right| \le \varepsilon \text{ and } \left|d(j,k)-d_m^{23}\right| \le \varepsilon \qquad (2.2)$$

or,

$$\left|d(j,k)-d_m^{13}\right| \le \varepsilon \text{ and } \left|d(i,k)-d_m^{23}\right| \le \varepsilon \qquad (2.3)$$

however, if more than one pair {pattern} was found in the catalog, then the process would be repeated by narrowing the uncertainty {tolerance}, ε, until the identification was unambiguous.

B. Groth

In 1986, Groth [38] created a Two-Dimensional Coordinate Pattern Matching algorithm which used sub-catalogs {feature lists}. The matching was based on the identification of similar geometrical configurations {features} of points {spots} as triangles in two lists {feature list and spot list}. Provided the two lists {feature list and spot list} had a sufficient number of points {spots} in common, the image distortion was not too severe, and the random coordinate errors were minimal, identification could be obtained. It is claimed to be insensitive to any translation, rotation, magnification, or image inversion. The objective of the algorithm was not to match all points {spots} in the two lists of arbitrary size, but from the matches found, a coordinate transformation between body and ECI coordinate systems could be derived. Other points {stars} in the lists could then be matched by comparison to the identified points {spots}, e.g., by matching points {stars} that were sufficiently close. Groth suggested that a faster way to search the sub-catalogs {feature list} would be to sort the triangles' sides [39], created in the pattern generation step, in order based on permutation-invariant values, such as the logarithm of the perimeter of a triangle.

C. Kosik

Kosik in 1991 [37] developed a Distance-Orientation method which improved upon the polygon technique used by Gotlieb by requiring an approximate estimate of attitude. This estimate enabled the projection of the star catalog region {sub-catalog} onto the star imagers FOV. The same stars would be found but projected onto a tilted (theoretical) star sensor {image}. As a consequence the adequate pairs {patterns} would verify a distance criterion {search tolerance}, but should also have approximately the same orientation. Each pair {pattern} could be considered as a set of vectors, and the vectors of both

measured stars {spots} and projected catalog stars {stars from feature list} should have approximately the same orientation.

In his algorithm a set of catalog pairs {star and spot pairing} is obtained from his distance criterion and each pair which could satisfy this criterion for angles around 0° and also for angles of obtention around 180° were kept. The obtention of an angle, according to Kosik, meant that the order of the catalog stars is opposite to what he defined as the right order (i .e., if a catalog pair of stars (1c,2c) compared to a measured pair of stars (1m,2m) differs from about 180° , then the catalog star 1c corresponds to spot 2m and the catalog star 2c corresponds to spot 1m). If more than one pair of stars {stars from feature list} is obtained it is necessary to select another measured star {spot} and continue the process. Thus, if any ambiguity existed, the stars {spots} were rejected and the identification proceeded to the next group {pattern}.

D. Anderson

In 1991, Anderson and Junkins [39],[40] attempted to address the uncertainty of star triplets {patterns of three spots} by proposing a permutation matrix, and the development of star {spot} pattern parameters {features} that were independent of the order in which the stars {spots} were selected to reduce search time in identification of the catalogs {feature lists} (i.e. the features from the image are unsorted). It was Anderson's desire to also find a means to enhance the performance of low earth orbit star trackers which were typically affected by nonlinearities such as lens distortion, coma, and chromatic aberration, as well as atmospheric refraction, thermal cycling, and vibration. Using star-triplet patterns {patterns of 3 spots}, Anderson proposed the use of an array processor to handle matrix multiplications required in his permutation matrices. However, his database storage remained higher than he had anticipated, and there were no advances made on the database search times based on his assumptions.

E. Renken

In 1992, Renken et al. developed his own method called the Renken method [41]. He began development with the creation of sub-catalogs {feature lists} based off of the Smithsonian-Astrophysical-Observatory (SAO) catalog. These sub-catalogs were reduced to include only the stars that were detectable by a CCD camera, which he left unspecified. Planets were added automatically to the sub-catalogs with their positions in order of date and time. Errors in the star catalogs were handled manually.

His algorithm used a procedure which initialized matching of variables by removing stars {spots} from the image based on thresholding the intensity of the stars {spots} in the image. He began by constructing an array of pixel-distance-xy {pixel based features} of all segmented objects {remaining image spots} which were multiplied by a value between the pixel distance on the CCD pixel array and the angle at the sky (degree per pixel). This value depended on the CCD pixel size and the focus length of the optic being used.

A tolerance value resulted in maximum and minimum values of the distance between stars {spots} which were separated into two arrays. This tolerance allowed Renken to compensate for inaccuracies in his star catalog and imaging errors of the optic device. The last phase before matching was to calculate the cosine of the minimum distance with respect to the maximum distance in his arrays.

Within all of his matching procedures {program} each object {spot} is connected to all other objects in a comprehensive approach. For his algorithm, Renken required a minimum of three matched stars {spot to star ID} for a 3-axis attitude determination.

To handle a potential variety of matching results, he implemented a verification procedure. This procedure considered the results of former attitude determinations stored in a history-buffer {initial attitude registry}, thus becoming a tracking system. However, to handle the correction of miss-matching {false identification} during potentially extreme situations, the history-buffer would be downlinked and pattern matching {identification} would need to be done by hand.

F. Liebe

In 1992 Liebe [42] established a method which he called the *Lost in Space Algorithm* for star identification by obtaining a set of what he called *Features* which were based strictly on the nearest two stars {spot neighbors} to what he called a *Central-Star*. In 1995 he used this algorithm in conjunction with a CCD imager and a microprocessor to create a star tracker with a precision of 1 arcsecond [42].

His algorithm approaches the identification process by obtaining a list of all the measured stars {spots} in an image then retrieving the first star {spot} in an image, labeling this as the Central-Star. Following, he then detected the nearest two stars {spot neighbors} to the Central-Star. These were recorded and used to calculate the dot-product angles {features} between the Central-Star and the other two stars {spots}. In addition to these angles {features}, he would calculate the sub-spherical angle {another feature} between the three stars {spots} using the Central-Star as the vertex. This sub-spherical angle he called the *Interior*

angle, which is calculated by taking the dot-product of the vectors between the Central-Star and the other stars {spots}. This he did by taking the angle from the planar projection of the vectors between the spots in the image, Figure 2.1.

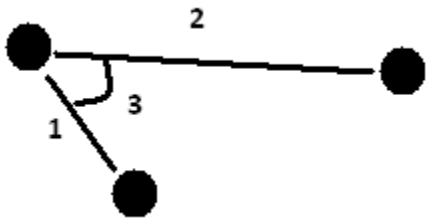

Figure 2.1 Liebe's pattern showing (1) and (2) as the dot product angles, and (3) as the interior angle of the Central-Star

The combination of these three stars {spots} and their associated angles was called a pattern. This process continued until every star {spot} in the image was used as a Central-Star and all patterns were recorded. A limitation of his algorithm is in the formation of the patterns using only the nearest neighboring spots rather than forming patterns using every combination of spots together in the image.

Once the patterns were found from the image, he compared them to a sub-catalog {feature list} where each star {spot} was solved for the most likely matching candidate. If multiple solutions existed for a given star {spot}, the most frequent result was selected as the correct solution; e.g. if spot 1 in the image is solved in each pattern and returns the results, Star 1, Star 4, Star 1, and Star 5 from the feature list, then the returned solution for spot 1 will be Star 1.

No type of additional validation or verification was found to be present in his method. This algorithm became fairly limited in scope as the number of false star spikes in the image increased. Since the Central-Star recognizes only the two closest spots near it, the addition of false star spikes reduces the chances of success as these false spikes approach the Central-Star.

G. Baldini

In 1993, Baldini et al. [43] proposed a Multi-Step Star-ID method. Baldini's method identified the brightest *b* stars {spots} in a given image, after which he then measured the angular separation {features}

of the sequence of five stars {spots}. He then proceeded through a linear examination of the catalog for every star in the catalog which fell within a prescribed tolerance. Comparing the distances {features} of each star {spot} in adjacent lists {feature lists} (somewhat similar to Groth's method) he would determine if any star {spot} exceeded the tolerance of the observed angular separation {feature}. As items {possible star ID's} were eliminated the number of each iteration comparison was reduced. Baldini was then left with *b* lists containing stars {spots} that met their neighboring distance criteria {tolerance}. He then formed all combinations {patterns} of the stars {spots}, discarding combinations whose sequence of angular separations {features} did not match the observed stars {spots} in his list {feature list}.

H. Scholl

In 1994, Scholl [44] published a more straightforward method called her Six-Feature Star-Pattern Algorithm. The image spots were to be ordered and removed {thresholded} by their relative intensities, eliminating the permutations that arise when considering the possible orders of three stars {spots}. Her desire was to eliminate the need for a-priori attitude knowledge of a star tracker. She states that the significance of a single triangle as a pattern is that it would be independent of any in-plane rotation angle and translation.

Uniquely, if multiple solutions exist for a triangle pattern, she automatically decreased the value of her feature tolerance around the magnitude and distance to progressively tighten until a single star field was identified.

I. Ketchum

In 1995 Ketchum et al. [45] proposed a 2nd Sequential Filter algorithm, following the work of van Bezooijen [46]. Ketchum uses the intensity of the brightest star {spot} in the image to determine the likelihood of pointing in any particular direction and filters the list {feature list} of possible stars using the brightness of the second brightest star {spot} in the image, although she admits the algorithm would need to search as much as 43% of the catalog {feature list} for the appropriate stars.

Ketchum uses in her analysis a star catalog called the standard GSFC Flight Dynamics Division (FDD) Multi-Mission System catalog. The catalog was used in the Gamma Ray Observatory and the Upper Atmospheric Research Satellite missions [45].

Her measurements are taken using a 4-degree radius FOV imager. The algorithm is based on constructing *polyhedrons* out of the stars {spots} in the image. However, due to the small FOV, there existed a nonzero probability that the FOV would not contain a primary bright star, which was the basis for her sub-catalogs {feature lists}.

Ketchum's algorithm uses several recursive steps to verify the identity of the stars {spots} and ensure that the correct sub-catalogs {feature lists} are used. Her algorithm is among of the first to attempt star identification without a-priori knowledge; however, it is dependent on the star intensities and thus dependent on the performance and calibration of the imagers used.

J. Mortari

In 2004, Mortari et al. [47] developed the Pyramid algorithm, supplemented with his k-vectoring technique [48]. This algorithm uses a minimum of 4 stars for feature extraction and pattern creation. Mortari's Pyramid design was described by Spratling [39] as using 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 do not match, suspecting them to be non-star spikes (false spots) on the image plane. Mortari's code had been tested to reject non-stars in an image containing only five real stars but with 63 non-stars included, however, this was done with very low centroiding error.

He generated patterns beginning with the first star {spot} of the image being the apex of the pattern (one of the corners) and would select in turn the next two stars {spots} of the image to build a triad pattern. With this established, the next star {spot} in the image was selected to verify the validity of the triad. This 4^{th} spot created another three possible triads, hence the impression of a Pyramid with 6 features. If this Pyramid did not match with the patterns in the sub-catalog {feature list}, then the algorithm kept the initial 3 stars {spots} and used the next observed star {spot} in the spot list to generate a new Pyramid.

In Figure 2.2, the three vertices *(i, j, k)* are the primary observable stars {spots} that the algorithm wishes to identify, and vertex *r* is the fourth star {spot} used as verification, with α's being the angular distance between the observable stars {spots}. With four triad patterns (each triad containing upwards of six features), the features are compared to patterns within the feature list using a feature tolerance. Out of a possible 24 features, Mortari uses only 6 for his identification. Furthermore, he uses a verification phase prior to returning a solution.

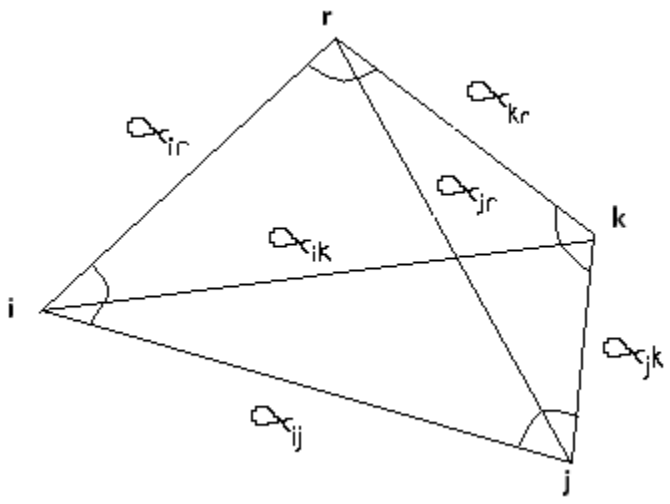

Figure 2.2 Depiction of Mortari's Pyramid scheme

The Pyramid algorithm was successfully tested in-flight on Draper's "Inertial Stellar Compass" star tracker [49] used on MIT's satellites HETE and HETE-2. This algorithm is presently under exclusive contract to StarVision Technologies, thus, neither pseudo code, nor programming was obtainable due to infringement issues.

K. Samaan

Recently in 2005, Samaan, in conjunction with Mortari and Junkins, presented two separate methods using an advanced searching routine called the *k-vector* [48],[50]. The first algorithm is called the Spherical Polygon search (SP-search) and the second the Star Neighborhood Approach (SNA). For the SP-search, the method accessed the stars that could potentially lie within the star tracker FOV and then calculated the interstar angles {features} between the measured stars {spots} and the cataloged stars. The SNA method also accessed candidate stars from the catalog {feature list}; performing its star identification by locating the observed cataloged stars {spots} by a cataloged knowledge {initial attitude estimate} of stars neighboring the identified stars {spots} from the previous image.

These two do require a-priori attitude knowledge, hence are not Lost in Space algorithms, yet are mentioned here for their unique ability to speed the processing of star identification against star catalogs and feature lists. Initial attitude knowledge is used to ease star identification by truncating the sub-catalogs

{feature lists}. It must be noted that the centroiding precision they used was on the order 17 μrad, which defines a high quality imager. They conducted several tests to examine the slide, or sweep, of spots in an image using a rate gyro to obtain an angular velocity vector and calculate the quaternion rotation matrix. These two methods were super-imposed into the Pyramid algorithm programming to speed searching through the star catalog. The exact method in which this was done was not specified.

L. Rousseau

Rousseau et al. [51] published a method in 2005 called Star Recognition Algorithm, which he claimed as being robust to errors introduced by a new CMOS Active Pixel Sensor (APS). The algorithm's metric is the sine of star-triangle {triangular pattern} interior angles {features}, yet instead of using any combination of stars {spots}, he used only the closest two stars {spots}, and used only one of the three (two independent) interior angles as a parameter. His pattern selection meant there was only one entry in the catalog for each star. Furthermore, Rousseau did not specify a method for selecting star triangles {patterns} from the catalog {feature list}.

Rousseau attempts to identify a group of 3 spots and uses the identification to compute the approximate attitude of the imager. This initial attitude estimate is then used to truncate the catalog {feature list} and finds all the stars that should be visible. 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 best of the matches of all the triangles is then selected.

It is unclear whether Rousseau's performance data is on his original 45,000 star catalog, or another mentioned, reduced 1,300 sub-catalog. Though he conducts his tests in MATLAB, which unquestionably increases 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. 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

M. Zhang

In 2007, Zhang et al. [52] proposed a feature extraction technique, similar to Liebe [42], using the inter-star (dot-product) angles {features} and the angle made by two stars {spots} relative to a central star {spot} (i.e. the interior angle of the 3 spots similar to Liebe). One of the most unique details of his algorithm is the use of polar coordinate values as a means of feature creation. Though Zhang's work is similar to the definition of a Grid Algorithm [52], his method still can be classified under the isomorphism class of algorithms as his creates new patterns per image and compares feature to feature, rather than grid to grid.

Differing from many other identification methods, Zhang creates a sub-catalog {feature list} called a *List Entry*. Rather than creating every possible combination of features from reference stars {main catalog database}, he selects only the *m* possible sub-catalogs {feature lists} containing *n* stars that fit the image. This greatly sped identification processing time and helped remove possible errors. However, he says that in the case of false-star spikes, this would potentially cause the algorithm to select the wrong sub-division of the catalog, which would not contain the necessary information for star identification.

N. Kolomenkin

Kolomenkin et al. in 2007 [53] presented the Voting Algorithm. The algorithm was based on a geometric voting scheme in which a pair {pattern} of stars {stars} in the catalog voted for a pair {pattern} of stars {spots} in the image if the angular distance {features} between the stars {stars and spots} of both pairs {patterns} was similar. He states that the angular distance is a symmetric relationship and that each of the two stars from the catalog would vote for each of the spots in the image. The identity of each star {spot} in the image would be matched to the catalog star {star} that cast the most votes for that star {spot}. After gathering all the identities of the stars {spots}, the ECI positions were used to compute the imagers pointing quaternion {attitude}.

He stated that nearly 80 pairs of catalog stars will be found for each image star {spot} pair {pattern}. Stars {spots} with incorrect identities will receive a very small number of votes, whereas correctly identified stars {spots} will support each other. He used what he called a *"clustering algorithm"* which was defined as a conditional statement that if the number of votes for a star {spot} was close to the maximal number of votes among all stars {spots} in the image then the star identification was considered correct.

This process was effective in eliminating erroneous matches and used to recognize the correctly identified stars.

Typically, the images contained false-star spikes and were matched to stars from the catalog in Kolomenkin's voting stage. Afterwards, a validation phase was used to allow the algorithm to handle even a large set of false stars. The algorithm was also able to handle true stars {spots} which were erroneously matched in the voting stage. The latter happened, he states, when the star {spot} had only a few close neighbors.

Kolomenkin affirms that there are many possible variations of this voting algorithm. These depended on specific camera qualities, fine tuning, and accuracy versus speed requirements. The basic algorithm did not exploit the star brightness information due to camera image and catalog star brightness values that could not be matched reliably at that time. However, a rough match could be made by dividing the brightness values of the image stars {spots} into a few (2 to 4) brightness groups. Comparing the brightness of catalog {reference stars} and image stars {spots} aided in the identification process and removed erroneous matches. Kolomenkin implemented and tested his algorithm on simulated data and on real sky images, but it is unclear if it has been tested on actual in-flight systems.

O. Tichy

In 2011 Tichy et al. [54] used two stars to create an identification algorithm which he labeled as the Two Star Voting algorithm. This method obtains patterns from the basic use of the dot-product between two 3-dimentional vectors (Figure 2.3). Using the center of the imager as his origin, the angular distance between the two spots were created and recorded as a single feature. This computation continued between each spot in the image until all possible combinations of two spots were created. Due to a single feature being created in each instance, the use of the word feature and pattern are interchangeable, however, strictly for the case of this method. The observable stars were indexed and used to index the patterns in a table containing the position of each observable star on the imager, and the angle between them.

Tichy advanced his identification method by incorporating the strategy of a geometric Voting Algorithm [53]. This algorithm operated in two phases - the voting phase and the validation phase. The angular distance of each pair of spots on the image was compared with the angular distance in a sub-

catalog. These pairs {patterns} of spots receive a vote as a corresponding pair {pattern} of stars if a match in the sub-catalog {feature list} is found for some fixed tolerance attached to the angular distance.

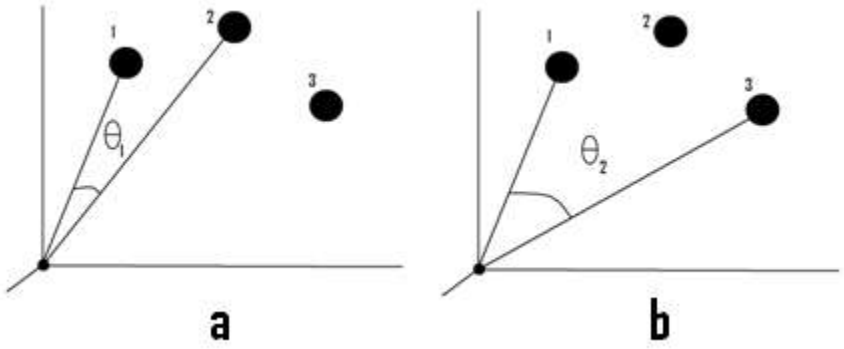

Figure 2.3 Two Star Method showing pattern creation: a) The first pattern with angular feature θ_1. b) Next pattern creation

Once these pairs {patterns} are all collected, he would compare the number of votes received for a reference star per spot in the image and maintain only the spots and reference stars that had the maximum votes. His method was restricted to stars which were within 0.7 rad beyond the FOV of the imager and required an a-priori attitude. Tichy assumed that a rough attitude estimate would be available from another source such as a magnetometer or Kalman filter prediction.

P. Computational Considerations

Spratling et al. [39] in 2009 compared the computation performance of the algorithms from Groth, Anderson, Liebe, Baldini, Mortari, and Zhang. He also included other identification methods that fell under the pattern/grid recognition class of identification.

Spratling says of Groth that his algorithm runs at a high polynomial power of order *n* stars {spots} when searching the sub-catalogs {feature lists}, but could be improved by sorting the triangles sides {features in the pattern} based on permutation resistant values, e.g. the logarithm of the perimeter of the triangle. Overall, the method ranked at a high level of computation time.

Baldini uses five spots, inherently containing twelve independent features, but uses only nine features when performing the identification process. Spratling [39] suggests this means the required field of view

may be larger for Baldini's method when compared to other methods to ensure that sufficient visible stars exist.

Spratling quotes Anderson by stating his method would improve in computation given an array processor to perform the matrix multiplication, decreasing the running time of the star identification process. However, it must be noted that the use of array processors use comparatively large amounts of power in contrast to a serial processor. His method also ranked high in computation time.

Liebe is said to have much reduced processing time searching through feature lists. Spratling provides the equations for Liebe's system time, where his feature extraction operation is of order,

$$O(f \log_2 b) \tag{2.4}$$

and his database size as,

$$O(n) \tag{2.5}$$

where *f* was the number of stars in a given sub-catalog {feature list}, *b* was the number of stars in the pattern, and *n* was the number of stars referenced in Liebe's star catalog. Though his feature extraction took longer than Anderson's, his database search time could be reduced. Spratling mentions that Liebe, by incorporating into his own algorithm an optional recursive algorithm, was able to identify stars upwards of 20 times faster than his original Lost in Space Algorithm.

During Baldini's processing of the distance comparison between feature lists, Spratling notes that although non-stars {false spots} would get weeded out in the process of identification, the addition of non-stars to the algorithms increases most of the steps linearly or quadratically. Baldini's method requires

$$O\left(b(\Delta m n)^2\right) \tag{2.6}$$

time to compute the operation of these spots if they are within the tolerance; where Δm represents the fraction of stars in the catalog that fall within the tolerance range. The disadvantage of this method is the requirement that star intensity values be used to aid in the identification process, which makes the

algorithm highly dependent on the performance and parameter details of the imager, but also uses more processing time for identification.

Mortari's Pyramid algorithm was among the fastest in star identification computation. Using what Mortari called his "k-vector" approach, the amount of time required to search the database {catalog} and tables {feature lists} could be independent of the size of the database. This was the fastest among the algorithms in terms of database searching with equation 2.7 as the feature extraction and equation 2.8 as the database search, where *k* is is the number of possible star {spot} pairs with inter-star angles within the tolerance.

$$O(b) \tag{2.7}$$

$$O(k) \tag{2.8}$$

Q. Author Summary

Provided in Table 2.3 is a listing of the identification algorithms previously discussed outlining the authors by date and algorithm name. Included is the minimum number of stars required in each algorithm to produce a solution and the number of features in each pattern. The table shows that the minimum number of spots required in the FOV of the star camera is within 3 to 4 spots, for the majority of the star identification methods. This has remained the minimum for the past 30 years.

The variability between the algorithms exists in the manner of searching the star catalogs and feature lists, the manner in which patterns between spots are constructed, the number of features in a pattern, and the verification used. All the authors conclude the necessity to have a tolerance applied to the features and that this tolerance selects multiple possible solutions to any given spot in the image. The value of this tolerance, and the way in which the feature lists are created and ordered can adversely affect the identification process and should be chosen carefully based on the characteristics of the optics being used.

As well, many of the authors used the magnitude intensities of the spots in the image to further reduce possible false spots and noise prior to identification. Some others, like Ketchum, use the spot magnitude to select which of a multitude of feature lists ought to be used for identification, rather than using a single large listing of stars.

Table 2.3 List of star identification methods and authors

Author	Year	Name	Min. # Stars Needed	# of Features per Pattern
Gottlieb	1978	Polygon Match	3	3
Groth	1986	2D Coordinate Pattern Matching	3	NA
Kosik	1991	Distance-Orientation	2	2
Anderson	1991	Permutation Matrix	NA	NA
Renken	1992	Renken	4	4
Liebe	1992	Lost in Space	3	3
Baldini	1993	Multi-Step Algorithm	5	1
Scholl	1994	6 Feature Method	3	6
Ketchum	1995	2^{nd} Sequential Filter	2	NA
Mortari	2004	Pyramid Algorithm	4	6
Rousseau	2005	Star Recognition	3	NA
Zhang	2007	Radial-Cyclic	2	NA
Kolomenkin	2007	Voting Algorithm	3	3
Tichy	2011	Two Star Voting	3	3

CHAPTER 3

METHODOLOGY AND DEVELOPMENT

I. Methodology

This section outlines the manner in which the star databases for identification are created and used, the inputs to the identification algorithms, concept and method of feature list truncation, and feature list organization. Figure 3.1 illustrates the manner in which the catalogs, imaging, and identification systems relate to one another.

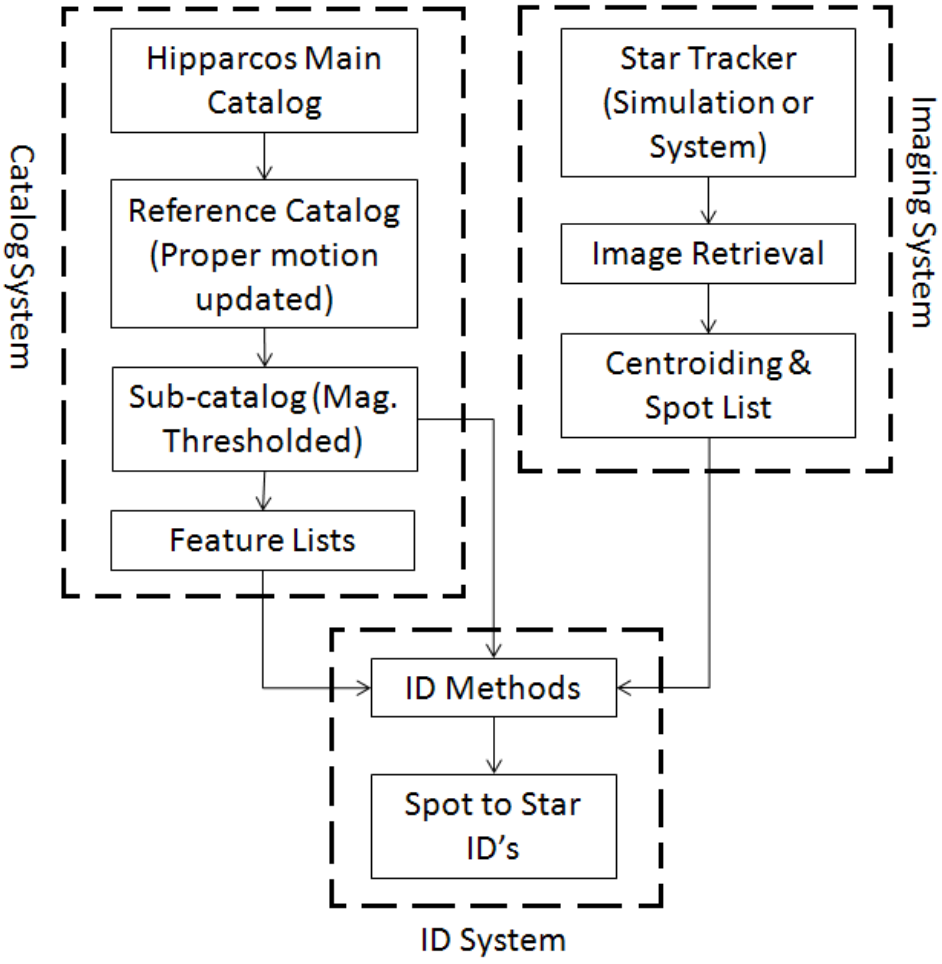

Figure 3.1 Catalog, Imaging, and ID system flowchart

A. Star Catalog Databases

1. Reference Catalog

The catalog processed in 2007 was selected based on its larger database of stars, its proper motion information (precession), and the use of the Julian 1991.25 epoch. Having a larger database meant a more comprehensive listing of the sky as compared to a guide star catalog, and allowed the flexibility to operate in any part of the night sky given a star intensity value. The main Hipparcos star catalog was retrieved from ESA's website [55].

Using the star precession information, right ascension and declination positions of the stars were updated from the J1991.25 epoch to correspond to the year 2012. This was implemented through:

$$RA_U = RA_i + \left(\frac{pm_{RA}}{1000 * 3600} \right) \Delta t \qquad (3.1)$$

and,

$$DEC_U = DEC_i + \left(\frac{pm_{DEC}}{1000 * 3600} \right) \Delta t \qquad (3.2)$$

These terms are defined as:

- RA_U – Updated right ascension coordinate of star from catalog in new epoch (mas/yr^2)
- RA_i – Right ascension coordinate of catalog star at epoch J1991.25 (mas/yr^2)
- pm_{RA} – Proper motion of RA_i (mas/yr^2)
- Δt – Time difference between epochs in fractional years
- DEC_U – Updated declination coordinate of star from catalog in new epoch (mas/yr^2)
- DEC_i – Declination coordinate of catalog star at epoch J1991.25 (mas/yr^2)
- pm_{DEC} – Proper motion of DEC_i (mas/yr^2)

The coordinate positions of the main Hipparcos catalog were replaced with the coordinate positions of the new epoch in a separate database. This new updated catalog is called the *Reference Catalog* (R.C.).

2. Magnitude dependent Sub-Catalogs

From this reference catalog any number of magnitude reduced sub-catalogs can be created by truncating the database based on the magnitude intensity of the star field desired and maintaining all stars of equal or brighter intensity. From the reference catalog several sub-catalogs were created by truncating the R.C. according to star intensity based on a desired magnitude threshold. These sub-catalogs contained solely the Hipparcos identification value (index number) of each star, magnitude intensities, and position in right ascension and declination in degrees (see Table 2.2 for Hipparcos formatting). The truncation used in this analysis was 3, 3.5, and 4 magnitude star fields.

B. Feature Lists

These sub-catalogs were used to create a new database called a *Feature List* (F.L.). A feature list database is an organized collection of *patterns* comprised of grouped *features*, where features are the individual angular displacements between stars or their vectors. Patterns are groups of stars with their associated features compiled from the sub-catalogs based on the individual characteristics of the identification algorithm they are to be used in. Figure 3.2 shows an example of 3 separate features between 3 stars.

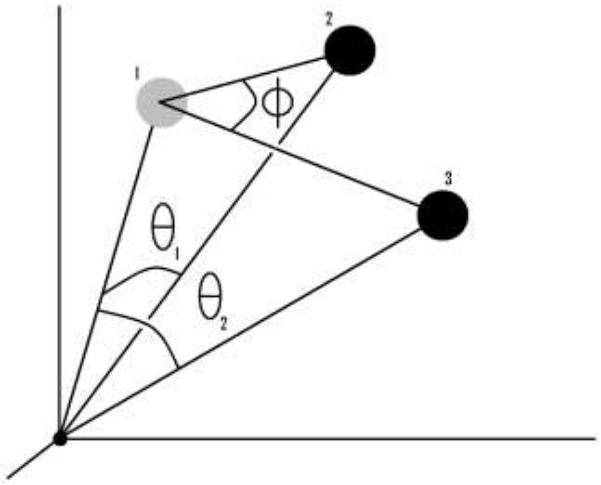

Figure 3.2 Illustration of two displacement features (θ_1, θ_2), and an interior feature φ, all 3 stars and 3 features make 1 pattern.

The identification algorithms directly compare patterns generated from the spots in the image to the patterns contained in the F.L. This is a similar process to what was used by Leibe (see Chapter 2II.F) and Samaan (see Chapter 2II.K).

1. Feature List Organization

An important aspect of these feature lists is their organization. The feature lists created are directly correlated to the type of identification method used and are individual to each algorithm, thus the number of stars in a pattern and the number of features will differ; however, similarities exist in the organization of the features where each pattern is structured with the stars listed first using their Hipparcos values, then the angular distance features (primary features), followed by the interior angle features (secondary features). The pattern is re-arranged to place the primary features in ascending order, which also realigns the star identification values that they may continue to correlate to their individual features. The patterns in the feature lists are then arranged in ascending order of the first feature of each pattern. An example of how this looks is shown in Table 3.1.

Table 3.1 General example of a feature list, showing stars and features arranged in patterns and the order of the features.

Stars in Pattern				**Features**							
Star 1	*Star 2*	…	*Star b*	θ_1	θ_2	…	θ_P	Φ_1	Φ_2	…	Φ_S
Hip 1	Hip 2	…	Hip b	$\theta_{1,1}$	$\theta_{1,2}$	…	$\theta_{1,P}$	$\phi_{1,1}$	$\phi_{1,2}$	…	$\phi_{1,S}$
Hip 1	Hip 2	…	Hip b+1	$\theta_{2,1}$	$\theta_{2,2}$	…	$\theta_{2,P}$	$\phi_{2,1}$	$\phi_{2,2}$	…	$\phi_{2,S}$
⋮	⋮	⋮	⋮	⋮	⋮	…	⋮	⋮	⋮	…	⋮
Hip n-b+1	Hip n-b+2	…	Hip n	$\theta_{p,1}$	$\theta_{p,2}$	…	$\theta_{p,P}$	$\phi_{p,1}$	$\phi_{p,2}$	…	$\phi_{p,S}$

Table 3.1 shows the patterns as they are found in the feature lists and equations 3.3 and 3.4 show how the features are organized, where:

$$n > P \geq p \geq S \tag{3.3}$$

$$\theta_{1,1} < \theta_{1,2} < \ldots < \theta_{1,P} \text{ and } \theta_{1,1} < \theta_{2,1} < \ldots < \theta_{p,1} \tag{3.4}$$

where the terms are defined as:

- n – number of stars in the FOV
- b – number of stars in the pattern
- P – number of primary features desired
- S – number of secondary features desired
- p – number of patterns created

The interior angles do not need to be repositioned in ascending or descending order relative to each other due to the intrinsic nature of all the angles in a pattern being directly correlated to one another.

2. Feature List Truncation

From the previous sub-section, the number of stars in the FOV will greatly influence the number of patterns in the feature list, and hence the overall bit size of the database and identification speed.

The feature lists are highly dependent on the magnitude threshold of the imager, which determines the number of stars or spots in the image. As seen in Figure 3.3, the number of stars exponentially increases with dimming magnitude intensity.

By examining the sky at varying FOV sizes, the minimum expected number of stars seen in an image exponentially increases with magnitude. Shown in Figure 3.4 is the minimum number of stars expected in any image for the entire sky with three different FOV sizes. It can be seen that for an imager of 50^{o} FOV the minimum number of stars the imager will see at a magnitude of 4 or dimmer will be near 40. It is not reasonable for star identification algorithms to attempt processing of so many stars.

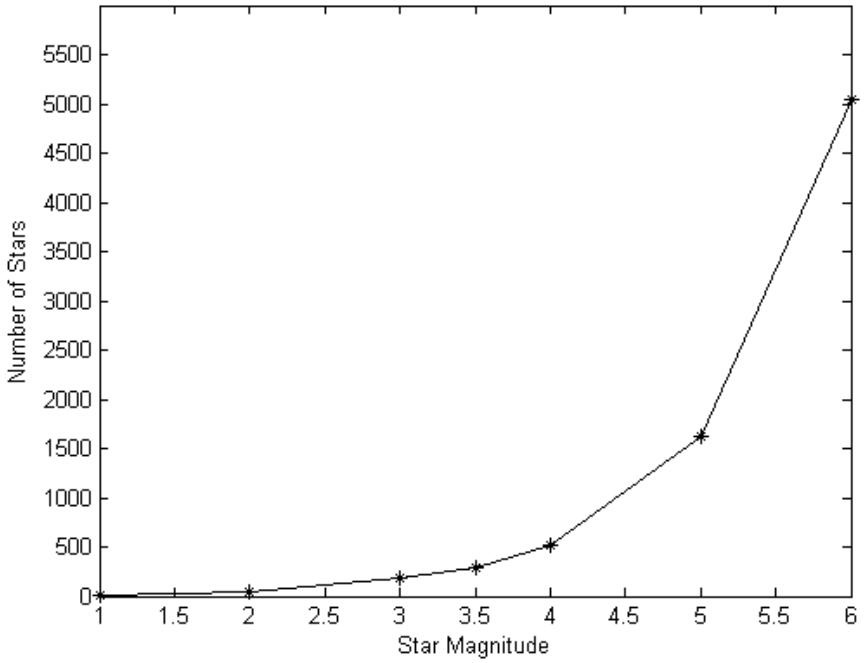

Figure 3.3 Depiction of number of stars in night sky based on magnitude

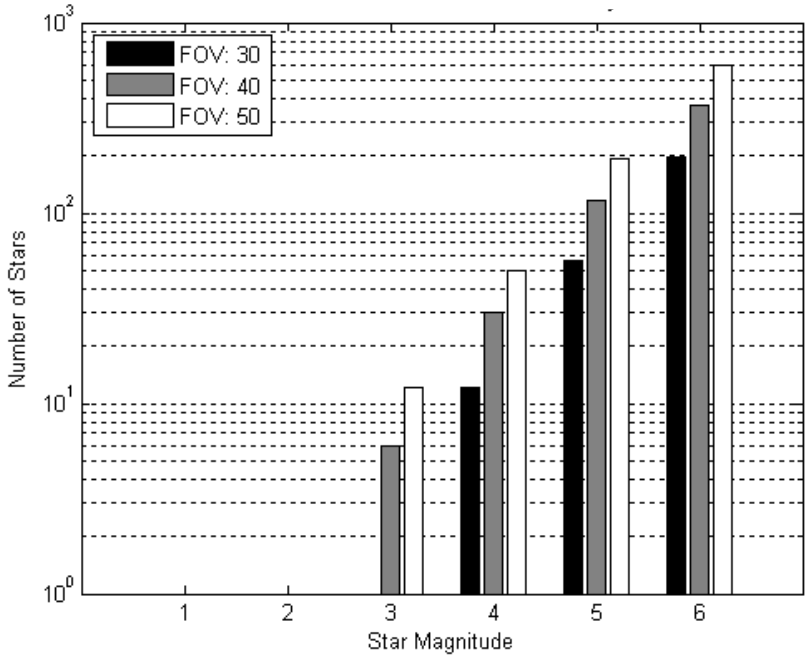

Figure 3.4 Minimum number of stars in FOV based on star intensity and FOV of an imager

With an imager, the number of observable stars (image spots) can be reduced by refinement of the pixels and centroiding [56],but for the feature lists, which are derived from the sub-catalog and whose patterns are proportional to the number of stars, the number of features can be excessive.

To reduce the feature list size and increase processing speed without removing stars from the sub-catalogs, the F.L.'s were truncated based on the FOV. Patterns were then created within a sub-grid of the FOV by using equation 3.5.

$$R_{FOV} = \begin{cases} \dfrac{FOV}{1} & 1 \le MT \le 3 \\ \dfrac{FOV}{2} & 3 < MT < 4 \\ \dfrac{FOV}{3} & 4 \le MT \le 5 \end{cases} \qquad (3.5)$$

where *FOV* is the field of view of the imager, *MT* is the magnitude threshold, and R_{FOV} is the reduced FOV for feature development. This reduced FOV can be better understood visually in Figure 3.5.

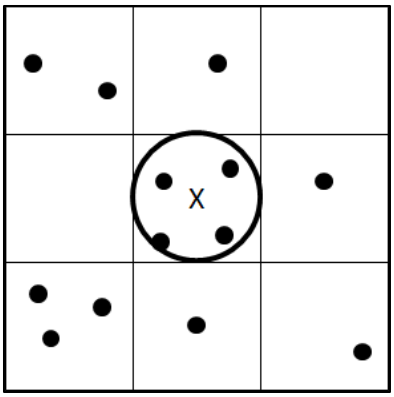

Figure 3.5 Example FOV in grid form for MT = 4, showing circular reduced FOV in center grid

To create features and patterns, a star is chosen (X) from the sub-catalog. The FOV (all 9 squares in the figure above) is centered on this star and is divided into grids, each the size of R_{FOV}. Only the stars that fit within one grid size of X are used to create patterns. Once all necessary patterns are calculated between the stars and star X, the next star is selected as X; the FOV is re-centered on that star, and again all necessary patterns are calculated.

By performing this intermediate step using equation 3.5 to reduce the FOV of the imager, the feature lists will contain fewer patterns in the database and star algorithms will require less iteration through the F.L. to obtain a star identification. However, this decreases the ability to obtain large numbers of fixed points for verification.

C. Image Spots to Spot List

The next phase in identification development was to obtain usable star information from the imager to input to the identification algorithms. The way in which these images were obtained is explained in detail in Chapters 5 and 6.

For experimental data, each image contained locations of illuminated pixels which were centroided and converted to 3-dimensional unit vectors [56]. These vectors were then indexed and called spots. A spot can be defined as being an observable star or noise on the pixel plane. For simulation data all the image data exists as 3-dimensional unit vectors. These spots are placed in a temporary database called a *Spot List* and are referenced by their index number. This spot list is removed and recreated with the instance of a new image. These spot lists may contain false star spikes, yet when passed to the identification methods the algorithms will not know which are false prior to identification. A simple example of how the spot list appears is shown in Table 3.2.

Table 3.2 Spot list format

Spot Index	X	Y	Z
1	-0.0386	-0.145	0.9887
2	0.1115	-0.0276	0.9934
3	0.0071	-0.0152	0.9999
4	-0.058	0.0299	0.9979
5	-0.0624	0.1241	0.9903
⋮	⋮	⋮	⋮
n	-0.0741	0.2006	0.9769

II. Development

This section describes the process of constructing the star identification algorithms, processing of spot data, verification techniques and voting algorithm.

A. Spot Processing and Verification

It was mentioned earlier in Chapter 2 that the three main star methods used will be the Liebe Lost in Space algorithm, Mortari's Pyramid algorithm, and Tichy's Two Star method. Of these, five variations were created by varying techniques in feature retrieval and validation. With each method a type of spot processing and verification processing are mentioned. The star processing refers to the manner in which the algorithm forms patterns from the spots and is done prior to the identification process. The verification processing possesses three groups: None, External, Internal. These are done as the last step in spot identification.

B. Spot Processing

1. Basic Processing

A basic type of spot processing is defined as an algorithm that forms spot patterns using a partial amount of spots relative to any given spot in the image. This is expressive of Liebe's and Mortari's methods which use a central spot and two neighboring spots, or use only a fixed amount of the available spots in the image.

2. Comprehensive Processing

A comprehensive type defines an algorithm in which all possible patterns are created with all usable spots in an image.

C. Verification Groups

1. Internal Verification

Internal verification uses a subroutine in the main program of the identification algorithm which compares the solution to a spot in an image to the solutions of the other spots it has already analyzed. There is no other database used to compare results and no other function calls.

Internal verification can also refer to the use of additional spots in the image to verify the condition of a group of spots, such as is used in the Pyramid algorithm. The group of spots is the primary focus of the algorithm, which obtains all information about that group, but then uses one or two other spots as a means of comparison to ensure that this group is within the correct quadrant of space. Internal uses a system of *Tagging* spot results and returning the most likely candidate solution.

Tagging is the process of obtaining a solution to a spot based on a pattern as compared to the feature list. From one pattern a spot might have a value of Hip 1, but from another pattern the exact same spot may be given a value of Hip 2. The most frequent value of that spot is tagged as the correct solution.

2. External Verification

External verification requires the use of multiple catalogs and additional subroutines or external functions as supplements to the main function of the identification algorithm. With external verification, the groups of spots are measured against both the feature list and the main Hipparcos catalog. The method used in this study is a Voting technique that utilizes a series of steps to compare patterns versus the feature list, obtain Hipparcos values for each spot, then contrasts these spot solutions to the sub-catalog to ensure that the results given are in the same sector of space, thus providing a cross-check against incorrect solutions that might of escaped earlier. External uses both tagging and votes to analyze spot solutions.

The tagging of spot values is first; then during the verification phase, each spot value is compared to the next spot in the list and against the sub-catalog. If the two spots correspond to each other, each is given a positive vote. If the two spots do not match with what the sub-catalog contains, then they are given a negative vote.

3. Voting Algorithm

Due to the extensiveness of the voting algorithm, and the benefit it provides in verification and validation, special mention is made here in regards to its usage and development.

The Voting Algorithm is a three stage process by which the patterns created in the Star Identification Algorithms are compared to the feature lists of each method, matches are recorded (see Chapter 4I), all possible Hipparcos numbers for a spot are listed and identified, and the final identification is voted and verified against the sub-catalog.

By implementing the rigor of a Voting Algorithm, any pattern of 2, 3, or 4 stars can be meticulously analyzed and verified using unaltered data given by the Hipparcos sub-catalog and will provide a higher degree of confidence in solution identification and accuracy. For consideration, the Voting Algorithm when in doubt concerning the identity of a spot on an image is able to gauge the probability of a spot's various solutions and verify against all other spots in the image. If there arises two or more solutions of equal votes, then the Voting Algorithm reports that there is no unique solution for that singular spot and it is rejected without inhibiting or harming the solutions of the remaining spots.

a. Stage 1: Pattern Matching

The Voting process begins with gathering all patterns developed by the preceding identification routine, such as the Comprehensive Triad - Brätt Algorithm (sec. III.E pg. 42), and compares all features in the pattern to the feature list, and obtains an index, or indices, of the locations of all possible matches for that pattern. These indices are gathered and a list of all possible Hipparcos numbers that match the spots in the pattern are retained until Stage 2 of the program, as shown in Figure 3.6.

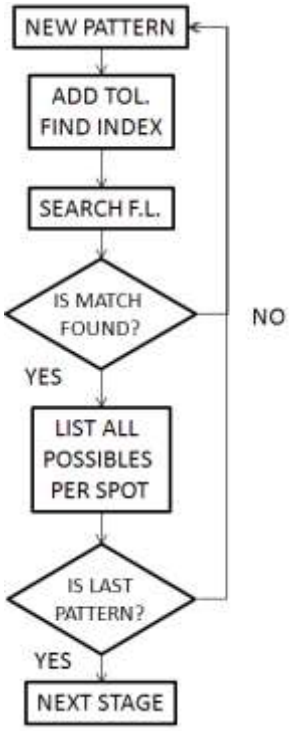

Figure 3.6 First stage of voting listing of all possible pattern matches in a feature list

During the second block in the previous figure, the addition of the catalog search tolerance can severely impact the results of the identification and voting. As the bounds for searching the Hipparcos database increase, the number of possible matches for each pattern is also increased, thus augmenting the number of possible solutions for an image spot and the probability of identification error. More will be mentioned on this searching criterion in Chapter 4I.C.2.

b. Stage 2: Spot Identification

With the list of Hipparcos numbers associated per spot, the second stage gathers all the unique Hipparcos stars and TAGS the number of instances the Hipparcos star is found for a singular spot. An example of this would look like the following (Tables 3.3 and 3.4):

Table 3.3 Example of Hipparcos numbers found for a single spot

Pattern #	1	2	3	4	5
Hip ID for spot *n*	744	51585	744	744	6123

Table 3.4 Example of tagged Hipparcos numbers and identified result

Tags	Hip ID
3	***744***
1	6123
1	51585

From the two tables above (Tables 3.3 and 3.4), assuming the program is processing spot *n* of the image, then all the star values found in the feature list for each of the patterns where spot *n* is located are tagged and the value with the most tags is found to coincide with the Hipparcos number 744. This process continues until all spots in the image are identified, shown in Figure 3.7.

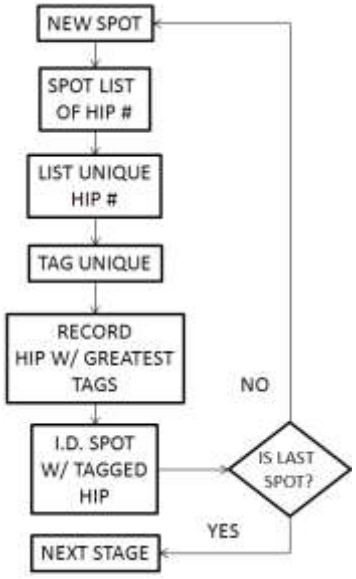

Figure 3.7 Second stage of voting where Hipparcos numbers tagged and identification of spots

c. Stage 3: Voting Verification

Once Stage 2 concludes the identification process, the imaged spots are prepared for final verification. Each spot in turn is taken and the dot product between it and the other spots is calculated. With this calculation, Stage 3 also retrieves the dot product between the Hipparcos ID's that were listed in Stage 2. If the angles (or features) between the spots and between the Hipparcos sub-catalog ID's match, within the given tolerance, then a positive vote is given to both spots, else a negative vote is given to the secondary spot being used for the analysis. These votes can be weighted to adjust for various camera aspects.

At the end of calculating votes, if the overall vote for a spot is greater than zero, it is recorded and the Hipparcos star matched in Stage 2 is passed as the acceptable solution to the spot; else if the vote was negative, or if there was no unique Hipparcos star matched in Stage 2 to a spot, then the spot is passed with a zero for the Hipparcos solution.

This final stage, seen in Figure 3.8, removes any possible doubts as to the identity of a spot in the camera image and validates it against the possibility of two identified spots having solutions outside the quadrant of space in view. The benefit to this approach is the use of two independent databases of stars, though the disadvantage is the additional processing time required for the operation.

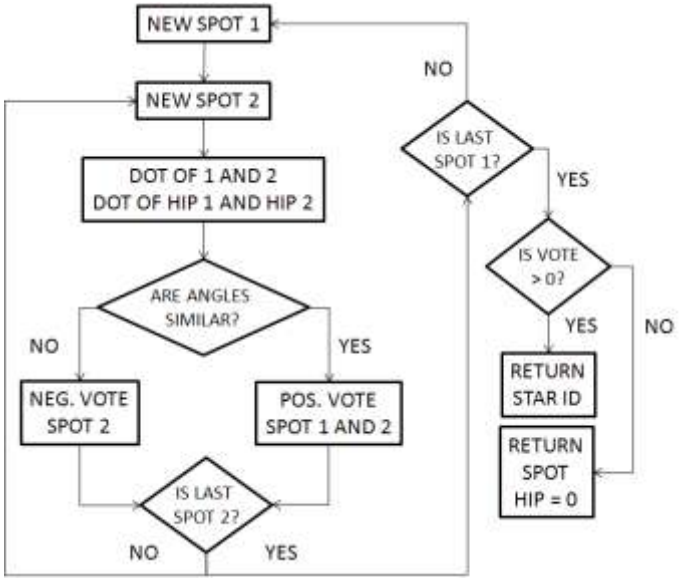

Figure 3.8 Final stage: Verification of identified spots against Hipparcos numbers found in catalog database

III. Implementation of Star Identifications

Of the methods discussed in Chapter 2, the two that have been the most tested with respect to space flight have been Liebe's Lost in Space [42], [57], [58], and Mortari's Pyramid [39],[47],[48],[50],[59] algorithms. It was decided to use these two as the basis of study based on their feature design using angular distances between observed stars (image spots), which will be measured as 3-dimensional unit vectors, and the use of planar angles (i.e. interior angles). The Two Star Algorithm by Tichy [54] which contained a useful verification sub-program that was motivated by Kolomenkin's voting scheme [53] will also be used because of its additional robustness to false spots.

Variants of these methods will be created by modifying the code slightly to build new algorithms for a case study. With Liebe's and Mortari's methods being so rigorously examined in the past, these will provide a firm foundation for developing Star Identification Algorithm testing in addition to using Kolomenkin's voting scheme to validate star identities.

A. Method Permutation Overview

Listed below in Table 3.5 are the LISA's with the number of features they create in each pattern, their processing type, and verification type. Shown as well is the general output from the identification algorithms as would be given to the spacecraft.

Table 3.5 Overview of LISA methods and permutations

		Star Processing		Verification Processing		
Method	**# of features**	*Basic*	*Comprehensive*	*None*	*Voting*	*Internal*
Two Star	1		X		X	
Liebe	3	X		X		
Liebe Voting	3	X			X	
Brätt	3		X		X	
Pyramid	6	X				X
Comp. Pyramid	6		X			X
Mod. Pyramid	6		X			X
Pyramid Voting	6	X			X	

From Table 3.5 it can be seen that several more permutations exist and would be useful for further research. Again it is mentioned that all the identification methods receive 3-dimensional unit vectors given by an image as a variable and return an array listing the number of tags or votes per spot, the Hipparcos value obtained, and the 3-dimensional vector location of that spot in the image, as seen in Table 3.6. The methods do not accept 2-dimensional vectors at this time.

Table 3.6 Example of end result unit vector output of a star ID method

Votes	Spot	Hip ID	X	Y	Z
15	1	46733	-0.19671	-0.29959	0.93357
15	2	48319	-0.11745	-0.29103	0.94948
-5	3	5663	0.1713	-0.29926	0.93867
⋮	⋮	⋮	⋮	⋮	⋮
-5	n-1	0	-0.42261	0.13457	0.89626
-2	n	0	-0.28967	0.36942	0.88296

B. Two Star Dot-Product with Voting Algorithm

This method was previously discussed in Chapter 2II.O where it was defined as being dependent on a-priori attitude knowledge. This was modified to disregard initial attitude information and act as a LISA (Lost in Space Algorithm). This was done to provide comparative results with Liebe's and Mortari's algorithms which use no prior attitude information.

Additionally, the algorithm's indexing and array matching routines were modified using improved programming techniques to speed the identification process and maintain consistence across programs. The voting algorithm that was used was also modified to accept 3 star inputs. This method uses comprehensive spot evaluation and external verification.

C. Liebe's Lost in Space Algorithm

Liebe's method has been discussed earlier in Chapter 2II.F and the general ideology outlined. The algorithm uses a basic processing style and no internal or external verification, solely tagging when comparing to the feature list. Furthermore, the algorithm uses no truncation when generating the feature list; the fact that Liebe chose only the nearest two stars to a central spot automatically shortens the database. The model of the method is shown in Figure 3.9, with Figure 3.10 demonstrating Liebe's feature creation. Furthermore, it must be noted that in Liebe's analysis he uses a FOV range of 8 to 36 degrees. This research will use a FOV of 50 degrees. Liebe uses basic star processing and internal verification.

D. Modified Liebe Algorithm (Inclusion of Voting)

This algorithm uses the methodology of three spots and three features per pattern and again chooses only the nearest two neighboring spots to a given Central-Star, as in the original Liebe, thus maintaining a basic processing type. But rather than assuming that the two stars nearest to the Central-Star are correct, the method uses an external verification to 'vote' if the obtained solution is physically possible or reasonable. The advantage is in using the magnitude reduced sub-catalog as an additional resource for verification.

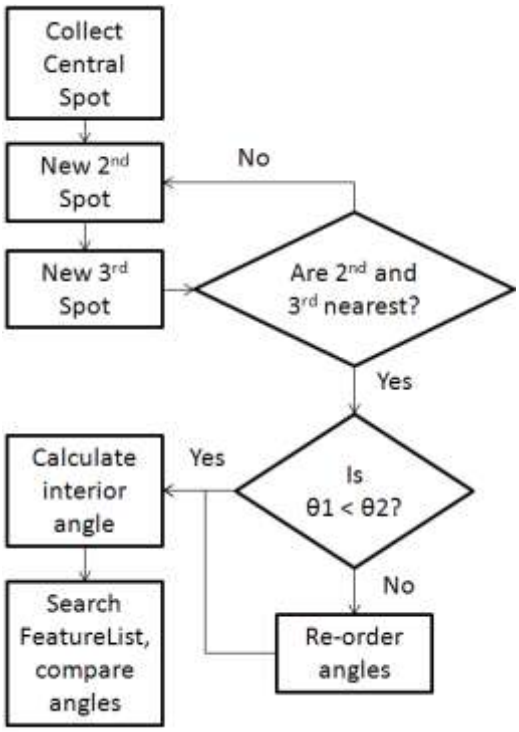

Figure 3.9 Basic flow diagram of Liebe's method

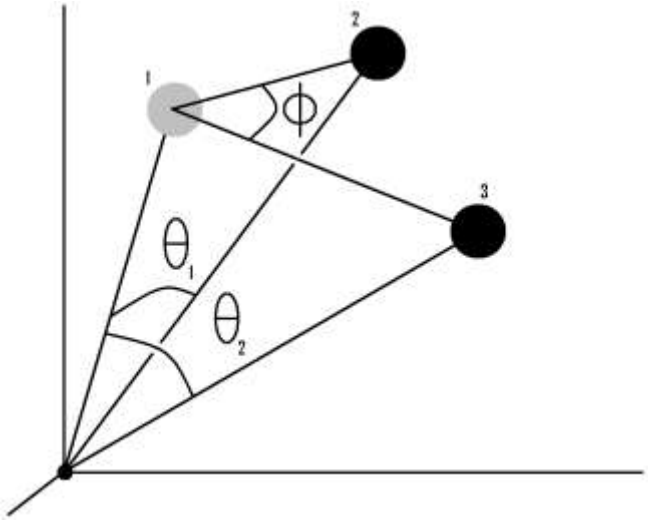

Figure 3.10 Liebe's method in 3 dimensions showing star 1 as the Central-Star with θ_1 and θ_2 as the primary features and ϕ as the Interior (secondary) angle

Where the feature list is used to measure and compare patterns, the Voting Algorithm (see Chapter 2II.N) verifies the solutions found are within the given criteria set forth by the search tolerance, which will be discussed in detail in Chapter 4I.C.2. Figure 3.11 shows the pattern comparison of the Liebe with Voting algorithm.

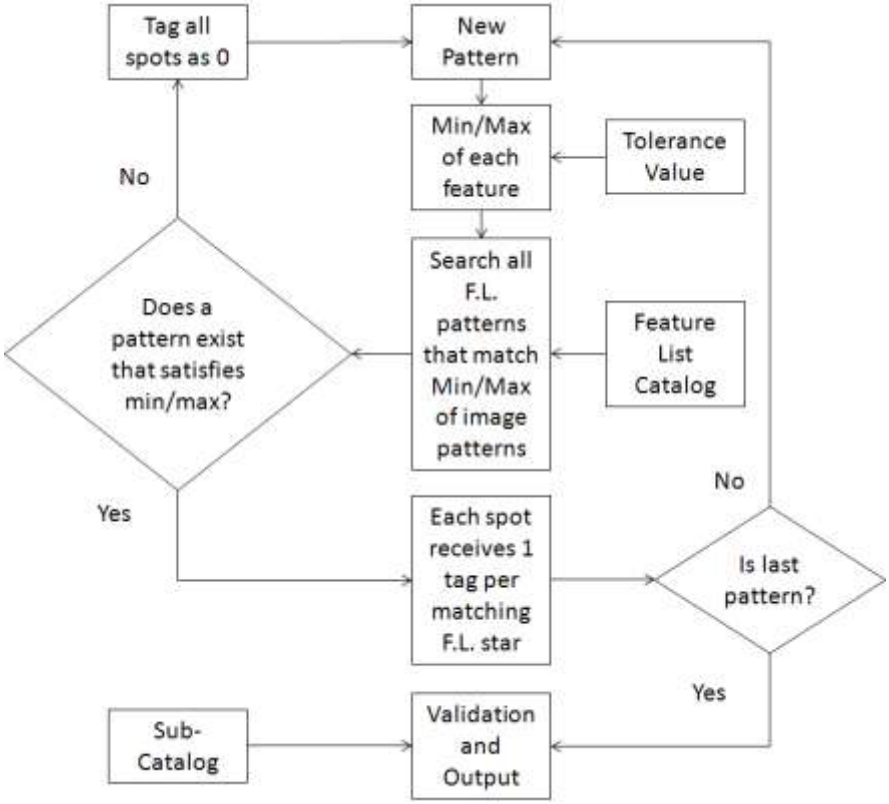

Figure 3.11 Pattern comparison to feature list database

The voting method is an added redundancy against false identification. It is anticipated that the accuracy and confidence of the identification solution will be superior to that of the Lost in Space method by itself, yet it is still subject to false identifications of stars due to its limited approach to image spot processing. The flow diagram of this method is the same as Figure 3.9 and the Search-FeatureList-compare-angles block of Figure 3.9 can be expanded to show how the features are compared to the feature list and sub-catalog, as seen in Figure 3.11.

E. Comprehensive Triad with Voting - Brätt's Algorithm

With the methodologies and strategies learned from constructing Liebe's method, and the notion of voting, the Comprehensive Triad with Voting was assembled using all possible combinations of the spots found in the image and organizing them according to the smallest primary angular feature, θ (Figure 3.12).

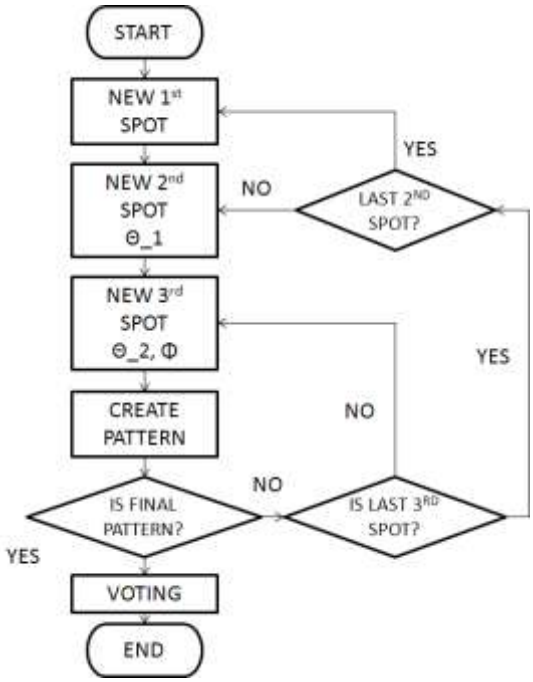

Figure 3.12 Logical flow diagram of Brätt Algorithm

The flow diagram shown above shows the Voting Algorithm as its own block in the structure of the algorithm. The organization of the patterns according to the smallest θ is simple, yet critical to the accuracy of the algorithm. The feature list for this method is unique compared to the feature lists of Liebe and the Two Star method. Where the feature list of Liebe is also ordered on θ, the number of patterns created is limited; however, the feature list for this algorithm contains all possible patterns between stars in a given FOV. This allows for a far greater approach in verifying the identity of the spots in an image. The use of a comprehensive processing and external verification is presumed to obtain a more precise solution.

F. Constrained Pyramid Algorithm

In Chapter 2II.J the Pyramid algorithm was presented. It is stated here that this is not the true Pyramid algorithm, but an interpretive construction based on the information provided by Spratling [39]. The original uses a k-vector search technique [48] that is not used in this research. A full analysis of the behavior and characteristics of the true Pyramid algorithm cannot be provided, but a likened star processing and form of verification used by the Pyramid algorithm can be compared to the other algorithms tested.

Unique qualities that have been changed to the Constrained Pyramid algorithm are its termination process (Figure 3.13), internal validation using a fourth image spot, and intensive feature evaluation.

Rather than attempting to identify each spot in the entire image, the algorithm will establish one *Pyramid* of 4 spots at a time, and if a Pyramid is successful in matching with known stars in the feature list, it will identify solely those spots and terminate the program, giving only four star identifications and positions as a result.

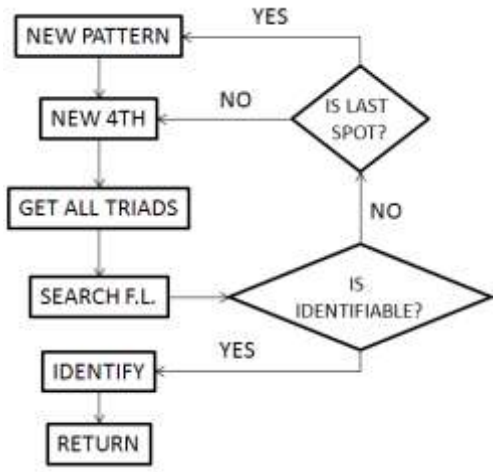

Figure 3.13 Logical flow diagram of Constrained Pyramid algorithm

This early return procedure reduces the time required to identify the image, and with four stars it would not be difficult to obtain a quaternion solution, however, the quaternion would not be as precise as one given by an identification method that could evaluate all observed stars correctly in the entire image. Additionally, if all four spots selected for a Pyramid are false but similar to a pattern in the feature list, it would return an erroneous attitude solution.

The internal validation using a fourth spot in the image has been discussed, but with it comes the possibility of increasing the number of evaluative features. Mortari states that his algorithm uses only 6 features (Figure 3.14) for identification out of a possible 24. These additional features are used in this analysis to further affirm and correlate image patterns to the feature list as his implementation of verification is unknown. These 24 features, after tolerances are applied, create an increased constraint on spot orientation displacement during identification, thus the name of the method.

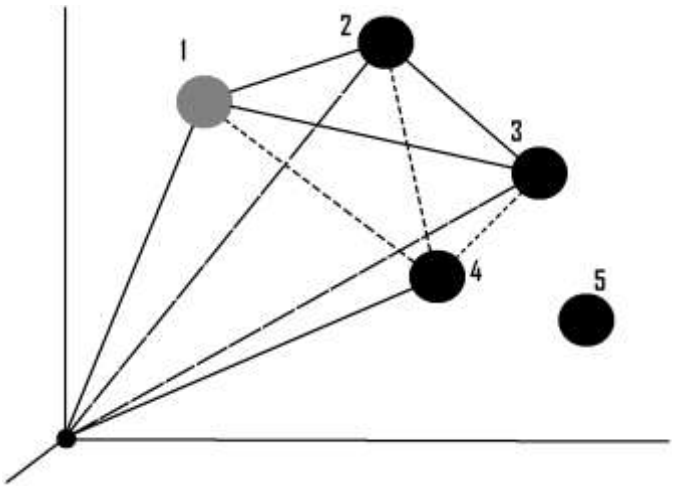

Figure 3.14 Depiction of Pyramid pattern creation where spot 1 is the apex, and spots 4 and 5 are the next '4th spot' for verification consideration

The k-vector searching enhances the ability for an identification method to search through the feature list quickly, yet it was desired to use a more simplistic searching method as the primary focus is to compare solution ability between different algorithms and not so much the ability to search through the feature lists quickly. The k-vectoring technique could be used on any of the algorithms in a future work.

G. Comprehensive Pyramid Algorithm

If one spot could be used to validate the image with three spots comprising the base, then it might be possible to use additional spots to validate. Because of this desire and motivation to evaluate all spots in a given image, the Comprehensive Pyramid and Modified Pyramid (sec. H) algorithms were created.

The Comprehensive Pyramid follows the same 24 feature requirement and 4-spot Pyramiding as the Constrained Pyramid. It differs by continuing the spot search and analysis of the image as a comprehensive

type processing, shown in Figure 3.15, rather than terminating with the first instance of a solution. This was done to test the robustness of the Pyramid program and assess if it could be used to identify all spots given the added constraints. This is a recursive methodology as it will overlap multiple times the spots it has already identified with new spots found. Thus, it does not provide the opportunity to validate a spot group or selection based on all the other spots in an image; however, it may find that there is a higher probability that one of the spots previously identified correlates to a different solution than previously given.

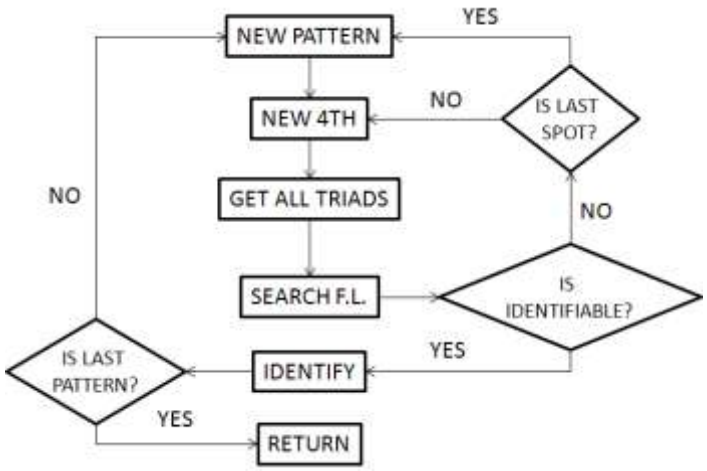

Figure 3.15 Logical flow diagram of Comprehensive Pyramid algorithm

The occurrence of an incorrect solution to a spot replacing a correct solution is highly undesirable as a correct solution may be overwritten by an incorrect assessment and never repaired. Though one would think that by recursively checking all existing spots that the solution would be validated, yet it is anticipated that this procedure might be unable to maintain hold on the previous solution to a spot and compare it to its new solution. Thus, it is a semi-validated solution, and is assumed that it will be weaker than the Constrained Pyramid algorithm itself.

H. Modified Pyramid Algorithm

Due to the high possibility of error from the recursive approach of the Comprehensive Pyramid algorithm, where it would have the ability to overwrite the correct solution to an identified spot, a second

modification to the Pyramid method was created by using the four primary spots identified in the Constrained Pyramid algorithm as the basis for all the verifications of the remaining spots.

Having identified the first four spots, the program enters a sub-function just before the return statement seen in Figure 3.15. These four identified spots will not change nor become overwritten during the process, but are used to create six additional triads using a fifth spot. In this manner all but one spot is known during the identification process. Once the 5th spot is verified, the search continues to the next spot in the sequence and performs the same operation, keeping the first four spots intact. This is shown in Figure 3.16.

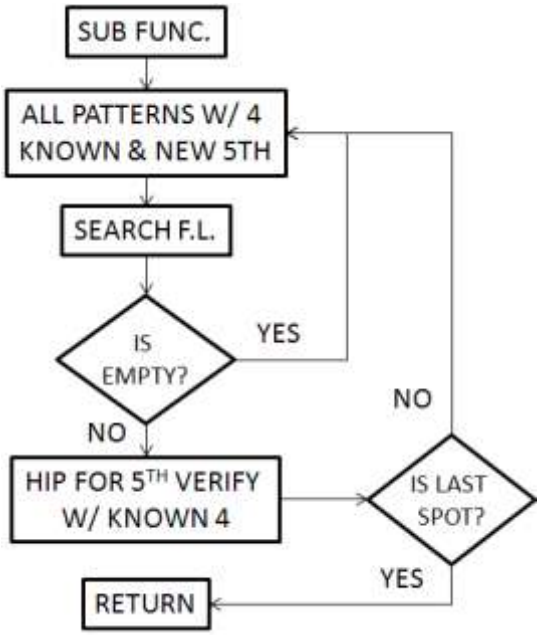

Figure 3.16 Flow diagram of Modified Pyramid algorithm

Once all spots are identified, the sub-function terminates returning the identified values of all the image spots. It is anticipated that the modification made will be far more accurate than the Comprehensive Pyramid algorithm. This method uses the same feature list as the Constrained Pyramid algorithm, however, because of this more intensive internal verification process, it is believed the algorithm will take the longest to complete.

I. Pyramid with Voting Algorithm

It was theorized that by including the same type of voting procedure as the Two Star, Liebe with Voting, and Brätt algorithms, that the functionality and accuracy of the Constrained Pyramid algorithm will be enhanced. The four spots identified by the method will be verified against the magnitude reduced catalog. This operation would be a redundant verification procedure, as the Constrained Pyramid algorithm already uses an internal verification, but it is anticipated that it may improve solution ability versus false spots. Figure 3.17 shows a basic flow diagram of the Pyramid with Voting algorithm.

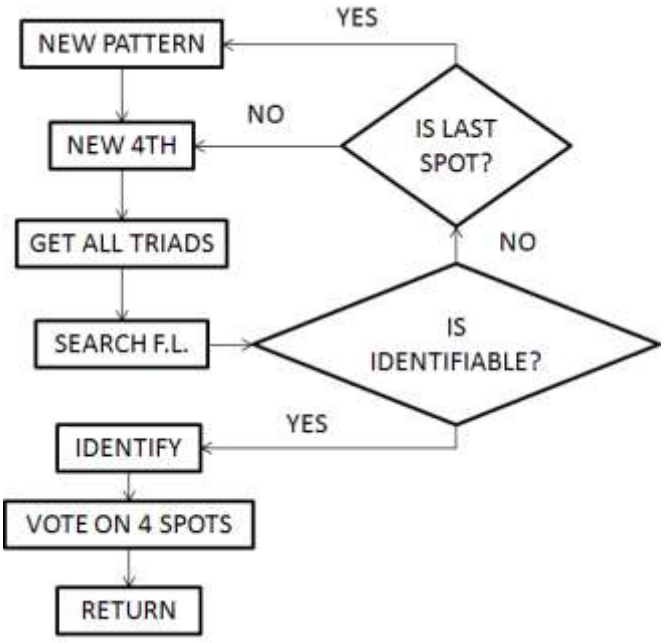

Figure 3.17 Logical flow diagram of Pyramid with Voting

This algorithm will display a solution exclusively for the four spots identified by the Constrained Pyramid algorithm and will return zeros for all other spots. Though the addition of the voting method is an additional use of processing time, it is offset by the short timeframe of gathering a single Pyramid group of stars and voting on solely those four, rather than the entire image. This method uses the same feature list as the Constrained Pyramid method.

IV. Star Camera Selection

The Aptina [28],[56], MT9P031 imaging system was selected for use in this study for its mono-color formatting which removes the issue of variable spectral intensities. This camera is comparative to the cell phone cameras mentioned in Chapter 1 at 381 mW consumption, 5MP resolution with a 50^{o} FOV, and its less than 0.5U size. It was also chosen for its 2.2 μm pixel dimensions and user controlled variable frame rates.

CHAPTER 4

TESTING CRITERIA

In this chapter the qualifications for solution acceptance are stated and described, and a premise for testing and analysis outlined. Detailed results of solution behavior of the star identification algorithms are presented in Chapters 5 and 6. Evaluation of the Lost in Space Algorithms (LISA's) is conducted outside of the main program of the identification methods.

I. Solution Evaluation

The accuracy of a solution from a given algorithm is of primary importance in this analysis. The accuracy of the identification solution is evaluated through:

- Correct/False/Empty Image Solutions
- True/False/Neutral Spot Matches
- Minimum Required Spots for Solution
- Probability of Error

A. Image and Spot Evaluation Criteria

1. Internal – Spot Match

A *Match* is the set of individual identifications found for a spot and star pair in an image and includes the number of *Tags* or *Votes* for that pair. For the simulated and experimental data used in this study, the exact identities of the observable stars (spots) in the images are known and used to measure success.

a. True Match

Success criteria: Must return a singular Hipparcos star solution for a given spot in an image that corresponds to the true identity of that spot and must receive a positive vote greater than zero, or receive a minimum of one tag.

b. False Match

Fail criteria: Contains a singular Hipparcos star solution for a given spot in an image and returns a positive vote or tag but is not the true identity of the spot in the image.

c. Neutral Match

Returns one or multiple Hipparcos solutions for a spot and returns a vote less than or equal to zero. Table 4.1 below better illustrates how these matches are formed.

Table 4.1 Example of matches

Spot	Votes	HipID Found	HipID True	Match Result
1	>0	H1	H1	TRUE
2	>0	H2	H3	FALSE
3	≤0	H4 or 0	H6	NEUTRAL
4	≤0	H5 or 0	0	NEUTRAL
5	>0	H7	0	FALSE
6	>0	0	0	TRUE

2. External – Image Identification

a. Correct Solutions

An identification method must return a solution in which no false matches have been encountered and the minimum number of true matches for obtaining a solution has been reached.

b. False Solutions

An identification method returns a solution containing any number of false matches from an image or returns fewer than the minimum required number of true matches.

c. Empty Solutions

In the course of identification, the algorithms may come across images where each spot is identified but during the algorithms' validation and verification protocol all the identifications are proven to be neutral matches. The solution in this instance is returned as a list of zeros for the identity of the spots in the image and is treated as an *Empty Solution.* In this situation the solution does not impair algorithm

suitability; instead the imager system should be designed to take another image and again attempt to identify its attitude.

B. Minimum Required Stars for Solution (MRSS)

To obtain a valid image solution, a limit was set stating that a minimum of four true matches must be acquired. This number was chosen based on the restrictions of the Pyramid algorithm described by Mortari, which required a minimum of four spots in an image to initiate the program and achieve a solution. This requirement was then passed to the additional algorithms to maintain analogous results.

C. Probability of Error

The percent failure of a solution returned by the LISA's was determined to be the most advantageous method to produce a structure for solution behavior analysis. Using the construct that a single match resulted in a complete solution failure, entire sets of images can be quickly evaluated and compared among algorithms. This solution error essentially implies a Go or No-Go qualification for an algorithm.

The averaged solution failure of each algorithm offers a means of comparison and judging of solution attainability. This probability of error is a function of:

$$PE = f\{N_f, T_{cat}, e_{cen}, MT, FOV\} \tag{4.1}$$

- N_f – Number of false spots in the image
- T_{cat} – Catalog tolerance range
- e_{cen} – Error in centroiding
- MT – Magnitude threshold
- FOV – Field of view of the star camera

Number of false spots, centroiding error, and incorrect tolerance ranges are the primary sources of error that will affect the solution of the identification methods. These will be used to determine at what point the algorithms will fail.

1. False Spots

During flight a satellite imager will encounter noise on the image plane (e.g. passing asteroids, other satellites, planets, or space debris), and may have broken or un-calibrated pixels. These sources of noise create false spots (or false star spikes) in the image plane yet do not exist in the catalog database of known stars. Incorrect identification of these false spots will adversely influence the solution of the algorithm and cause satellite course deviations.

2. Catalog Tolerance Range

This specifies the amount of variability in the angles of comparison between features in the patterns built by the algorithms and those in the Feature List database and sub-catalog. No atmospheric or lens distortion effects will be removed from experimentation with the Aptina, therefore, the *Catalog Tolerance* will be used to correct for these deviances as a means of circumventing additional process time needed for image correction and to ascertain if image perturbations could be described by a single overall error bound. Figure 4.1 visually describes how a feature, or angle, derived from a pattern is widened using the catalog search tolerance.

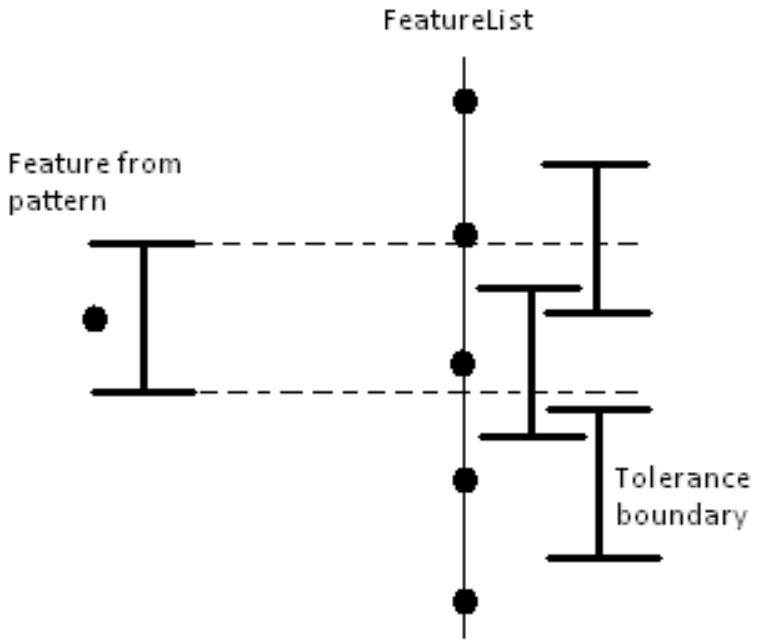

Figure 4.1 Example of tolerance bounds on a feature and overlapping of features in feature list

The same tolerance is applied to the features in the feature list, shown as large thick black I's. It can be seen that one angle in the feature list is within the boundary of the pattern angle, but because of the larger boundary, an angle just above in the feature list is also called because its boundaries overlap with the feature from the pattern. Thus, this feature in the pattern, and the spot associated to it, have two possible solutions. This further illustrates that a set range of tolerance values must exist for the algorithms, otherwise too many probable solutions could exist for any specific spot in an image.

3. Centroiding Error Range

Centroiding error is the amount of variation to be added to the centroid position of each spot. This is used solely for Simulation data. For simulating the behavior of a real imager with atmospheric conditions and pixel distortion due to lenses, heating and cooling effects, and noise, each spot will be repositioned in the image with a random angular displacement (using the Monte Carlo method) based on the selected Centroid Error inputted. Figure 4.2 shows that once a centroid error is integrated, the true position of the spot will be shifted in any direction within this boundary.

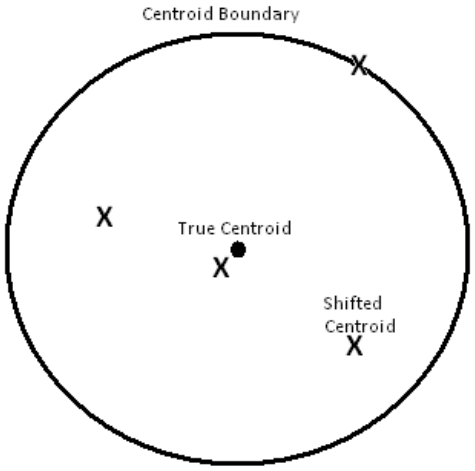

Figure 4.2 Example of an image spot centroid and the possible area of existence given a centroid error range

For example, if a centroid error of 1 μrad is selected, then each spot will be moved randomly between 0 and 1μrad in a random direction. This denotes that any two spots in an image may be at most 2 μrad of radial distance from one another.

II. Computational Considerations

One of the desires in identification systems is to obtain solutions with swiftness. Ideally, it is preferred that the algorithms perform in real-time. These Lost in Space Algorithms are not real-time solutions, yet, with the use of Kalmen filtering the time loss issue could be accounted for. The measurement of speed is a function of the size of the sub-catalog database D_{Cat}, the size of the feature list D_{Feat}, the number of spots in an image S, and the efficiency of the algorithm E.

$$Speed = f\{D_{Cat}, D_{Feat}, S, E\} \tag{4.2}$$

A. FLOPS, TIC-TOC, and Profiling

The speed with which an identification algorithm can develop a solution will be measured using the Profiling subroutine in MATLAB where the number of seconds taken from when the input was given to when the solution is outputted will be recorded. The Profiling command option will be used due to its more comprehensive analysis. Despite the fact that it includes an overhead timing to the algorithms, it was preferred rather than using TIC-TOC functioning as this will add extra lines of coding to each algorithm and cannot give detailed information regarding coding speed.

MATLAB has removed the use of FLOPS and operation counts since version 6.0, thus the ability to measure directly the efficiency through the number of floating point operations in each of the algorithms will not be calculated.

B. Algorithm Order and Feature Creation Time

The size of the Feature Lists will be recorded to determine which algorithms can search for matches the quickest. As the magnitude threshold of the system increases (e.g. 3 → 3.5 → 4) the number of possible stars in the line of sight of the imager exponentially increases, thus increasing the amount of time needed to create features and patterns, and increasing the database sizes of the Feature Lists. Below in Table 4.2 is the progression of each algorithm based on the number of stars in the FOV of the imager.

Table 4.2 Algorithm order and feature list sizes based on *n* stars in FOV

Method	Order	# of patterns in list
Two Star	$O(n^2)$	$\frac{n!}{2(n-2)!}$
Liebe	$O(n^3)$	n
Liebe Vote	$O(n^3)$	n
Brätt	$O(n^3)$	$\frac{n!}{2(n-3)!}$
Pyramid	$O(n^4)$	$\frac{n!}{6(n-3)!}$
Comp. Pyramid	$O(n^4)$	$\frac{n!}{6(n-3)!}$
Mod. Pyramid	$O(n^4)$	$\frac{n!}{6(n-3)!}$
Pyramid Vote	$O(n^4)$	$\frac{n!}{6(n-3)!}$

The Constrained Pyramid algorithm contains a return line that terminates the program after the first 4 spots in the image have been identified and verified. Thus, the pattern equation in Table 4.2 holds for the feature list database but not for the overall time-estimate of the algorithm. Since the Modified Pyramid and Pyramid with Voting algorithms are the same as the Constrained Pyramid with respect to pattern generation, with the addition of a sub-function, they follow very closely the same order of time during identification.

III. Algorithm Robustness

Robustness is the algorithm's ability to handle abnormal situations. The issues facing these algorithms can be encompassed into two primary divisions: Error Prevention and Computational Failure.

A. Error Prevention

Each algorithm has unique means of preventing possibilities of errors from entering the final solution output. Such situations of error robustness include:

- Competence to negotiate false detections in the image
- Proficiency of validating abnormal solutions prior to output
- Ability to operate despite abnormalities in input

To negotiate false detections, The Two Star, Liebe with Voting, and Brätt algorithms use the Voting method to rigidly confirm, or remove potentially invalid, identities. Additionally, the use of 3 stars for pattern creation gives the Liebe with Voting and Brätt algorithms additional competence in error prevention, rather than the use of two stars only.

The Brätt algorithm additionally constructs patterns between all spots in the image, whereas the Liebe with Voting limits the combinations of patterns to strictly the two most adjacent spots to a target, or central spot. This limitation removes the ability to verify the identity of the spots early in the second stage of the voting process, while the Brätt algorithm uses all combinations and the second stage of the voting process to enhance the verification phase, thus further removing potential inaccuracies.

The Pyramid-type algorithms allow the use of a fourth star and three additional features in a pattern as their means of error prevention. With the Pyramid algorithm terminating early with the first likely combination of four spots, the algorithm reduces the chance of encountering false spots but also reduces the ability to validate across the entire FOV of the imager.

B. Computational Failure

Computational failure refers to circumstances in which the algorithms will be halted or crash during star identification. Such instances of robustness can be measured as:

- Ability to not break down easily or not be wholly affected by a solution failure
- Negotiation of exceptional circumstances such as too many or too few spots
- Failure if no matches to spots, patterns, or features are found

Each algorithm is equipped with conditional statements to prevent empty variables from passing through the code, to deal with images with insufficient spots, and to phase out incomplete solutions.

However, due to the nature of the sub-catalog and feature list databases, if the search tolerance (sec. I.C.2) is large, then the number of possible identities for a single spot multiplies and causes the code to

become retarded in its identification process. If the number of spots in an image are also very large, the lag in solution time becomes sizeable and could cause system failure depending on the system being used.

No upper limit is set to the number of spots in an image that any of the algorithms can solve, yet from Figure 3.4 it can be seen that the issue of an overabundance of spots will not exist.

IV. Memory and Disc Space Management

Crucial to spaceflight is the compressibility of the identification programs to increase memory space for scientific instrument systems on board. As such, the desire with the Lost in Space Algorithms was to condense the codes and databases as much as possible. These algorithms were created using MATLAB functions and files. The program sizes can be further reduced once they are compiled and optimized for spaceflight.

A. Short-Term Usage

Each algorithm should have minimal RAM usage and outputs that do not overflow or processes that do not use substantial amounts of RAM. The programs were created to output directly to the command line a structured variable containing the number of corresponding votes to the star identification, the spot number, star identification value, and spot location in the image. These algorithms can be easily modified to output results to a number of various output formats such as: *.MAT, *.DAT, *.TXT, etc., which can be maintained indefinitely if desired, or replaced with the solution of the next image.

B. Long-Term Usage

The databases and actual algorithm codes will be the only permanent files that will be placed on the hard-drive of a given spacecraft. There is no other need for long-term storage access.

C. Feature Lists and Patterns

To reduce the amount of storage needed, the databases ought to be as small as possible yet well-defined and indexed to avoid solution error. As well, the main Hipparcos catalog should be truncated to include only the set of stars that the imager is able to detect.

CHAPTER 5

SIMULATION TESTING

I. Simulation Testing

To validate the criteria listed in Chapter 4, the Lost in Space Algorithms were tested against 12,000 simulated images. The simulation codes can be found in the appendix. This simulation used information based on the Aptina imager in this study.

The images were split into two magnitude threshold sets of 3 and 3.5. 100 randomized simulated camera positions were chosen using a Monte Carlo randomization. Figure 5.2 shows these random locations of the images taken using 3.5 magnitude stars and brighter. The same image for magnitude 3 stars can also be found in Appendix B. In this chapter, only the simulations conducted using 3.5 magnitude thresholding will be discussed. Additional figures and data can be found in the Appendix. A depiction of the manner in which these inputs were added is shown in Figure 5.1.

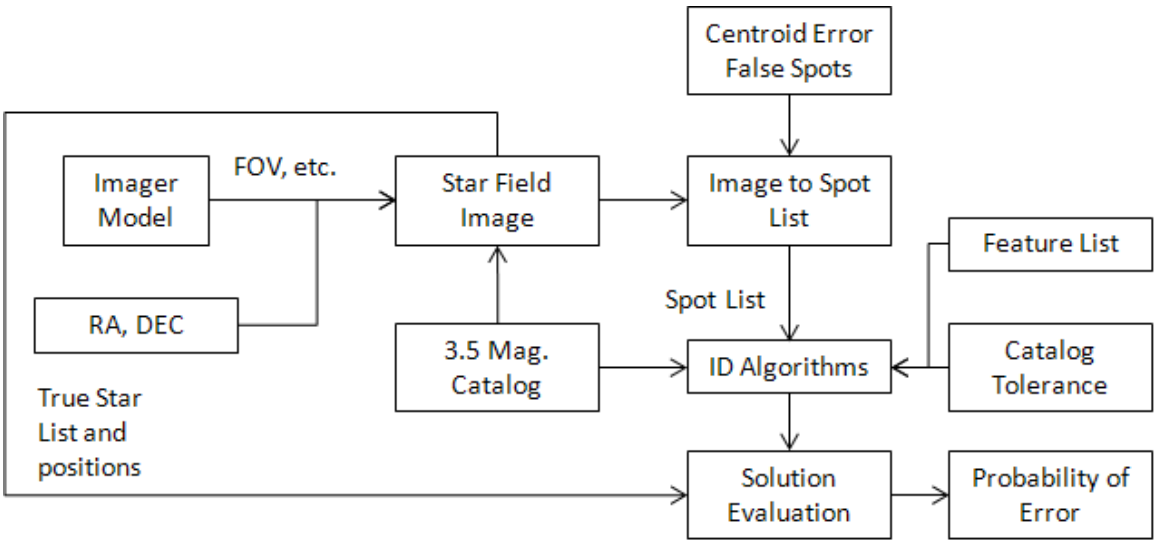

Figure 5.1 Flow diagram of simulation model with random Monte Carlo inputs

Multiple instances of catalog searching tolerances, centroiding error, and false spots were added to each random camera position. The direction (not magnitude) of centroiding error and location of false spots

were randomly inserted into the images. Centroiding error was used to simulate atmospheric and lens distortion effects that might be experienced during experimental testing.

Catalog tolerance was applied to the features in the identifications methods during pattern comparison to the feature list, expressed simply as:

$$\theta_{i,MAX} = \theta_i + T \tag{5.1}$$

$$\theta_{i,MIN} = \theta_i - T \tag{5.2}$$

where $\theta_{i,MAX}$ and $\theta_{i,MIN}$ represent the maximum and minimum values of the feature θ_i once the tolerance T has been applied. The algorithm then searches the feature list for all features between this maximum and minimum value. The results from the search are then marked as possible solutions to the spots being investigated. The catalog tolerance ranges were selected to be from 1 to 5 mrad, which corresponds to 3.44 to 17.19 arcmins of deviation. This was selected to show a worst case scenario that might be anticipated if using a cell phone camera.

Throughout the simulation process, false spots were added to the image plane at random locations also using a Monte Carlo randomization procedure. Between 0 and 3 false spots were placed at a time to all instances of a simulation image. This meant that each randomly chosen image was solved four times based on false spots: once with no false spots, once with one false spot, etc. This was done to verify whether the algorithms would be able to remove random intrusions.

This range was selected as being an appropriate number to simulate objects that may pass by the FOV of an imager in space that would cause false star spikes. With an average of 27 stars in any image, the false spots could comprise one ninth or more of the spots inputted on the simulated image plane.

With each image, the simulation solved the identity of each spot using all the identification algorithms and evaluated solution probability, processing time, precision, and percent of empty solutions. Figure 5.3 shows the resulting failed solutions of the algorithms averaged across the number of false spots and centroiding error (see Appendix B for 3D figures of solutions as functions of centroid and catalog tolerance). Figure 5.2 shows the 100 random locations used in the experiment.

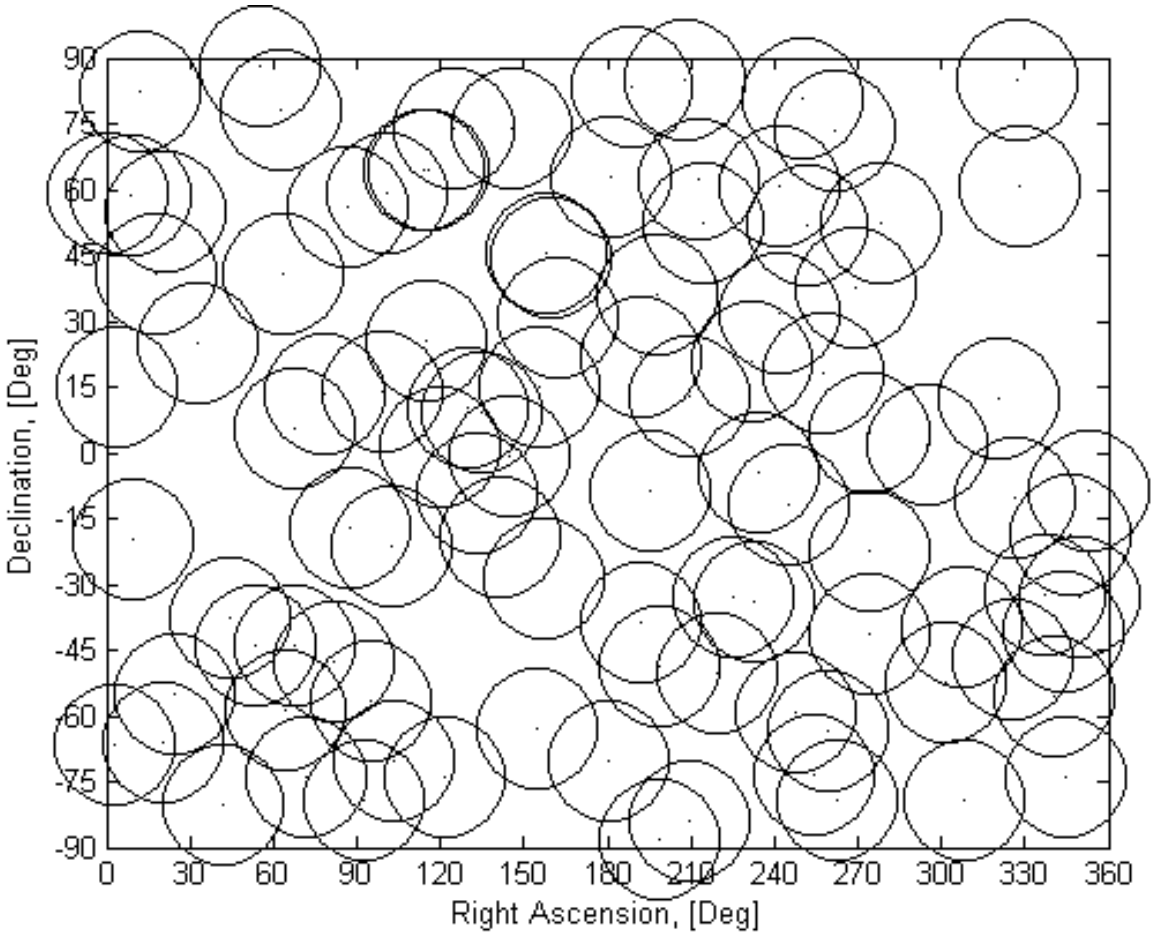

Figure 5.2 Camera positions of 100 random simulated images with approximate range of imager FOV as a Miller cylindrical projection. 3.5 mag. star field.

A. Percent Failure vs. Catalog Tolerance

Taking the incorrectly solved simulated images of three of the algorithms and averaging them against the number of false spots and against centroid errors, gives the following figure. This is the probability of error based on the searching tolerance.

To aid in the interpretation of the figures, it is stated that the plots of failed matches express a sub-division of the no-solution (or false solution) plots. The percent of false matches describes how many of the stars, on average, in an image have failed for a given solution failure.

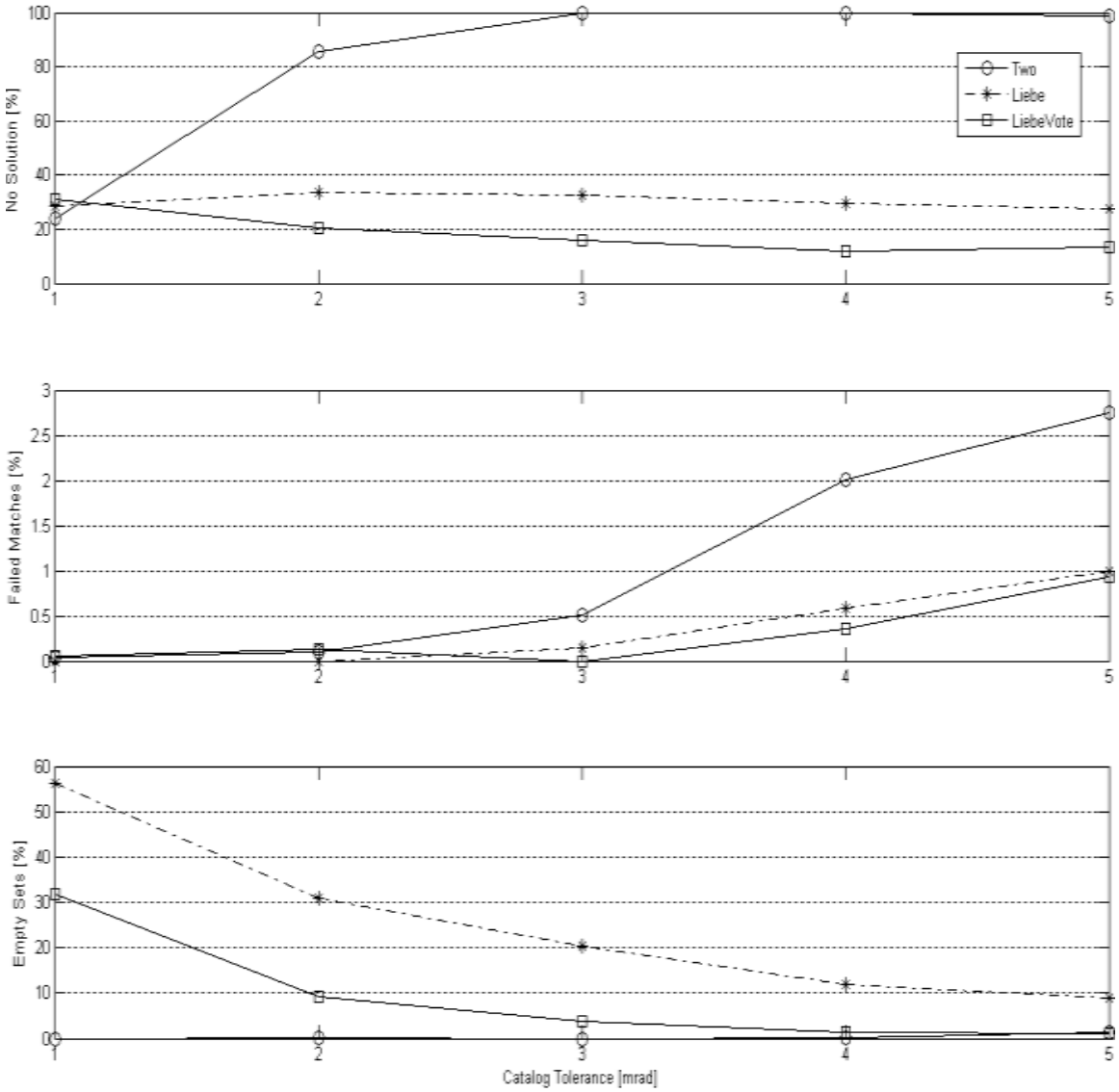

Figure 5.3 Failed ID algorithms averaged against centroiding and false spots

It is evident from Figure 5.3 to see the Two Star, Liebe, and Liebe with Voting fail severely. It can be stated that these are highly sensitive to the searching tolerance and have a high probability of failing at the given tolerance ranges. What can also be seen is the Liebe with Voting algorithm does improve the performance of the standard Liebe algorithm, both in reducing the number of failed solutions and the number of empty solutions. Figure 5.4 shows acceptable ID algorithms.

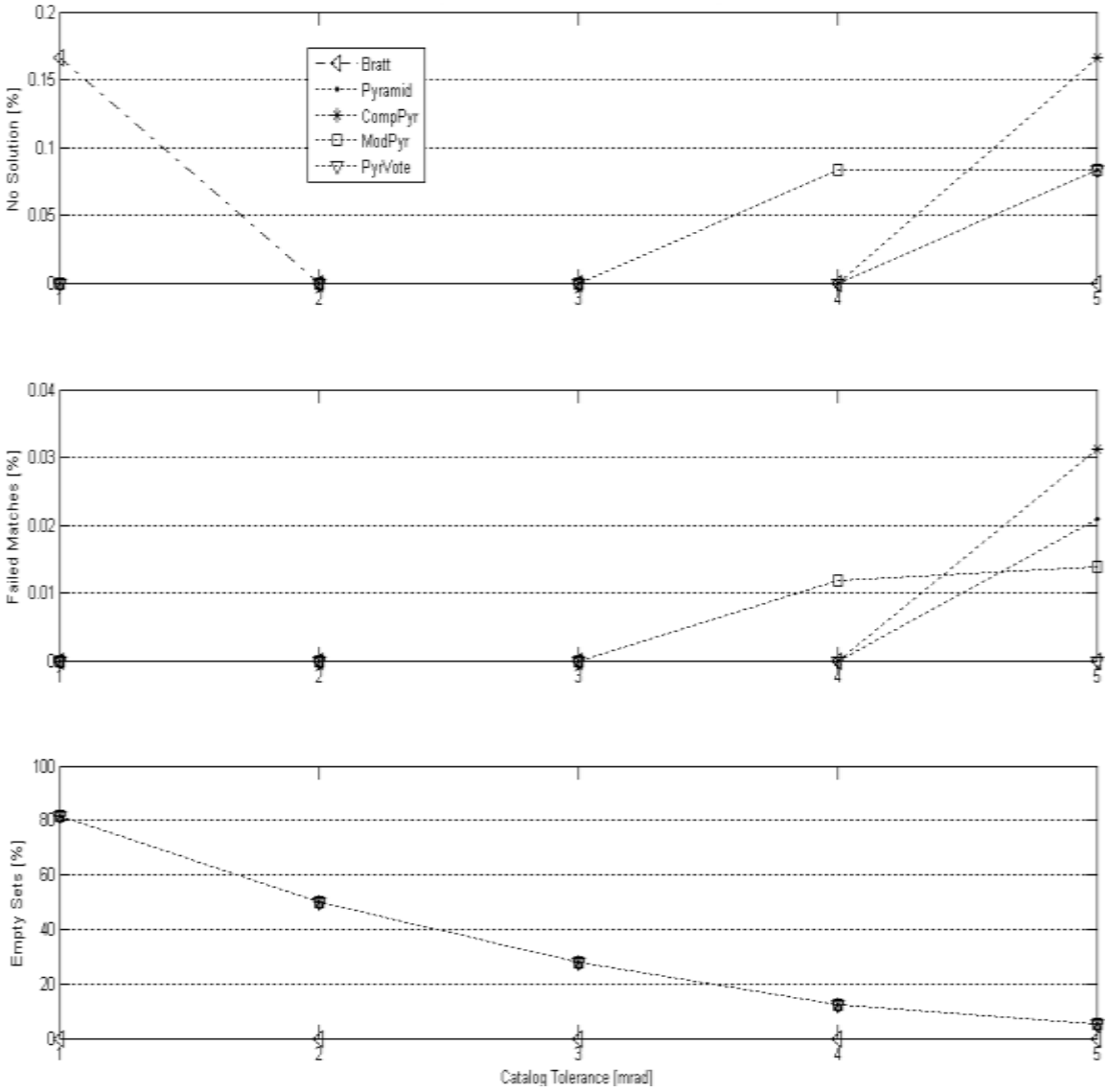

Figure 5.4 Acceptable ID algorithms averaged against centroiding and false spots for mag. 3.5 star fields

Interestingly, it appears in Figure 5.4 as though the Brätt algorithm does not fare as well with very low searching tolerances, but remains stronger than the other algorithms with increasing range. When a failed solution appears but there are no false matches, then the algorithm identified less than the MRSS spots correctly and the rest were rejected during the verification phase. This signified that there were enough spots in the image to obtain a solution; however, the solution was inadequate according to the MRSS restriction imposed.

At the 1 mrad range, the Brätt algorithm consistently obtains a minimum of 3 correct matches without encountering a false match. This however is below the standard of 4 matches that was selected based on the nature of the Pyramid type algorithms, thus the solution behavior would improve if the MRSS was set to 3, not 4.

The Constrained Pyramid and Pyramid with Voting follow each other exactly in the percent of failed solutions, but the Pyramid with Voting has far better precision (number of correct matches) than the Constrained Pyramid algorithm. Interestingly enough, the Modified Pyramid algorithm fails sooner, at 4 mrad. Surprisingly, all the Pyramid type algorithms return the same probability of an empty set of solutions for the tolerance range, which follows a decreasing quadratic trend. These empty solution sets are attributed to the highly constrained requirement for the algorithms to satisfy 24 features, rather than 6. The Brätt algorithm however never returns an empty set, showing that it returns a higher number of solutions to work with. All the algorithms have a less than 0.2% probability of image solution failure.

By looking at the average number of failed matches among the algorithms (Figure 5.5), it can be clearly seen the amount of improvement in using 3 or 4 spots in a pattern versus 2, and the benefit of using voting as a means of verification in star identification.

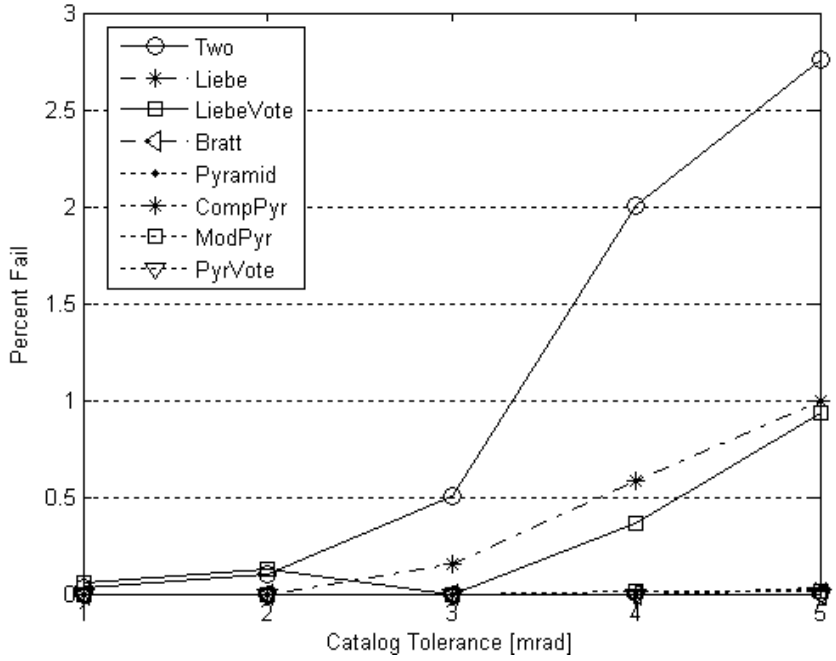

Figure 5.5 Failed match comparison of simulated data between algorithms at mag. 3.5 intensity

Though the number of failed solutions of the Liebe and Liebe with Voting algorithms appeared to slightly decrease earlier in Figure 5.3, in the figure above the amount of falsely identified spots in an image can be seen to increase with increasing search tolerance. This signifies that at lower tolerance values, the algorithms have greater restrictions imposed on the star identification and verification, thus many spots that are identified in the identification phase of the algorithm are rejected during the verification phase. As the tolerance values increase, the restrictions on identification are relaxed and the number of possible ID's for a spot increase, as well as the probability of error.

Furthermore, from Figure 5.5 it can be said that at a catalog search tolerance of 3 mrad, the Liebe with Voting method will give a false image solution, though not necessarily containing false matches, nearly 20% of the time. This is certainly not desirable and is regarded as a poor identification method given the boundary inputs.

Figure 5.6 shows the average number of empty solution sets returned by the algorithms which demonstrates the programs' ability to output a solution during conditions of false spots and centroiding errors. If an identification algorithm has low false solutions and low false matches, yet a high probability of empty solutions, then it is still an invalid method.

From Figure 5.6 it can be seen that all the Pyramid type algorithms return a high volume of empty solution sets; more especially at 1 mrad. The Two Star and Brätt algorithms return nearly no empty solutions for the range of catalog search tolerance, though the Two Star method begins to separate slightly at 5 mrad. This is valuable as it shows the Pyramid algorithms would need to continually re-take images and solve them in order to obtain a valid and usable solution. Also it can be shown that the Liebe with Voting improves over the Liebe method in its ability to obtain solutions as the catalog tolerance increases.

These figures have all shown the solution behavior of the algorithms based on the overall averaging of the false spot and centroiding parameters and how well the catalog search tolerance aids in overcoming these errors.

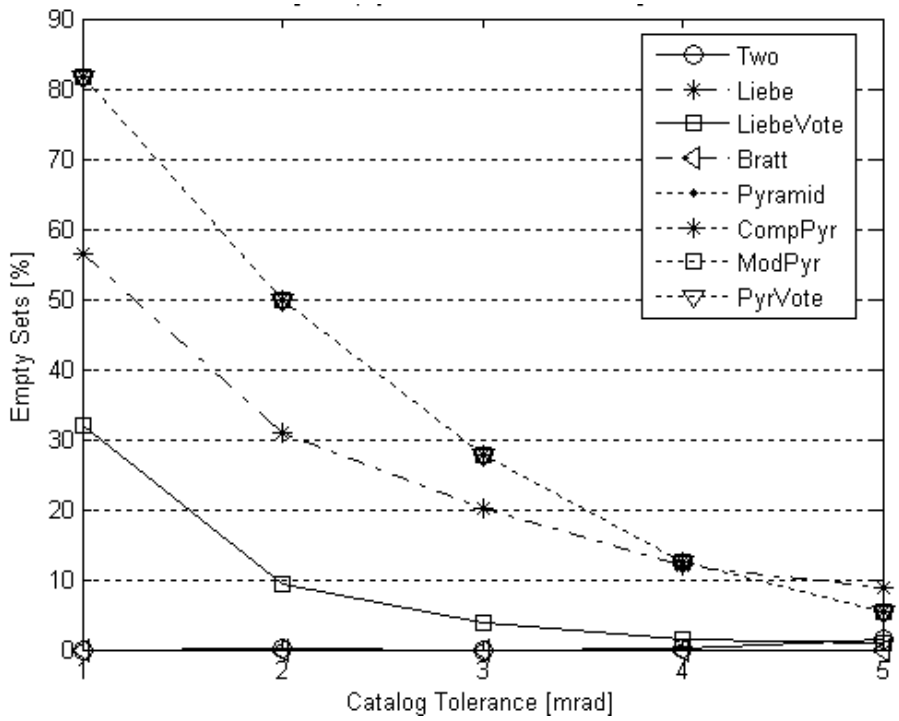

Figure 5.6 Average number of empty solution sets of simulation data for 3.5 magnitude intensity threshold

B. Simulated Pixel Distortion

Obviously, with high distortion in the image due to atmosphere, lens, and heating and cooling effects, the number of false matches of a solution will increase. More particularly will be discussed how the solutions are affected due to pixel distortion.

The centroiding errors were calculated based on the radial pixel value of the Aptina imager, ranging from 1 to 3 pixels of distortion. For the Aptina, .33 mrad is equivalent to 1 pixel of centroiding misalignment of a spot. The images have been averaged across all false spots and the catalog tolerance range, thus showing algorithm behavior strictly as a function of pixel distortion.

The following figures (Figures 5.7 and 5.8) demonstrate which algorithms are not satisfactory algorithms for the camera parameters. It was shown earlier that by raising the catalog search tolerance it is possible to overcome pixel distortion errors, though only marginally. At .66 mrad, and higher, a star shown as a spot in an image has become an entirely new star to the perspective of these methods.

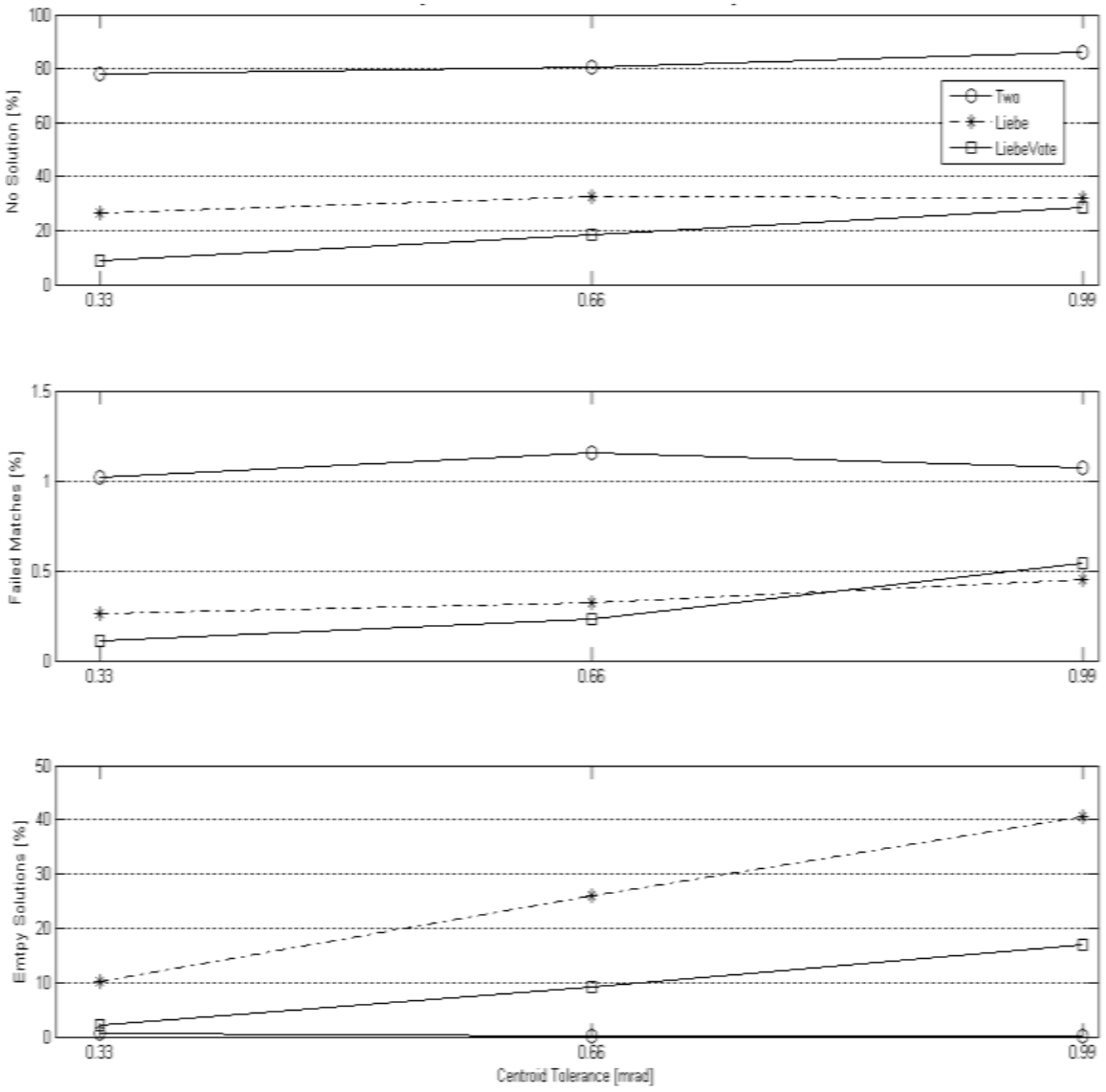

Figure 5.7 Average simulation solution failures of failed methods as a function of centroiding error for 3.5 magnitude threshold

Figure 5.7 demonstrates that again the Two Star, Liebe, and Liebe with Voting algorithms are not satisfactory algorithms for the camera parameters. It was shown earlier that by raising the catalog search tolerance it is possible to overcome pixel distortion errors, though only marginally for these algorithms. At .66 mrad, and higher, a star shown as a spot in an image has become either unrecognizable or an entirely new star to the perspective of these three methods.

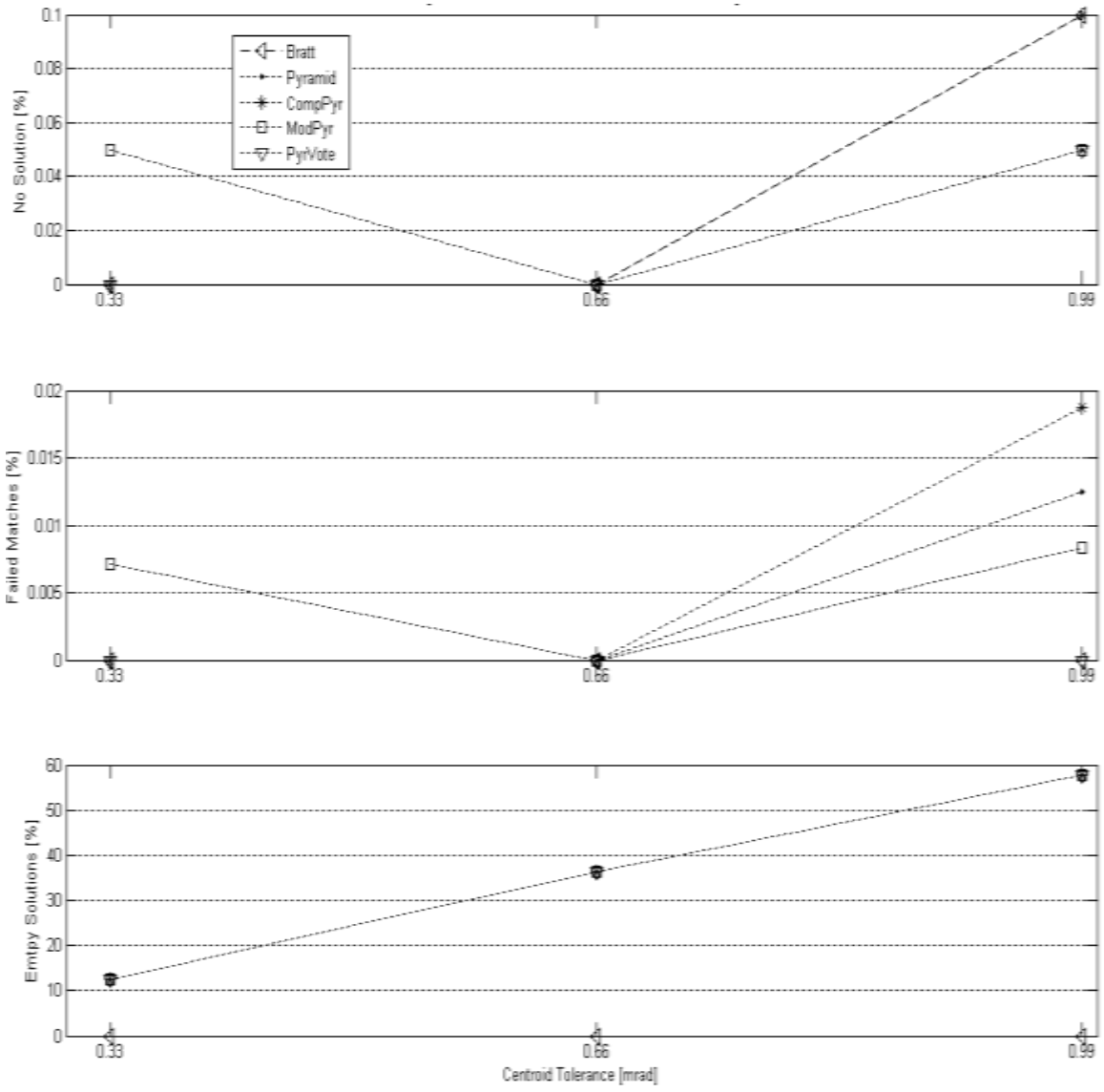

Figure 5.8 Average simulation failures of acceptable algorithms at 3.5 magnitude threshold

Refining the view to the last 5 algorithms (Figure 5.8) shows the Pyramid algorithms dealing relatively well with respect to pixel distortion, and the Brätt and Comprehensive Pyramid algorithms coincide, reaching a maximum of 0.1% image solution failure, once the distortion in the image plane reached 3 pixels in the Aptina imager. However, the Brätt and Pyramid with Voting algorithms show no false matches. This means improved internal verification than the other algorithms. This states that the Brätt and Pyramid with Voting begin to return fewer matches than the MRSS requirement. Furthermore, the Brätt algorithms returns no empty solutions.

Figure 5.9 expresses if the algorithms have the ability to return a solution. Illustrated are the Pyramid type algorithms which have a nearly 60% inability to resolve an image at large pixel distortions, whereas the Two Star is nearly zero, and the Brätt method is strictly zero.

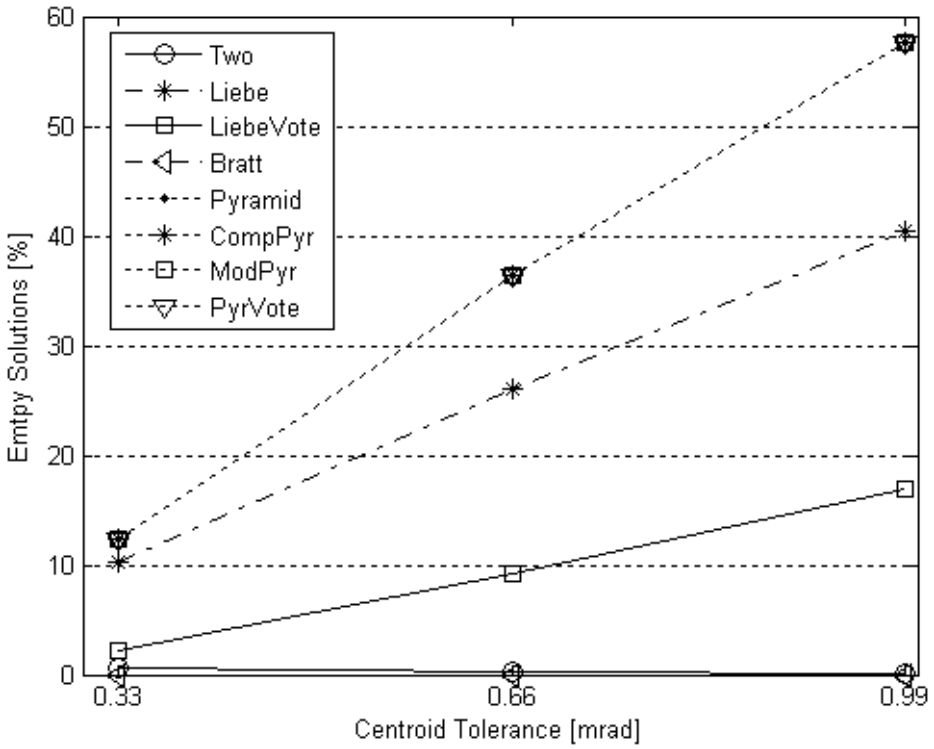

Figure 5.9 Average of empty solution sets for simulation due to pixel distortion, magnitude 3.5 threshold

From these results, a trend in algorithm performance begins to show. It is seen that the Two Star method, though able to deliver a solution nearly 100% of the time, would return incorrect attitude solutions. The same can be said of the Liebe and modified Liebe algorithms, though their failures are not as severe as the Two Star method.

With the Constrained Pyramid algorithm, though solutions are returned with rather high confidence of correct identifications, the number of empty solutions is high. Also, it can be shown from these figures that the Pyramid with Voting and Brätt algorithms that incorporated the Voting strategy with their identification maintained excellent results of low to no solution error and no false matching; though the Pyramid with Voting algorithm follows the same trend as the basic Pyramid algorithm with respect to empty solutions.

Thus, some of the algorithms fare well against high pixel distortion, and others fare well with low Catalog search tolerance. Figure 5.10 shows the overall performance of the algorithms in relation to each other and shows the Brätt algorithm as the overall best for solution acquisition and identification.

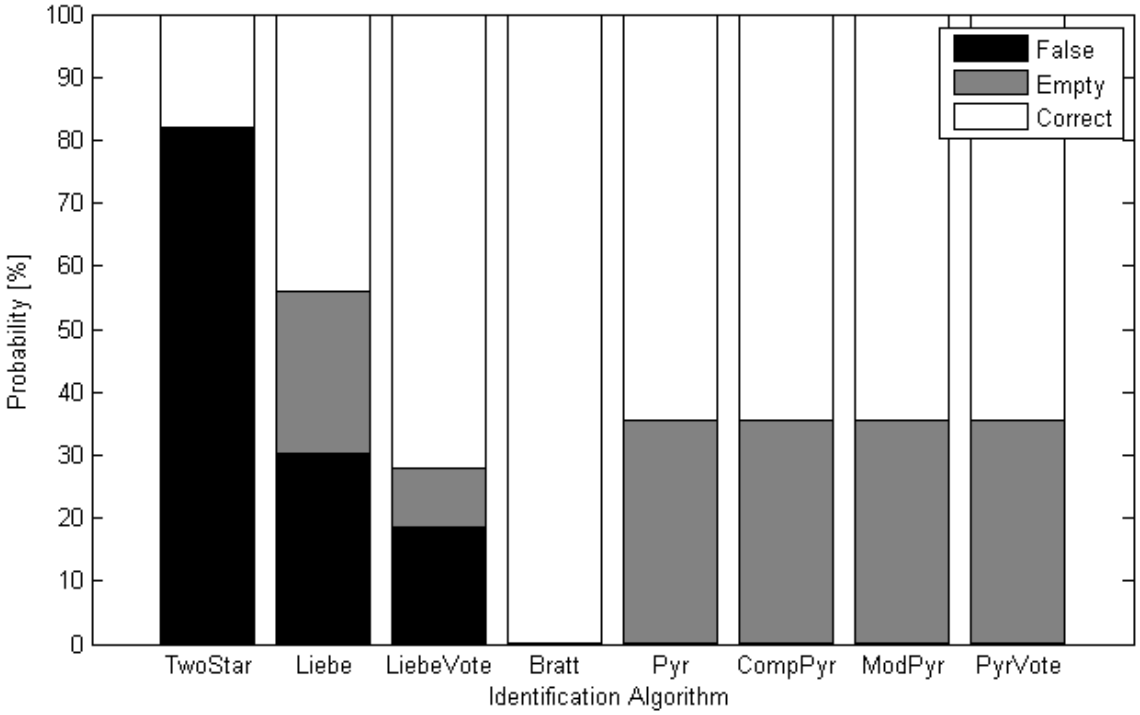

Figure 5.10 Overall probabilities of failure of simulated identification algorithms

II. Computational Impacts

The times shown in Table 5.1 are a result of processing simulated images in the form of a structured variable on a Pentium 4, Duel CPU 3.40 GHz processor, with 1.25 GB RAM using MS OS XP Service Pack 3.

It must be understood that these solutions were developed using MATLAB version 7.11.0 with *.M files that were not optimized for flight control, nor compiled; therefore, these speeds and processing times are not a valid estimate of the actual performance of the algorithms, but are useful as a means of comparison among each other. To obtain these times, the MATLAB Profiler tool was engaged, hence, overhead times are included in the results shown. Additionally, the average processing speed of the CPU

stayed at 1.6 GHz for simulation. It is anticipated that with optimized flight coding these times would be reduced by a factor of ten. Table 5.1 shows averaged times for solving simulated data of 6000 images with an average of 27 spots per image.

Table 5.1 Average time [sec] per image for solution of all simulation data at 27 average spots per image

	Magnitude Threshold	
Method	**3**	**3.5**
Two Star	0.0549	0.0622
Liebe	0.0141	0.019
Liebe Vote	0.7179	0.3295
Brätt	6.2604	3.6095
Pyramid	2.5017	3.3252
Comp. Pyramid	14.156	5.8783
Mod. Pyramid	2.7578	3.4005
Pyramid Vote	2.5535	3.3299

From profiling the algorithms, it was found that 95% of the time spent on any algorithm using the Voting technique was taken up during the indexing and searching of the patterns versus the feature list entries. Thus, search times could be significantly reduced using a k-vectoring technique or optimized searching method [48]. With all Pyramid type algorithms, the majority of the time was exhausted during the feature list searching as well; however, despite having low analysis time, the time would nearly have to be tripled in order to output a solution viable for navigation, as can be verified by the number of empty solutions seen in Figure 5.6.

Speed of identification can be increased by refinement of the imager to diminish the number of spots that will be shown on an image. By reducing the number of spots shown on an image this will not only help in improving the timing, but reduce the probability of error of the algorithms.

III. Memory Usage Results

As MAT or DAT files, the star identification outputs are just under 1 KB in size. This output is overwritten after the identification of the next image, thus being maintained in the RAM of the onboard computer. No other outputs are given. The most intensive use of the RAM was during index searching of

the feature list. The total amount of hard-drive space required is shown in Table 5.2 and is subject to the star magnitude strength of the imager used.

Table 5.2 Total permanent hard-drive space [MB] required

	Magnitude Threshold		
Method	**3**	**3.5**	**4**
Two Star	0.57	0.46	0.68
Liebe	0.09	0.15	0.26
Liebe Vote	39.04	16.05	18.08
Brätt	39.04	16.05	18.08
Pyramid	16.04	6.06	7.09
Comp. Pyramid	16.04	6.06	7.09
Mod. Pyramid	16.05	6.06	7.09
Pyramid Vote	16.04	6.06	7.09

Individually the magnitude reduced sub-catalog database is also subject to the imager used and allows the flexibility to select which database would be most appropriate. See Table 5.3.

Table 5.3 Sub-catalog database size [MB]

Magnitude	**3**	**3.5**	**4**
Size	0.027	0.042	0.074

The largest component of the memory system used is taken by the feature lists which vary based on magnitude, shown in Table 5.4. Truncation of the feature lists was discussed in Chapter 3I.B.2. The feature lists can also be viewed by how many patterns and features are in each based on magnitude threshold and algorithm type (Table 5.5). On observing these results, one would ask why patterns for Liebe with Voting and Brätt algorithms with only 3 features are significantly more numerous than patterns of 6 features. With the 3 feature patterns, the patterns require a central spot and form an interior angle which is unique between the set of stars in the sky and changes with which star is targeted first. Hence, if 3 stars are given, the interior angle of stars 1-2-3 is different than the interior angle of 2-1-3, which differs from stars 3-1-2. Therefore, the number of combinations is considerable compared to the Pyramid algorithms which do not prioritize on a central spot; all combinations 1-2-3, 2-1-3, and 3-1-2 are equal to the Pyramid algorithms.

Table 5.4 Feature list space usage [MB]

	Magnitude Threshold		
Method	**3**	**3.5**	**4**
Two Star	0.533	0.413	0.594
Liebe	0.062	0.101	0.182
Liebe Vote	39	16	18
Brätt	39	16	18
Pyramid	16	6	7
Comp. Pyramid	16	6	7
Mod. Pyramid	16	6	7
Pyramid Vote	16	6	7

Table 5.5 Number of patterns in feature list

Method	Magnitude Threshold			Features in Pattern
	3	**3.5**	**4**	
Two Star	3100	2405	3460	1
Liebe	177	288	518	3
Liebe Vote	115227	46628	55324	3
Brätt	115227	46628	55324	3
Pyramid	30793	11910	14067	6
Comp. Pyramid	30793	11910	14067	6
Mod. Pyramid	30793	11910	14067	6
Pyramid Vote	30793	11910	14067	6

CHAPTER 6

EXPERIMENTAL RESULTS

I. Experimental Testing

The simulation results have shown an estimate and basis for measuring the quality and functionality of the identification methods. Of greater importance is the actual proficiency of the algorithms with true data.

An extensive study was made into the retrieval of images from an Aptina [28] imager (confidential spec. sheet from Micron), the Droid X2 [60] cell phone camera, and DISC [61] imager from Space Dynamics Laboratory, with primary focus on cellular phone cameras, by Fowler [56]. His study provided a program that retrieves images from a star camera showing observable stars as highlighted pixels. These pixels are converted into 3-dimensional unit vector spots which he did by centroiding groups of immediately adjacent illuminated pixels and thresholding the pixel intensity based on the noise to signal ratio. Fowler's program was used in this study to provide the spot inputs for all the experimental data studied. Fowler's program permitted the use of Atmospheric and Lens distortion correction which were both deactivated to test the algorithms against corrupt data.

In total, 422 images were taken on two separate nights with this Aptina imager over the course of 4 hours per night. The first set of data collected 170 images and was taken October 5th, 2012, at an approximate elevation of 4,600 feet in Logan UT, USA; the second set collected 252 images on November 6th, 2012, at the same location.

Shown will be the results at a magnitude threshold of 3.5 as a means of comparison with the simulation data in Chapter 5, however, the testing included thresholding of 3 and 4 magnitude star fields as well; additional figures may be seen in the Appendix (see Appendix BII). The solutions to these images were taken at higher search tolerances (> 5 mrad) than the simulation data due to pollution, light cloud cover, the desire to test the limitations of the algorithms at increased tolerances, and the lack of information as to the amount of distortion that could be expected on the image plane from the Aptina camera.

A. Real Data vs. Catalog Tolerance

From the October data set it was discovered that the Two Star method performed better than anticipated with respect to the simulation, reaching approximately 45% solution error at the extreme end of

the search tolerance. All the other algorithms performed as expected within the 1 to 5 mrad range, yet show very distinct behaviors beyond the simulation's catalog search tolerance range. As can been seen in Figure 6.1, the upper limit of the Pyramid type algorithms exists near the 5 mrad tolerance. The Brätt algorithm however retrieved no erroneous solutions despite the increase in tolerance. The algorithms behaved as expected within the range of 1 to 5 mrads, yet all fail beyond this range. The Brätt algorithm is the only algorithm that maintains itself stable at a probable solution failure of 0%, as seen in Figure 6.1.

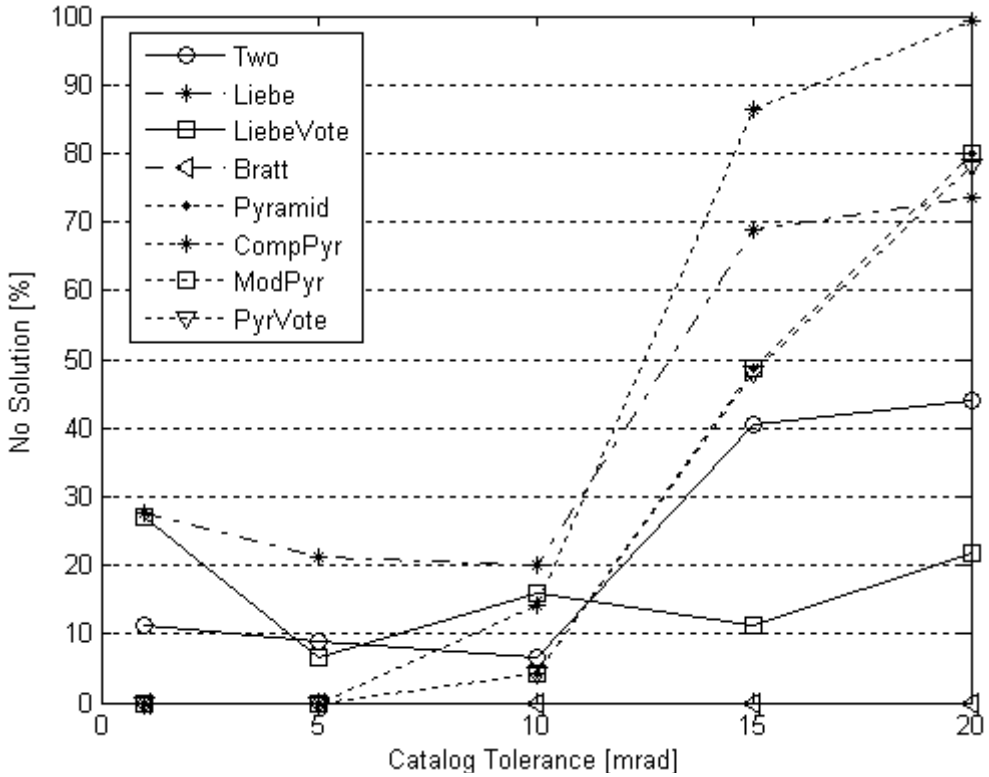

Figure 6.1 Average solution failure of experimental data sets for Oct data at 3.5 magnitude threshold

From the simulation trials, it was expected that the Pyramid algorithms would be more accurate than what is shown, yet it can be clearly seen at 10 mrad of catalog tolerance all the Pyramid algorithms are erroneous. It can be shown that the Pyramid with Voting follows very closely the behavior of the Constrained Pyramid algorithm, with only slightly improved results at the extreme limit of 20 mrad. This supports the assumption that the Pyramid with Voting would be an improvement over the basic structure of the Pyramid scheme, though with less improvement than expected.

Comparing these results to the November testing, approximately the same results can be seen in Figure 6.2, though it is clear that the data is more erroneous than the data from October. Most of the algorithms behave as expected, with exception that now the Brätt algorithm fails at 1 mrad, and peels upwards at 15 mrad, It remains under 8% during the entire range (Figure 6.2).

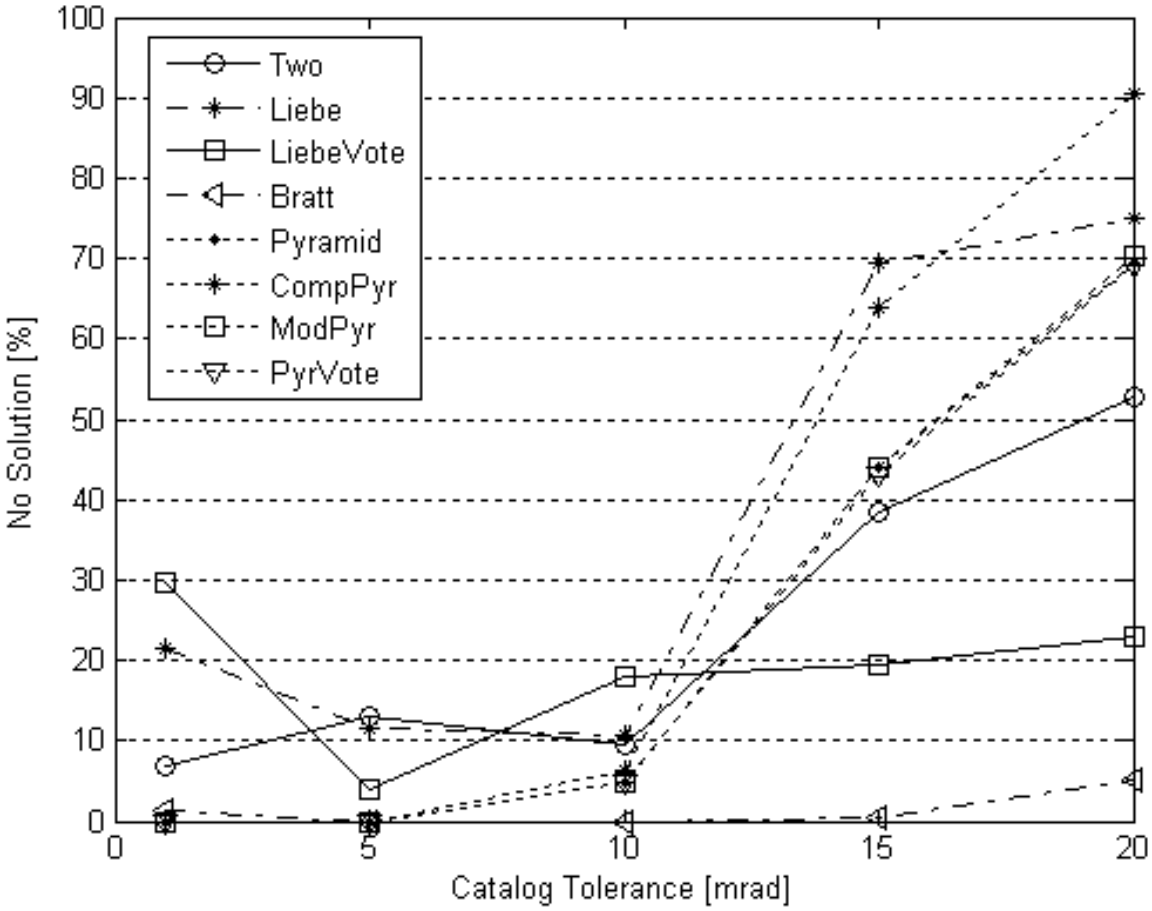

Figure 6.2 Average solution failure for experimental data sets for Nov data at 3.5 magnitude threshold

This proves that the Pyramid algorithms are satisfactory within the range of 1 to 5 mrad, and shows the Brätt method is valid between the range of 1 and 10 mrad; twice the range of the Pyramid processes. As well, it can be concluded that the Two Star, Liebe, and Liebe with Voting are entirely unacceptable methods for the Aptina imager, yet, may be suitable for imagers of higher quality where the margin of pixel distortion is far lower, and thus the search tolerance can be lowered. For additional comparison with the

simulation data, the images shown below (Figures 6.3 and 6.4) further confirm the results that were anticipated.

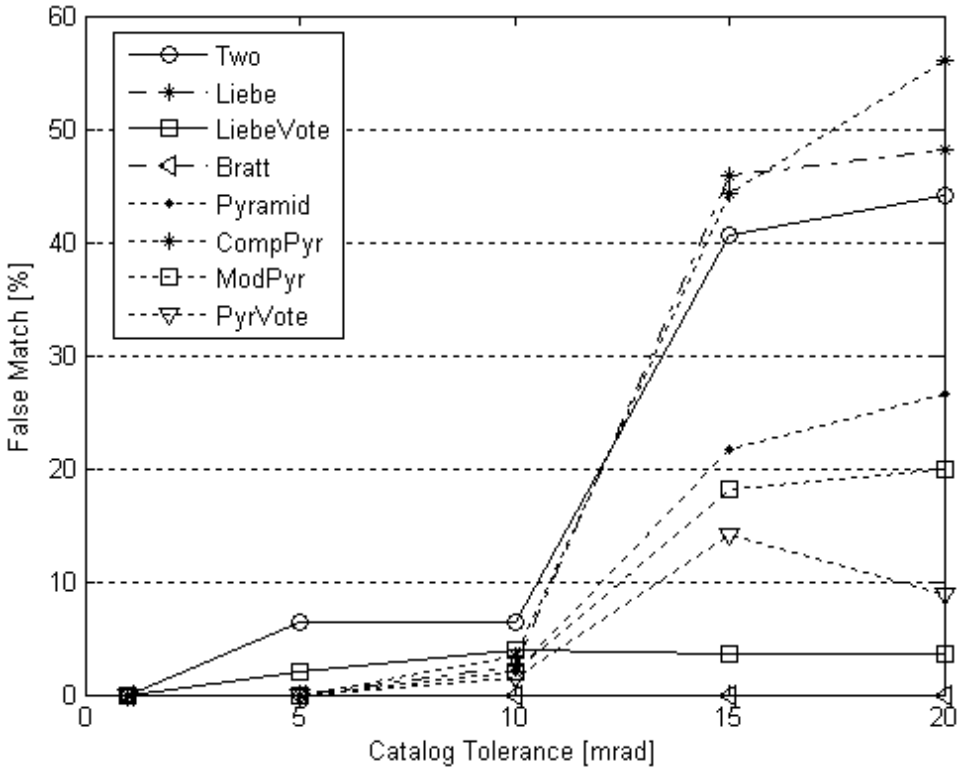

Figure 6.3 Average false matches of experimental data during Oct test, 3.5 theshold

Again, Figures 6.3 and 6.4 confirm the results derived from the simulation trials where between 1 and 5 mrad the worst performer was the Two Star algorithm, which reached under 3% false identification in the simulation Figure 5.3, and an average of 9% in the combined October and November tests. The tests performed in October and November signify that there existed a greater pixel distortion than anticipated within the range of 1 to 5 mrad, which can be attributed to the environmental conditions experienced during the night sky experiment.

Looking at the November data (Figure 6.4), again it is seen that all of the Pyramid type algorithms and Brätt method are accurate within the bounds of 1 and 5 mrad, and the Brätt algorithm remains under an error of 0.46% for a false match as far up as 20 mrad of catalog tolerance.

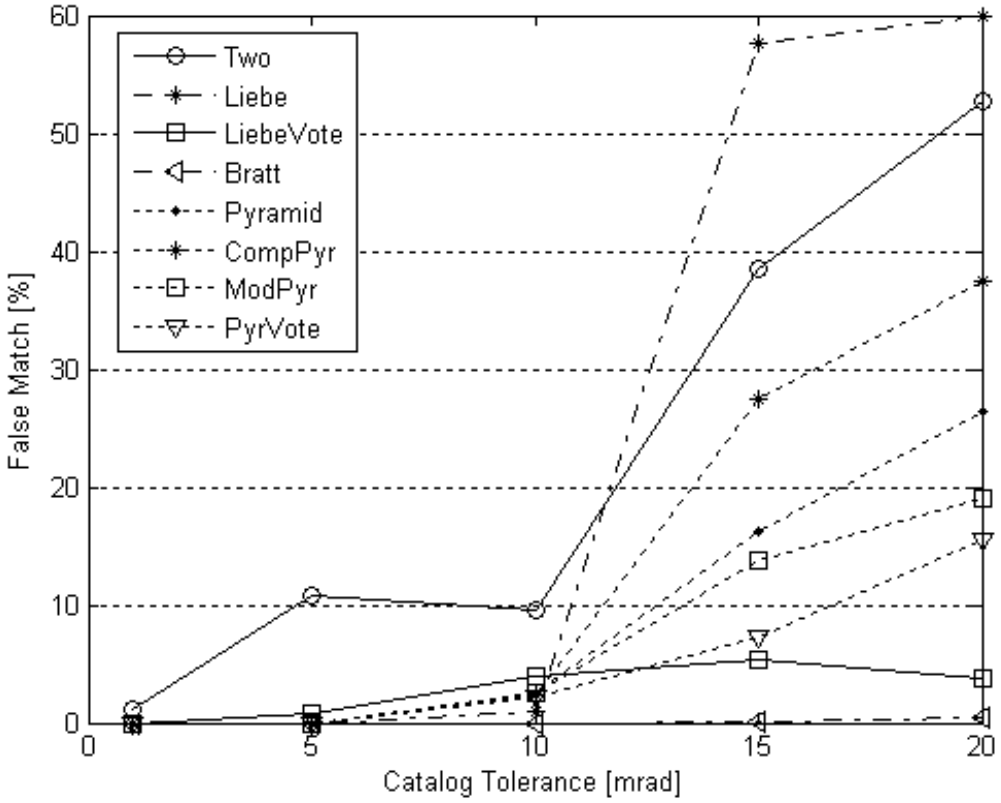

Figure 6.4 Average false matches of experimental data during November test, 3.5 threshold

Additionally, it must be clarified that the Brätt algorithm, which in Figure 6.2 appeared to fail within the 1 to 10 mrad range, does not produce any falsely identified stars, or matches, for that range of tolerance. This denotes that some of the solutions returned contained less than the MRSS of 4 matches in the star ID output.

The following two figures (Figures 6.5 and 6.6) are given to provide a means of comparing the likelihood of the algorithms' ability to obtain a solution during testing against the simulation estimate. They show nearly the same trend as the simulation though not as quadratic.

Strictly for the case of these two experiments, Figures 6.5 and 6.6 show the Brätt algorithm has a 100 % probability of returning a solution between the range of 1 to 15 mrad, and at 20 mrad all but the Two Star method have nearly a 97% chance of procuring a solution.

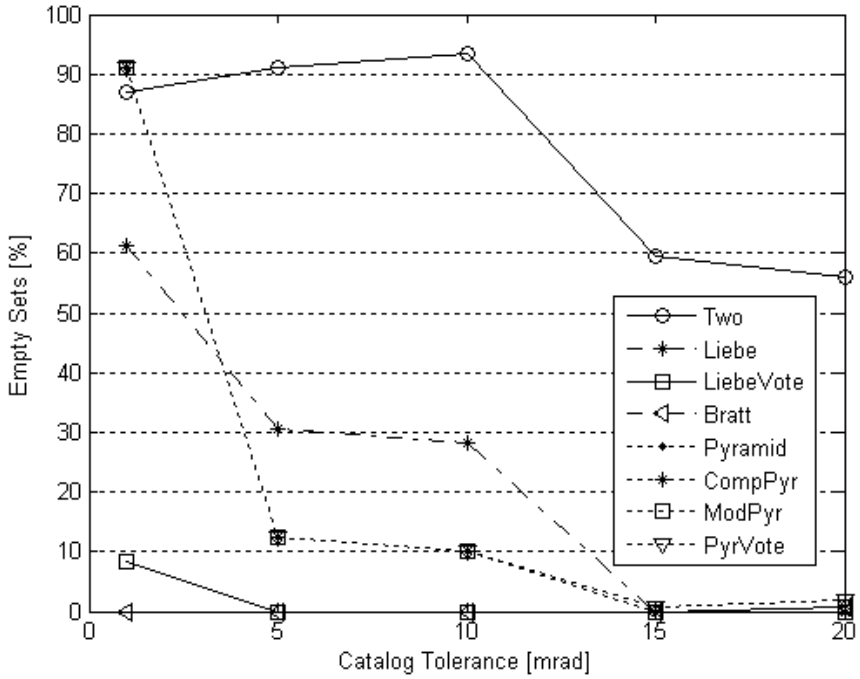

Figure 6.5 Average empty set for Oct data, 3.5 threshold

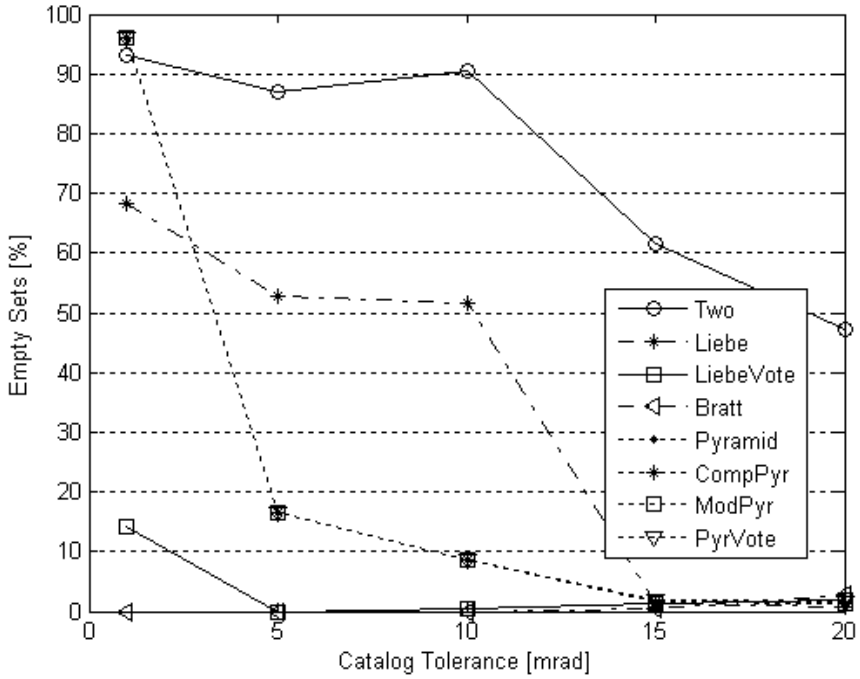

Figure 6.6 Average empty set for Nov data, 3.5 threshold

The Pyramid type algorithms behave as expected from the simulation results, showing an extreme level of empty solutions at 1 mrad. From Figures 6.3 and 6.4, beyond 10 mrad the Pyramid algorithms all begin failing. However, at 5 mrad there exists nearly 0% probability that the algorithms will fail, with a reduced amount of empty solutions. Thus, this proves that there exists a tolerance at which these algorithms will be acceptable in combination with the Aptina imager.

The next two images (Figures 6.7 and 6.8) show the overall combined performance of all the algorithms as a function of light intensity thresholding, averaging across the 1 to 20 mrad catalog range.

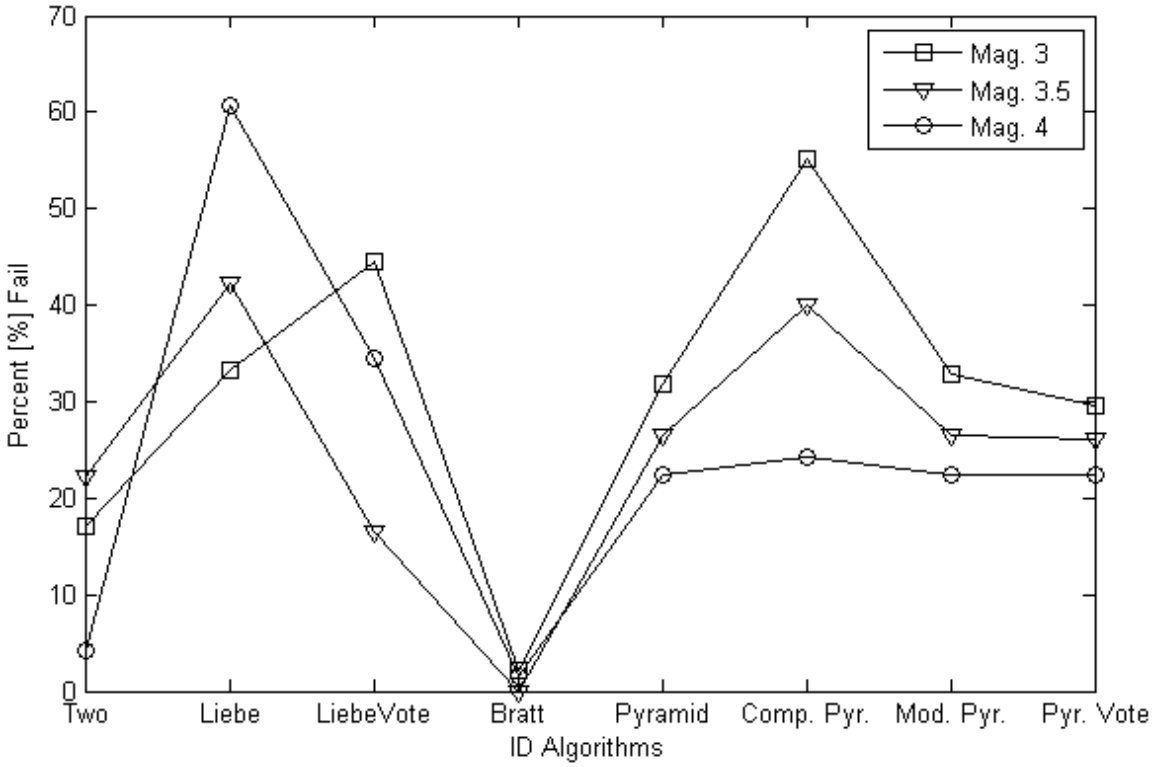

Figure 6.7 Probability of solution error as a function of magnitude threshold for all algorithms during Oct test

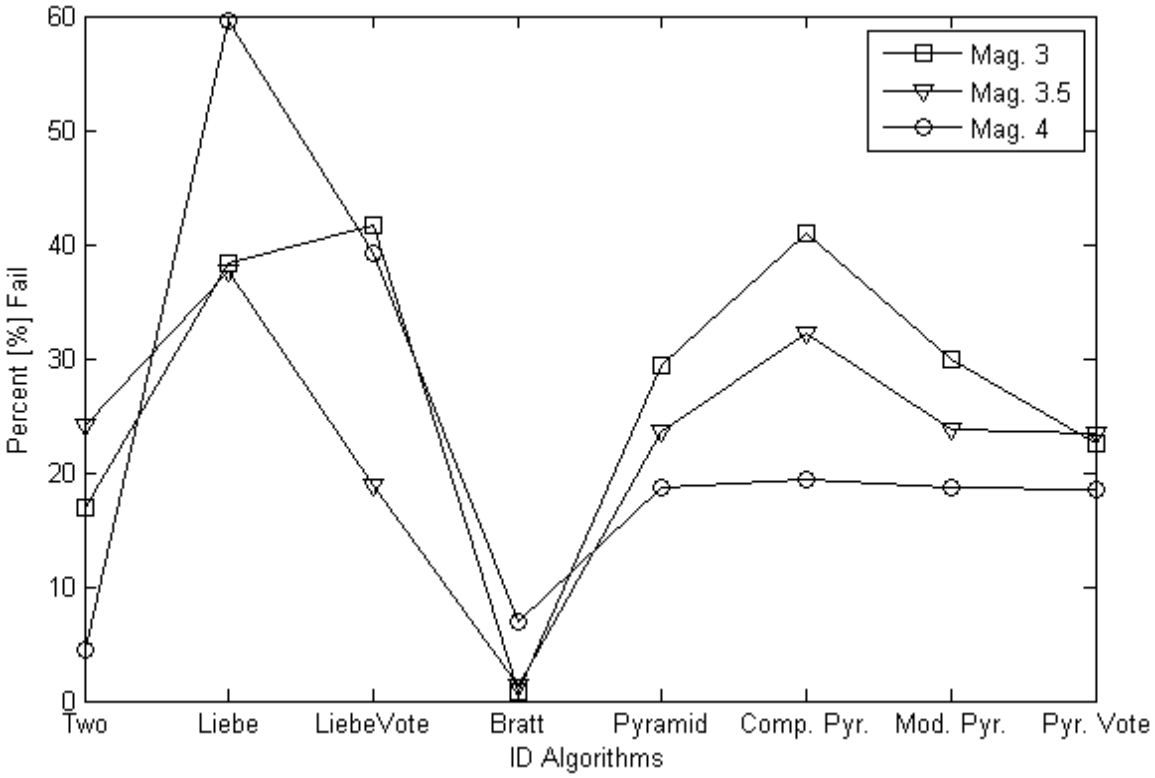

Figure 6.8 Probability of solution error as a function of magnitude threshold for all algorithms during Nov test

From Figures 6.5 and 6.6 it can be established that all but one of the algorithms have a high probability of obtaining a solution to the images given by an imager, and from the last two plots (Figures 6.7 and 6.8) it can be seen that the Pyramid algorithms manifest improved results at a magnitude threshold of 4, the Liebe methods do not follow an exact order or preference, the Two Star method is far more stable at a magnitude threshold of 4, and the Brätt algorithm yields the lowest overall error for all magnitude types; behaving better during the October testing than the November. This difference is attributed to an increase of light pollution during the November test.

Figures 6.9 and 6.10 show the overall solution behavior of the algorithms for the 1 to 5 mrad tolerance range, which was used during simulation, with the dependence on magnitude intensity removed.

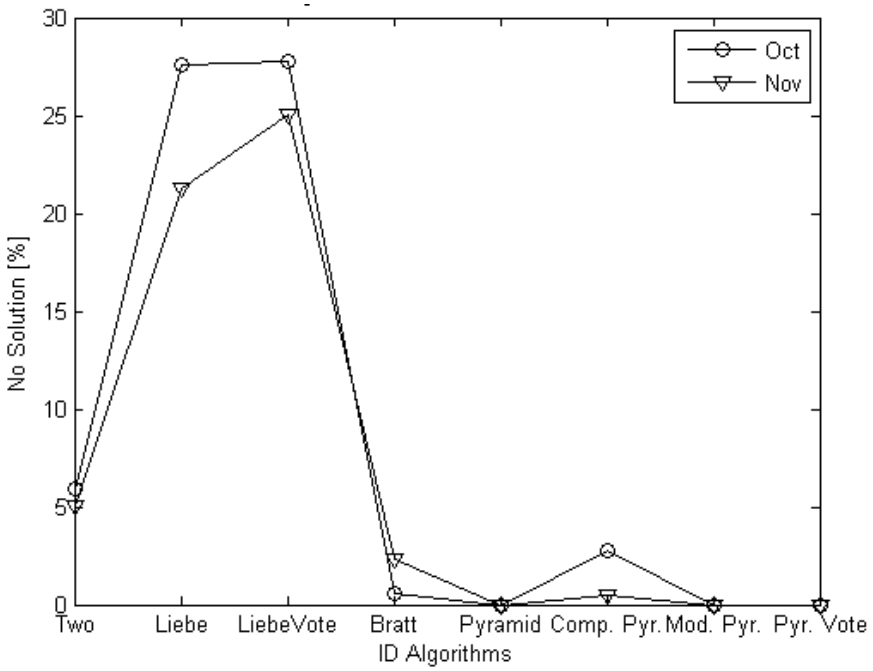

Figure 6.9 Overall solution failure for experimental data for tolerances 1 to 5 mrad

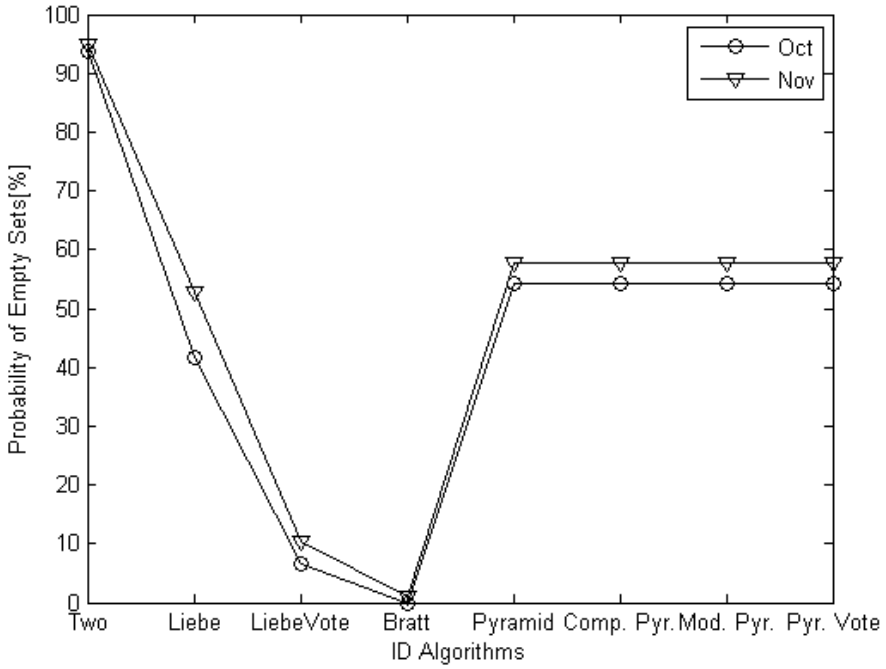

Figure 6.10 Overall empty solution sets for experimental data for tolerances 1 to 5 mrad

From Figure 6.9 it might be concluded that the, Constrained Pyramid, Modified Pyramid, and Pyramid with Voting are the 3 algorithms that attained the best solution success. However, it has been shown that the Brätt algorithm did not obtain any false matches for the 1 to 5 mrad tolerance range, thus the lack of performance is due to the increased restriction of an MRSS of 4. As it was mentioned in Chapter 5, the behavior of this algorithm would be greatly improved if this restriction (which was imposed because of the nature of the Pyramid algorithms) was set to 3 matches. Figure 6.11 shows the overall combined results of the algorithms.

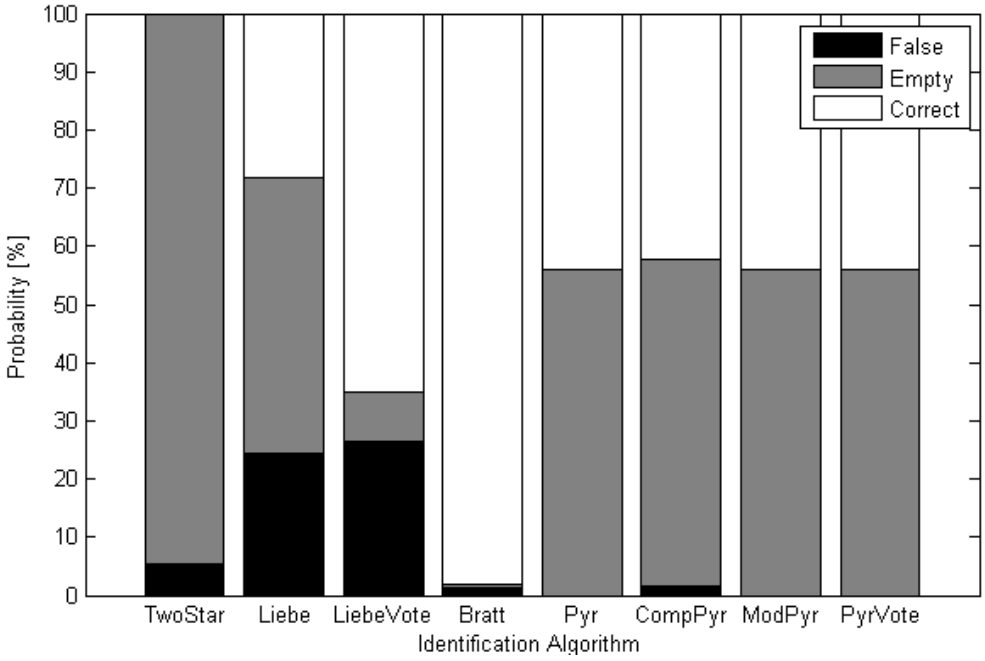

Figure 6.11 Solution comparison of experimental data for tolerances 1 to 5 mrad

Figure 6.11 demonstrates a perspective of the behavior of the identification algorithms as they relate to each other for the tolerance range of 1 to 5 mrad. The data from the October and November tests are combined and averaged across magnitude and tolerance. These results follow very closely the results from simulation, with exception of the Two Star algorithm which has a far greater number of empty solutions in the experimental. This states that the solutions returned by the Two Star method were rejected during verification. Obviously, the Two Star method has far fewer restrictions imposed on identification, and thus has a greater tendency to return false or empty solutions as compared to the other algorithms.

B. Experimental Data Computation

Comparing the timing values of Chapter 5 with the overall times of the experimental data tests prove them to be nearly equivalent. Here the times of the actual Aptina test images are split to show average times during each testing night. Table 6.1 contains images averaging at 13 spots per image, where Table 6.2 comprises times with an average of 11 spots per image.

Table 6.1 Average time [sec] per solution method to solve Oct data.

	Magnitude Threshold		
Method	**3**	**3.5**	**4**
Two Star	0.1035	0.138	0.2028
Liebe	0.0204	0.0211	0.0841
Liebe Vote	0.7239	0.3479	1.2149
Brätt	8.4274	3.8601	4.3422
Pyramid	4.3698	2.654	5.0846
Comp. Pyramid	15.8612	6.2578	7.0792
Mod. Pyramid	4.6371	2.7407	5.1597
Pyramid Vote	4.383	2.6496	5.0845

Table 6.2 Average time [sec] per solution method to solve Nov data.

	Magnitude Threshold		
Method	**3**	**3.5**	**4**
Two Star	0.0709	0.0755	0.1058
Liebe	0.012	0.0103	0.065
Liebe Vote	0.6248	0.2854	0.7757
Brätt	4.7347	2.0948	2.3664
Pyramid	3.3348	1.4014	2.5285
Comp. Pyramid	8.3781	3.0278	3.4138
Mod. Pyramid	3.5408	1.4735	2.5906
Pyramid Vote	3.3805	1.4182	2.5486

Comparing these times to the number of empty solutions, it can be assumed that the time for deriving a solution (not necessarily valid) for the Pyramid type methods will need to be increased when using a search tolerance range of 1 to 5 mrad, as the output would be empty. The imager would then need to re-take an image for processing.

Furthermore, it must be noted that the times between tests are significantly different. Firstly, the times for the Pyramid type algorithms do decrease during November due to the reduced number of spots per image, however, not substantially. The algorithms which are comprehensive (Brätt and Comp. Pyramid) nearly double in time for magnitude 3 star fields. This shows that large numbers of spots have a serious impact on the solution speed of these algorithms. Additionally, much can be said concerning the time difference between magnitudes. 3.5 and 4 magnitude fields are lower in time than magnitude 3 fields due to the truncation of the feature lists. Thus, it can be said that by truncating the feature lists, solution speed can be increased for algorithms which use a comprehensive star processing. This is further confirmed with comparison to simulation results.

CHAPTER 7

SUMMARY

The Hipparcos catalog was used as the main reference for stars. This database proved to be what was needed to satisfy the requirements towards star identification. The use of Liebe's and Mortari's algorithms, along with Tichy's two star method using voting, were fundamental in the development of the star identification algorithms constructed and tested in this analysis.

The identification algorithms were modeled and analyzed against two main sources of error: Centroiding distortion, and False spots. Through simulation, the algorithms were tested using Monte Carlo randomization of the placement of these centroiding errors and locations of false spots. Using a catalog tolerance range of 1 to 5 mrad, these algorithms solved 12,000 images which were used to model expected experimental values.

Test results from the simulation and the October and November tests showed 4 main algorithms that met the demands of solution acceptance, computational performance, and robustness. These algorithms were the Constrained Pyramid, Modified Pyramid, Pyramid with Voting, and Brätt algorithms. The Aptina [28] MT9P031 camera from Micron was used as the basis of experimental study, using it's parameters as the inputs to the algorithms and simulation model.

The results from the algorithms show that it is possible to use lower quality imaging devices in star navigation. Table 7.1 shows a performance analysis of the algorithms and lays claim to the Brätt Three Star with Voting and Constrained Pyramid with Voting methods as the preferred Lost in Space Algorithms.

Table 7.1 Performance analysis of star identification algorithms

Risk Situations	Algorithms and relative rating (1 good, 9 poor)							
	Two Star	*Liebe*	*Liebe Vote*	*Brätt*	*Pyramid*	*Comp. Pyr.*	*Mod. Pyr.*	*Pyr. Vote*
Slow Processing	1	1	3	7	5	9	5	5
False Matching in Simulation	9	7	7	1	5	5	3	1
False Matching in Experimental	9	5	7	1	1	3	1	1
False Solutions in Simulation	9	7	5	1	1	1	3	1
False Solutions in Experimental	5	7	9	3	1	3	1	1
Empty Solutions in Simulation	1	7	5	1	9	9	9	9
Empty Solutions in Experimental	5	3	1	1	5	5	5	5
Memory Storage Overcapacity	3	1	9	9	7	7	7	7
High RAM Usage	3	1	7	9	5	9	7	7
Poor Verification	7	7	5	3	5	7	3	1
Total	*52*	*46*	*58*	***36***	*44*	*58*	*44*	***38***

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

It was found that indeed star identification algorithms can be used in conjunction with low quality imagers, such as the Aptina [28] MT9P031 from Micron. It was also discovered that the algorithms are very sensitive to the magnitude reduction used to propagate images. Not all the algorithms tested passed the criteria for acceptance. It was found that use of a catalog search tolerance was appropriate, but only within certain boundaries, which are dependent on the type of camera used. The tolerance boundary was found to be most beneficial at 1 to 5 mrads for most of the algorithms when dealing with the Aptina imager from Micron. The algorithms that performed the best were: Brätt Three Star with Voting and Constrained Pyramid with Voting. These two proved to be the best at star identification and meeting the requirements of speed, memory usage, solution accuracy, and verification.

In future work it is suggested to incorporate k-vectoring with the identification algorithms to improve identification speed, as well the use of SQLite database for storage of the feature lists and star catalogs. The Brätt algorithm would be improved if the minimum required spots for a solution were reduced from 4 to 3; as well, if the number of features were increased to 6 the feature list size would be reduced by nearly 50%. For the Pyramid type algorithms, it is recommended that they be constrained to only 9 features rather than the 24 used. Additionally, there are several permutations of star processing and verification combinations that could be performed and tested. Future work can also include the use of magnitude intensity thresholding during identification to use only the brightest n desired spots, rather than thresholding the feature lists and star catalogs alone, this would speed processing time and improve identification. As well if an initial attitude estimate were given, these algorithms would obtain identifications faster and more accurately. Thus these algorithms could be used as tracking algorithms, and in the case where the spacecraft becomes disoriented, the algorithms can revert to their lost in space programming. Lastly, it is recommended to test these algorithms with other cellular phone type cameras to further validate the claim that these algorithms are acceptable for use with low quality, but high resolution, imagers.

REFERENCES

[1] Wikipedia, “CubeSat.” Available: http://en.wikipedia.org/wiki/CubeSat [retrieved 31 Jan 2013].

[2] Greenland, S., and Clark, C., *CubeSat Platforms as an On-Orbit Technology Validation and Verification Vehicle*, Madeira, Portugal: 2010.

[3] Long, M., and Twiggs, R., “A CubeSat Derived Design for a Unique Academic Research Mission in Earthquake Signature Detection,” *Small Satellite Conference*, vol. 9, 2002, pp. 1–17.

[4] Selva, D., and Krejci, D., “A Survey and Assessment of the Capabilities of Cubesats for Earth Observation,” *Acta Astronautica*, vol. 74, May, 2012, pp. 50–68.

[5] Wertz, J. R., *Spacecraft Attitude Determination and Control*, D. Reidel Publishing Company, Dordrecht, Netherlands, 2002.

[6] Wikipedia, “Attitude Control.” Available: http://en.wikipedia.org/wiki/Attitude_control [retrieved 31 Jan 2013].

[7] Abate, J. E., “Tracking and Scanning,” *IEEE Transactions on Aerospace and Navigational Electronics*, 1963, pp. 171–181.

[8] Ryan, K., Fullmer, R., and Wassom, S., “Experimental Testing of the Accuracy of Attitude Determination Solutions for a Spin-Stabilized Spacecraft,” *AAS*, 2011, pp. 1–11.

[9] Babcock, E. P., “CubeSat Attitude Determination via Kalman Filtering of Magnetometer and Solar Cell Data.”

[10] Eisenman, A. R., and Liebe, C. C., “The New Generation of Autonomous Star Trackers,” 1997, pp. 1–12.

[11] NASA, *Spacecraft Star Trackers*, Houston, 1970.

[12] Ball Aerospace, *CT-602 Star Tracker*, 2013.

[13] Ball Aerospace, *CT-601 High Accuracy Star Tracker*, 1995.

[14] Lee, S., Ortiz, G. G., and Alexander, J. W., *Star Tracker-Based Acquisition, Tracking, and Pointing Technology for Deep-Space Optical Communications*, 2005.

[15] TERMA, “Terma HE-5AS Star Tracker,” *Terma Space*. Available: http://www.terma.com/media/101677/star_tracker_he-5as.pdf [retrieved 31 Jan 2013].

[16] Percival, J. W., Babler, B., and Bonomo, R., “The ST5000 Ultra-Low-Cost Star Tracker and Low-Bandwidth Digital Imager,” 2000, p. 1.

[17] Ball Aerospace, *High Accuracy Star Tracker (HAST)*, 2013.

[18] Ball Aerospace, *CT-633 Stellar Attitude Sensor*, 2013.

[19] French, J., and Sternberger, K., “Recalibrating the Star Sensor : From the IBEX Satellite to the RENU Rocket,” 2011, pp. 11–15.

[20] Surrey, “Altair HB + Star Tracker (2-Unit Package),” *Satellite Technology US LLC.* Available: http://www.sst-us.com/shop/satellite-subsystems/aocs/altair-hb--star-tracker--2-unit-package- [retreieved 31 Jan 2013].

[21] Blue Canyon, *Blue Canyon Technologies XACT datasheet*, 2012.

[22] Berlin Space Technologies, *Star Tracker ST-200*, 2012.

[23] Birnbaum, M. M., “Spacecraft Attitude Control Using Star Field Trackers,” 1997.

[24] “2013 Best Smartphones Review and Comparisons,” *Tech Media Network.* Available: http://cell-phones.toptenreviews.com/smartphones [retrieved 31 Jan 2013].

[25] Scott, P., “Beyond Megapixels: The Quest for A Better Cellphone Camera Comparison Standard,” *CameraTechnica2.* Available: www.cameratechnica.com/2012/05/11/beyond-megapixels-the-quest-for-a-better-cellphone-camera-comparison-standard [retrieved 31 Jan 2013].

[26] Stoker, G., “Best camera phone: 6 Handsets Tested,” *Techradar.* Available: http://www.techradar.com/us/news/phone-and/communications/mobile-phones/best-camera-phone-6-handsets-tested-904250#articleContent [retrieved 31 Jan 2013].

[27] Dolcourt, J., “Best Camera Phones,” *CNET.* Available: http://reviews.cnet.com/best-camera-phones/. [retrieved 31 Jan 2013].

[28] Micron Technologies, *5Mp 1/2.5-inch CMOS Digital Image Sensor Data Sheet*, 2006.

[29] Thurmond, R., “A History of Star Catalogues,” 2003, pp. 1–55.

[30] CDS, “Henry Draper Catalogue and Extension 1.” Available: http://cdsarc.u-strasbg.fr/viz-bin/Cat?III/135A [retrieved 31 Jan 2013].

[31] NASA, “Positions and Proper Motions Catalog.” Available: http://heasarc.gsfc.nasa.gov/W3Browse/star-catalog/ppm.html [retrieved 31 Jan 2013].

[32] ESA, “Hipparcos Main Catalog.” Available: http://heasarc.gsfc.nasa.gov/W3Browse/all/hipparcos.html [retrieved 31 Jan 2013].

[33] Wikipedia, “Star catalogue.” Available: http://en.wikipedia.org/wiki/Star_catalogue#HIP [retrieved 31 Jan 2013].

[34] Roser, S., and Bastian, U., “PPM Star Catalogue,” 1991.

[35] ESA, “Hipparcos Main Catalog,” *HEASARC Archive.* Available: http://heasarc.gsfc.nasa.gov/W3Browse/all/hipparcos.html [retrieved 31 Jan 2013].

[36] Na, M., and Jia, P., “A Survey of All-sky Autonomous Star Identification Algorithms,” *First International Symposium on Systems and Control in Aerospace and Astronautics*, Dept. of Computers, Science, and Technology, Beijing, 2006, pp. 896–901.

[37] Kosik, J. C., *"Star Pattern Identification: Application to the Precise Attitude Determination of the Auroral Spacecraft,"* Journal of Guidance, Control, and Dynamics, Toulouse, Vol. 14, No. 2, 1991, pp. 230-235.

[38] Groth, E. J., “A Pattern-Matching Algorithm for Two-Dimensional Coordinate Lists,” *The Astronomical Journal*, vol. 91, 1986, pp. 1244–1248.

[39] Spratling, B. B., and Mortari, D., "A Survey on Star Identification Algorithms," *Algorithms*, vol. 2, Jan. 2009, pp. 93–107.

[40] Anderson, D. S., "Autonomous Star Sensing and Pattern Recognition for Spacecraft Attitude Determination," M.S. Thesis, Dept. of Engineering, University of Texas A&M, Texes, 1991.

[41] Renken, H., and Rath, H. J., *" Three-Axis Attitude Determination by Image-Processed Star Constellation Matching."* Available: http://renken.de/dglr_1997_matching.pdf [retrieved 31 Jan 2013].

[42] Liebe, C. C., *Star Trackers for attitude Determination*, Denmark: 1995.

[43] Baldini, D., Barni, M., Foggi, A., Bernelli, G., and Mecocci, A., "A New Star Constellation Matching Algorithm for Satellite Attitude Determination," *ESA*, vol. 17, 1993, pp. 185–198.

[44] Scholl, M. S., "Six-feature star-pattern identification algorithm," *Applied Optics*, vol. 33, 1994, pp. 4459–4464.

[45] Ketchum, E. A., and Tolsen, R. H., "Onboard Star Identification without A Priori Attitude Information," *Journal of Guidance, Control, and Dynamics*, vol. 18, 1995, pp. 242–246.

[46] Van Bezooijen, R. W. H., "Autonomous Star Referenced Attitude Determination," *Proceedings of the Annual Rocky Mountain Guidance and Control Conference*, Keystone, CO: 1989.

[47] Mortari, D., Samaan, M. A., Bruccoleri, C., and Junkins, J. L., "The Pyramid Star Identification Technique," 2004, pp. 1–39.

[48] Mortari, D., and Neta, B., "k-Vector Range Searching Techniques," *AAS 00-128*.

[49] 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*, USU: 2002.

[50] Samaan, M. A., Mortari, D., and Junkins, J. L., "Recursive Mode Star Identification Algorithms," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 41, 2005, pp. 1246–1254.

[51] au Rousseau, G. L., Bostel, J., and Mazari, B., "Star Recognition Algorithm for APS Star Tracker," *IEEE A&E Systems Magazine*, 2005, pp. 27–31.

[52] Zhang, G., 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, Jul. 2008, pp. 891–897.

[53] Kolomenkin, M., Pollak, S., Shimshoni, I., and Lindenbaum, M., "Geometric Voting Algorithm for Star Trackers," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 44, 2008, pp. 441–456.

[54] Tichy, V., Fullmer, R., and Fowler, D., "Preliminary tests of commercial imagers for nano-satellite attitude determination," 2011, pp. 1–9.

[55] European Space Agency "Access the Catalogue Data Tools for interrogating the Catalogues Alternative access to the Hipparcos and Tycho Catalogues." Available: http://www.rssd.esa.int/index.php?project=HIPPARCOS&page=Research_tools [retrieved 31 Jan 2013].

[56] Fowler, D. M., "Error Modeling and Analysis of Cellular Phone Imagers used as Star Cameras," Utah State University, 2013.

[57] Liebe, C. C., and Joergensen, J. L., "Algorithms Onboard the Oersted Microsatellite Stellar Compass," *Proc. SPIE*, vol. 2810, 1996, pp. 239–251.

[58] Liebe, C. C., "Algorithm for Rapid Searching Among Star-Catalog Entries," *NASA's Jet Propulsion Laboratory - Tech Briefs.* Available: http://www.techbriefs.com/component/content/article/2913 [retrieved 31 Jan 2013].

[59] Samaan, M. A., Mortari, D., and Junkins, J. L., "Nondimensional Star Identification for Uncalibrated Star Cameras," *The Journal of the Astronautical Sciences*, vol. 54, 2006, pp. 95–111.

[60] "Droid X2 by Motorola," *Motorola.* Available: http://www.motorola.com [retrieved 31 Jan 2013].

[61] Space Dynamics Laboratory, *Compact ,Light-Weight, & Low Power Star Imaging for Nano- & Pico-Satellites,* 2010.

APPENDICES

APPENDIX A

CODING

This Appendix details the programs used for analysis during simulation and the star identification algorithm codes themselves. First, it is important to mention that the Hipparcos star catalog used in the analysis was downloaded from ESA [55] in its original format as a text document. This text document was converted to a MATLAB structured variable for ease of use in programming and extraction of information. The feature lists were constructed into *.MAT files as structured arrays that contain information of the stars used in each pattern and the features between them.

I. Simulation Codes

This section contains the codes for the main simulation run program. It outlines the various sub-functions used to generate star fields and spot list inputs to the identification algorithm programs. These programs were all developed in MATLAB.

A. Simulation Main

Star Identification Simulation Module

```
%Simulation program to test computation speed and accuracy of star
%identification algorithms. This program creates a simulated star image
%from ECI coordinates given by the 1991 Hipparcos Catalog. The positions of
%these stars have been updated to correspond to current star positions as
%of 2012. Based on a field of view (FOV) and a visual magnitude threshold
%(Mag_Cut) given by the USER, the Catalog is broken into Featurelists, and
%based on a given initial attitude vector (ViewVec) provided by the USER,
%the simulation gives a sampled sky image for processing.
%
%After an image has been created, 6 identification methods process the
%image and attempt to identify all spots found to what their corresponding
%Hipparcos number ought to be.
%
%The simulation program will output a SimAnsM#.mat file into the Current
%Folder of MatLab which contains 6 structured arrays holding all statistics
%of each algorithm. The statistics are ordered in the following:
%
% Catalog Estimate - Centroid Error - # fake spots - Ave. RCValue - Ave.
% Quality - Ave. % of failed solutions - Ave. % false identifications -
% Profiler Statistics
%
%This Simulation Module requires the following *.m files and *.mat files to
%exist in the Current Folder:
%
```

```
% HIP_ALL.mat, CompDual_Feature_Extract.m, Triad_Feature_Extrac.m,
% CompTriad_Feature_Extract.m, Pyramid_Feature_Extract.m, View_to_Quat.m,
% FOV_star_generator.m, Camera_Ref_Frame.m, getTwoStar_ID.m,
% getThreeStar_ID.m, getThreeStarVote_ID.m, getCompThreeStar_ID.m,
% getPyramid_ID.m, getCompPyramid_ID.m, Voting_Algorithm.m, IDAccuracy.m,
% rotateVector.m
%
%Created by: Steven Bratt
%
%User Inputs:
%
% Mag_Cut: The visual magnitude threshold to truncate the Hipparcos Catalog
% FOV:     The full field of view of the desired simulated camera image
%          (circular camera view)
% ViewVec: Initial attitude (given in Deg) based on right ascension and
%          declination
% rot:     Number of degrees to rotate image
% IA:      (Optional) Will truncate the Featurelists to 1 FOV of an
%          initial attitude
% ecen:    Centroiding error constraint (rad). Determines amount of
%          physical error to input in creating the image. For an Aptina
%          camera, 1 mrad = 3.438 arcmins, or 3.3E-4 rad per pixel
% ecat:    Catalog estimate (rad). Determines tolerance on searching the
%          Catalog and Featurelists for identification.
% Nfake:   Number of false spots, or false stars, to include in the image.
%          Allocation of these spots will be randomized.
% MRSS:    The minimum required number of stars for a desired solution.
% n_iter:  The number of iterations to make per run. A run will be 1 ecen,
%          1 ecat, and 1 Nfake. (i.e. If ecen = [1:2]*(3.3*10.^[-4,-3],
%          ecat = 15*10^-3, Nfake = 0, and n_iter = 2, the simulation will
%          run 2 times per ecen, and a total of 6 runs.
%
%Input Example:
%
% Mag_Cut = 4;        %Star brightness value cutoff
% FOV     = 50;       %Radius of FOV [DEG]
% ViewVec = [187 54]; %Initial Attitude Vector [Deg] (Big Dipper)
% rot     = 90;       %Rotational image angle [Deg]
% IA      = [180 57]; %Estimated Initial Attitude Vector
%
% ecen  = [1]*(3.3*10.^[-4]);  %Error boundary in Centroid position [Rad]
%                              (1 pixel error)
% ecat  = [15]*10.^[-3];       %Error boundary in Catalog search [Rad]
% Nfake = 2;                   %Number of False Spots to place in Camera Frame
% MRSS  = 4;                   %Minimum required stars for a solution
%
% n_iter = 10;    %Number of iterations to run for Probability of Error

Initialize

close all; clear all; clc;
profile off

USER INPUTS

Mag_Cut = 3.5;      %Star brightness value cutoff
```

```
FOV     = 50;       %Radius of FOV [DEG]

ViewVec = [187 54]; %Initial Attitude Vector [Deg] (Big Dipper)
rot     = 90;       %Rotational image angle [Deg]
IA      = [];       %Estimated Initial Attitude Vector

ecen  = [1:3]*(3.3*10.^[-4]); %Error boundary in Centroid position [Rad]
ecat  = [1:5]*10.^[-3];       %Error boundary in Catalog search [Rad]
Nfake = [0:3];                %Number of False Spots to place in Camera Frame
MRSS  = 4;                    %Minimum required stars for a solution

n_iter = 1;         %Number of iterations to run for Probability of Error

INPUT CHECK

% Date Folder to save output
dirDate = datestr(now(),'mmmdd-yy-HH_MM');
if ~exist(dirDate,'dir')
   mkdir(dirDate);
end

%Initial Parameters
n1 = length(ecat);
n2 = length(ecen);
n3 = length(Nfake);

%Randomized Line of Sight of camera (If no View Vector is inputted)
if isempty(ViewVec) && isempty(IA)

   NumViews = 100;

   RA      = 0+(360-0).*rand(NumViews,1);
   Dec     = -90+(180).*rand(NumViews,1);
   ViewVec = [RA Dec];
   rot     = 0+(360).*rand(NumViews,1);
   n4      = NumViews;

else
    n4 = 1;
end

%Parameters for waitbar
I  = 0;                 %Counter for waitbar
II = n1*n2*n3*n_iter*n4;   %Waitbar limit

DATA PROCESSING
GET Catalogs and Feature Lists

if exist(['HIP_',num2str(Mag_Cut),'.mat'],'file') == 2

   catalog = load(['HIP_',num2str(Mag_Cut),'.mat']);

elseif Mag_Cut <= 6 && Mag_Cut >= 1

   fprintf('\nCreating new tables for mag. %g stars\n',Mag_Cut)
```

```
    mc   = ceil(Mag_Cut);
    cata = load(['HIP_',num2str(mc),'.mat']);
    ii   = [cata.cat.Mag] <= 3.5;
    cat  = cata.cat(ii);

    save(['HIP_',num2str(Mag_Cut),'.mat'],'cat')
    catalog = load(['HIP_',num2str(Mag_Cut),'.mat']);

else
    error('Mag_Cut outside permisible range.')
end

%Develop/Create Feature List Database
CompDual_Feature_Extract(Mag_Cut,FOV)
Triad_Feature_Extract(Mag_Cut,FOV)
CompTriad_Feature_Extract(Mag_Cut,FOV)
Pyramid_Feature_Extract(Mag_Cut,FOV)

%Load in Feature Lists
featurelist1      = load(['CompDuelStar_M',num2str(Mag_Cut),'_F',num2str(FOV),'.mat']);
featurelist2      = load(['TriadStar_M',num2str(Mag_Cut),'_F',num2str(FOV),'.mat']);
featurelist3      = load(['CompTriadStar_M',num2str(Mag_Cut),'_F',num2str(FOV),'.mat']);
featurelist4.feat = featurelist3.feat;
featurelist5      = load(['PyramidStar_M',num2str(Mag_Cut),'_F',num2str(FOV),'.mat']);
featurelist6.feat = featurelist5.feat;
featurelist7.feat = featurelist5.feat;
featurelist8.feat = featurelist5.feat;

If Given An Initial Attitude ----------------------------------------

if ~isempty(IA)

    k = 0;
    IA = [cosd(IA(1))*cosd(IA(2))... X [rad]
          sind(IA(1))*cosd(IA(2))... Y [rad]
          sind(IA(2))];             %Z [rad]

    fov   = FOV*pi/180;
    [~,n] = size(catalog.cat);

    for i = 1:n

        angle = acos(dot(catalog.cat(i).XYZ,IA));

        if angle <= fov

            k       = k+1;
            index(k) = catalog.cat(i).HipID;

        end
    end

    %Two Star w/ Voting
    featurelist1.feat = featurelist1.feat(ismember([featurelist1.feat.HipID1],index));
    featurelist1.feat = featurelist1.feat(ismember([featurelist1.feat.HipID2],index));
```

```
    %Three Star
    featurelist2.feat = featurelist2.feat(ismember([featurelist2.feat.HipID1],index));
    featurelist2.feat = featurelist2.feat(ismember([featurelist2.feat.HipID2],index));
    featurelist2.feat = featurelist2.feat(ismember([featurelist2.feat.HipID3],index));

    %Three Star w/ Voting
    featurelist3.feat = featurelist3.feat(ismember([featurelist3.feat.HipID1],index));
    featurelist3.feat = featurelist3.feat(ismember([featurelist3.feat.HipID2],index));
    featurelist3.feat = featurelist3.feat(ismember([featurelist3.feat.HipID3],index));

    featurelist4.feat = featurelist3.feat;

    %Pyramid
    featurelist5.feat = featurelist5.feat(ismember([featurelist5.feat.HipID1],index));
    featurelist5.feat = featurelist5.feat(ismember([featurelist5.feat.HipID2],index));
    featurelist5.feat = featurelist5.feat(ismember([featurelist5.feat.HipID3],index));

    featurelist6.feat = featurelist5.feat;

    featurelist7.feat = featurelist5.feat;

    featurelist8.feat = featurelist5.feat;
end

%-------------------------------------------------------------------------

fprintf('Tables created\n\n')

Star Selection Based on FOV and Attitude (ViewVec)

fprintf('** Probability of Error in Progress...\n')
handle = waitbar(0,'Testing in progress...','Name','Simulation Testing');
ProfStats(n4,1) = struct('data',[],'View',[],'Method',[]);

for m = 1:n4

    profile on
    [q,R] = View_to_Quat(ViewVec(m,:),rot(m));       %Finds quaternion and rotation matrix
    [Sky] = FOV_star_generator(ViewVec,Mag_Cut,FOV);  %Selects stars w/in view

ID Methods, Probability of Error, and Statistics

    index = 0;
    Prob1 = zeros(n1*n2*n3,7);
    Prob2 = Prob1;
    Prob3 = Prob1;
    Prob4 = Prob1;
    Prob5 = Prob1;
    Prob6 = Prob1;
    Prob7 = Prob1;
    Prob8 = Prob1;

    for i = 1:n1                    %Run to last entry of ecat
        for j = 1:n2                %Run to last entry of ecen
            for k = 1:n3             %Run to last entry of Nfake
```

```
ProbE1 = zeros(n_iter,5);
ProbE2 = ProbE1;
ProbE3 = ProbE1;
ProbE4 = ProbE1;
ProbE5 = ProbE1;
ProbE6 = ProbE1;
ProbE7 = ProbE1;
ProbE8 = ProbE1;

for iterate = 1:n_iter  %Run for # of desired iterations

   %Body Frame
   [spotlist]       = Camera_Ref_Frame(Sky,R,Nfake(k),ecen(j));

   %Two Star Method with Voting
   [starID,~]       = getTwoStar_ID(catalog,featurelist1,spotlist,ecat(i));
   [stats,~]        = IDAccuracy(starID,Sky,MRSS);
   ProbE1(iterate,:) = [stats.RCvalue stats.quality stats.NoSol stats.PercFalse stats.EmptySol];

   %Liebe Three Star Method
   [starID,~]       = getThreeStar_ID(catalog,featurelist2,spotlist,ecat(i));
   [stats,~]        = IDAccuracy(starID,Sky,MRSS);
   ProbE2(iterate,:) = [stats.RCvalue stats.quality stats.NoSol stats.PercFalse stats.EmptySol];

   %Liebe Three Star Method with Voting
   [starID,~]       = getThreeStarVote_ID(catalog,featurelist3,spotlist,ecat(i));
   [stats,~]        = IDAccuracy(starID,Sky,MRSS);
   ProbE3(iterate,:) = [stats.RCvalue stats.quality stats.NoSol stats.PercFalse stats.EmptySol];

   %Comprehensive Three Star Method with Voting (Bratt's Method)
   [starID,~]       = getCompThreeStar_ID(catalog,featurelist4,spotlist,ecat(i));
   [stats,~]        = IDAccuracy(starID,Sky,MRSS);
   ProbE4(iterate,:) = [stats.RCvalue stats.quality stats.NoSol stats.PercFalse stats.EmptySol];

   %Mortari's Pyramid Method
   [starID,~]       = getPyramid_ID(catalog,featurelist5,spotlist,ecat(i));
   [stats,~]        = IDAccuracy(starID,Sky,MRSS);
   ProbE5(iterate,:) = [stats.RCvalue stats.quality stats.NoSol stats.PercFalse stats.EmptySol];

   %Comprehensive Pyramid Method
   [starID,~]       = getCompPyramid_ID(catalog,featurelist6,spotlist,ecat(i));
   [stats,~]        = IDAccuracy(starID,Sky,MRSS);
   ProbE6(iterate,:) = [stats.RCvalue stats.quality stats.NoSol stats.PercFalse stats.EmptySol];

   %Comp. Mod. Pyramid Method
   [starID,~]       = getPyramid_ID_mod(catalog,featurelist7,spotlist,ecat(i));
   [stats,~]        = IDAccuracy(starID,Sky,MRSS);
   ProbE7(iterate,:) = [stats.RCvalue stats.quality stats.NoSol stats.PercFalse stats.EmptySol];

   %Pyramid with Voting
   [starID,~]       = getPyramidVote_ID(catalog,featurelist8,spotlist,ecat(i));
   [stats,~]        = IDAccuracy(starID,Sky,MRSS);
   ProbE8(iterate,:) = [stats.RCvalue stats.quality stats.NoSol stats.PercFalse stats.EmptySol];

   %Wait bar counter and window
   I = I + 1;
```

```
            waitbar(I/II,handle,sprintf('Testing in progress...%2.2f %%',I/II*100))

        end

        %Update Probability of Error for Nfake and ecen
        index = index + 1;
        ProfStats(m).Method(1).Prob(index,:) = [ecat(i) ecen(j) Nfake(k) sum(ProbE1,1)/n_iter];
        ProfStats(m).Method(2).Prob(index,:) = [ecat(i) ecen(j) Nfake(k) sum(ProbE2,1)/n_iter];
        ProfStats(m).Method(3).Prob(index,:) = [ecat(i) ecen(j) Nfake(k) sum(ProbE3,1)/n_iter];
        ProfStats(m).Method(4).Prob(index,:) = [ecat(i) ecen(j) Nfake(k) sum(ProbE4,1)/n_iter];
        ProfStats(m).Method(5).Prob(index,:) = [ecat(i) ecen(j) Nfake(k) sum(ProbE5,1)/n_iter];
        ProfStats(m).Method(6).Prob(index,:) = [ecat(i) ecen(j) Nfake(k) sum(ProbE6,1)/n_iter];
        ProfStats(m).Method(7).Prob(index,:) = [ecat(i) ecen(j) Nfake(k) sum(ProbE7,1)/n_iter];
        ProfStats(m).Method(8).Prob(index,:) = [ecat(i) ecen(j) Nfake(k) sum(ProbE8,1)/n_iter];

        ProfStats(m).data = profile('info');
        ProfStats(m).View = ViewVec(m,:);

      end
    end
  end
  profile off

end

save([dirDate filesep 'SimAnsM',num2str(Mag_Cut),'run.mat'],'ProfStats')
delete(handle)

Post Processing

close all;clc
[SimTime,Table] = SimPostProcess(3.5,ecat,ecen,Nfake);

End of Program.
```

B. Feature List Creation

1. **Two Star Features List**

```
function [] = CompDual_Feature_Extract(magcut,FOV)

% S.Bratt Function to create a catalog of dual star features based on an
% inputed star magnitude cut-off. Creates a list of dot-product angles
% between two stars. The list is constructed of the HIP numbers of the two
% stars and the angle between them in radians.

Hiparcos Catalog Parameters

if exist(['CompDuelStar_M',num2str(magcut),'_F',num2str(FOV),'.mat'],'file') == 2

else

  catalog = load(['HIP_',num2str(magcut),'.mat']);
  N = size(catalog.cat,2);

  if FOV > 10
```

```
        if magcut > 5
            fovr = (FOV/4)*pi/180;
        elseif magcut <= 5 && magcut >= 4
            fovr = (FOV/3)*pi/180;
        elseif magcut < 4 && magcut > 3
            fovr = (FOV/2)*pi/180;
        elseif magcut <= 3
            fovr = (FOV)*pi/180;
        end

    else
        fovr = FOV*pi/180;
    end
```

Feature Extraction

```
    %Initial parameters
    feat(10000000,1) = struct('HipID1',[],'HipID2',[],'theta1',[]);
    L    = 0;

    handle = waitbar(0,'Initializing...');

    %Feature creation
    for j = 1:N-1

        Hip1   = catalog.cat(j).HipID; %Cooresponding HIP# to vector A
        A      = catalog.cat(j).XYZ;   %Desired vector for comparison

        for i = j+1:N

            %Retrieve 2nd vector
            B = catalog.cat(i).XYZ;

            %Find angle between desired vector and 2nd vector
            theta1 = acos(dot(A,B)); %[rad]

            if theta1 <= fovr
                Hip2  = catalog.cat(i).HipID;

                %Update counter
                L   = L + 1;

                %Incremented Feature Table
                feat(L).HipID1 = Hip1;
                feat(L).HipID2 = Hip2;
                feat(L).theta1 = theta1;

            end
        end

        waitbar(j/(N-1),handle,sprintf('Building Two Star Feature Table...%2.1f %%',j/(N-1)*100))

    end

    feat = feat(1:L);
```

```
    delete(handle)

    %Exporting Feature Table to *.MAT file
    save(['CompDuelStar_M',num2str(magcut),'_F',num2str(FOV),'.mat'],'feat','-v6')

end

End of Program.

End
```

2. **Liebe Feature List**

```
function [] = Triad_Feature_Extract(magcut,FOV)

% S.Bratt Function to create a catalog of triad feature based on an
% inputed star magnitude cut-off. Creates a list dot-product angles between
% a star and the next two CLOSEST stars. Also finds the interior
% dot-product angle between those three stars.

Hiparcos Catalog Parameters

if exist(['TriadStar_M',num2str(magcut),'_F',num2str(FOV),'.mat'],'file') == 2

else

    catalog = load(['HIP_',num2str(magcut),'.mat']);
    N = size(catalog.cat,2);

Feature Extraction

    %Initial parameters
    handle = waitbar(0,'Initializing...');
    feat(10000000,1) = struct('HipID1',[],'HipID2',[],'HipID3',[],'theta1',[],'theta2',[],'phi',[]);

    %Feature creation
    for j = 1:N

        A      = catalog.cat(j).XYZ;    %Desired vector for comparison
        B      = 0;            %Initialize 2nd vector
        C      = 0;            %Initialize 3rd vector
        Hip1   = catalog.cat(j).HipID;      %Cooresponding HIP# to vector A
        Hip2   = 0;             %Initialize 2nd HIP#
        Hip3   = 0;             %Initialize 3rd HIP#
        theta1 = 360;           %Initialize 1st angle
        theta2 = 360;           %Initialize 2nd angle

        for i = 1:N
            %Retrieve 2nd vector
            NewXYZ = catalog.cat(i).XYZ;

            if NewXYZ ~= A;
                %Find angle between desired vector and 2nd vector
                theta = acos(dot(A,NewXYZ));
```

```
            %Compare and update angles
            if theta1 < theta && theta < theta2

               theta2 = theta;
               Hip3   = catalog.cat(i).HipID;
               C      = NewXYZ;

            elseif theta < theta1

               theta2 = theta1;
               theta1 = theta;
               Hip3   = Hip2;
               Hip2   = catalog.cat(i).HipID;
               C      = B;
               B      = NewXYZ;

            end
          end
       end

       %Interior vectors and magnitudes
       Vec1 = B-A;
       Vec2 = C-A;
       v1   = sqrt(Vec1(1)^2+Vec1(2)^2+Vec1(3)^2);
       v2   = sqrt(Vec2(1)^2+Vec2(2)^2+Vec2(3)^2);

       %Find interior angle
       phi = acos(dot(Vec1,Vec2)/(v1*v2));

       if theta1 > theta2
           ang1 = theta2;
           ang2 = theta1;
           Hiparc2 = Hip3;
           Hiparc3 = Hip2;
       else
           ang1 = theta1;
           ang2 = theta2;
           Hiparc2 = Hip2;
           Hiparc3 = Hip3;
       end

       %Incremented Feature Table
       feat(j).HipID1 = Hip1;
       feat(j).HipID2 = Hiparc2;
       feat(j).HipID3 = Hiparc3;
       feat(j).theta1 = ang1;
       feat(j).theta2 = ang2;
       feat(j).phi    = phi;

       waitbar(j/N,handle,sprintf('Building Three Star Feature Table...%2.1f %%',j/N*100))

    end

    feat = feat(1:j);

    delete(handle)
```

```
   %Exporting Feature Table to Data file
   save(['TriadStar_M',num2str(magcut),'_F',num2str(FOV),'.mat'],'feat','-v6')

end

End of Program.

End
```

3. **Liebe with Voting and Brätt Feature List**

```
function [] = CompTriad_Feature_Extract(magcut,FOV)

% Comprehensive Tried Feature Extraction based on a magnitude threshold and
% radius of FOV (field of view).

Hiparcos Catalog Parameters

if exist(['CompTriadStar_M',num2str(magcut),'_F',num2str(FOV),'.mat'],'file') == 2

else

   catalog = load(['HIP_',num2str(magcut),'.mat']);
   N = size(catalog.cat,2);

    if FOV > 10

        if magcut > 5
            fovr = (FOV/4)*pi/180;
        elseif magcut <= 5 && magcut >= 4
            fovr = (FOV/3)*pi/180;
        elseif magcut < 4 && magcut > 3
            fovr = (FOV/2)*pi/180;
        elseif magcut <= 3
            fovr = (FOV)*pi/180;
        end

    else
        fovr = FOV*pi/180;
    end

Feature Extraction

   %Initial Parameters
   feat(10000000,1) = struct('HipID1',[],'HipID2',[],'HipID3',[],...
                       'theta1',[],'theta2',[],'phi',[]);

   handle = waitbar(0,'Initializing...');
   L      = 0;

   for i = 1:N

      Hip1 = catalog.cat(i).HipID;
      A    = catalog.cat(i).XYZ;
```

```
for j = 1:N

   if j ~= i

      Hip2   = catalog.cat(j).HipID;
      B      = catalog.cat(j).XYZ;
      theta1 = acos(dot(A,B));

      if theta1 <= fovr

         for k = 1:N

            if k ~= i && k > j

               Hip3   = catalog.cat(k).HipID;
               C      = catalog.cat(k).XYZ;
               theta2 = acos(dot(A,C));

               if theta2 <= fovr

                  Vec1   = B-A;
                  Vec2   = C-A;
                  v1     = sqrt(Vec1(1)^2+Vec1(2)^2+Vec1(3)^2);
                  v2     = sqrt(Vec2(1)^2+Vec2(2)^2+Vec2(3)^2);

                  phi    = acos(dot(Vec1,Vec2)/(v1*v2));

                  if theta1 > theta2
                      ang1 = theta2;
                      ang2 = theta1;
                      Hiparc2 = Hip3;
                      Hiparc3 = Hip2;
                  else
                      ang1 = theta1;
                      ang2 = theta2;
                      Hiparc2 = Hip2;
                      Hiparc3 = Hip3;
                  end

                  %Incremented Feature Table
                  L = L + 1;
                  feat(L).HipID1 = Hip1;
                  feat(L).HipID2 = Hiparc2;
                  feat(L).HipID3 = Hiparc3;
                  feat(L).theta1 = ang1;
                  feat(L).theta2 = ang2;
                  feat(L).phi    = phi;

               end
            end
         end
      end
   end
end

waitbar(i/N,handle,sprintf('Building Comp. Three Star Table...%2.1f %%',i/N*100))
```

```
    end

    feat = feat(1:L);

    delete(handle)

    %Exporting Feature Table to Data file
    save(['CompTriadStar_M',num2str(magcut),'_F',num2str(FOV),'.mat'],'feat','-v6')

end

End of Program.

end
```

4. **All Pyramid Feature Lists**

```
function [] = Pyramid_Feature_Extract(magcut,FOV)

if exist(['PyramidStar_M',num2str(magcut),'_F',num2str(FOV),'.mat'],'file') == 2

else

    catalog = load(['HIP_',num2str(magcut),'.mat']);
    N = size(catalog.cat,2);

    if FOV > 10

        if magcut > 5
            fovr = (FOV/4)*pi/180;
        elseif magcut <= 5 && magcut >= 4
            fovr = (FOV/3)*pi/180;
        elseif magcut < 4 && magcut > 3
            fovr = (FOV/2)*pi/180;
        elseif magcut <= 3
            fovr = (FOV)*pi/180;
        end

    else
        fovr = FOV*pi/180;
    end

Feature Extraction

    feat(1000000,1) = struct('HipID1',[],'HipID2',[],'HipID3',[],...
                             'theta1',[],'theta2',[],'theta3',[],...
                             'phi1',[],'phi2',[],'phi3',[]);

    handle = waitbar(0,'Initializing...');
    L      = 0;

    for i = 1:N-2

        Hip1 = catalog.cat(i).HipID;
        A    = catalog.cat(i).XYZ;
```

```
for j = i+1:N-1

    Hip2   = catalog.cat(j).HipID;
    B      = catalog.cat(j).XYZ;
    theta1 = acos(dot(A,B));

    if theta1 <= fovr

        for k = j+1:N

            Hip3   = catalog.cat(k).HipID;
            C      = catalog.cat(k).XYZ;
            theta2 = acos(dot(A,C));
            theta3 = acos(dot(B,C));

            if theta2 <= fovr

                V12 = B-A;
                V13 = C-A;
                V23 = C-B;

                v12 = sqrt(V12(1)^2+V12(2)^2+V12(3)^2);
                v13 = sqrt(V13(1)^2+V13(2)^2+V13(3)^2);
                v23 = sqrt(V23(1)^2+V23(2)^2+V23(3)^2);

                phi1 = acos(dot(V12,V13)/(v12*v13));
                phi2 = acos(dot(V12,V23)/(v12*v23));
                phi3 = acos(dot(V13,V23)/(v13*v23));

                if theta1 < theta2 && theta1 < theta3
                    if theta2 < theta3
                        H1 = Hip1;   H2 = Hip2;   H3 = Hip3;
                        T1 = theta1; T2 = theta2; T3 = theta3;
                        P1 = phi1;   P2 = phi2;   P3 = phi3;
                    else
                        H1 = Hip2;   H2 = Hip1;   H3 = Hip3;
                        T1 = theta1; T2 = theta3; T3 = theta2;
                        P1 = phi1;   P2 = phi3;   P3 = phi2;
                    end
                elseif theta2 < theta1 && theta2 < theta3
                    if theta1 < theta3
                        H1 = Hip1;   H2 = Hip3;   H3 = Hip2;
                        T1 = theta2; T2 = theta1; T3 = theta3;
                        P1 = phi2;   P2 = phi1;   P3 = phi3;
                    else
                        H1 = Hip3;   H2 = Hip1;   H3 = Hip2;
                        T1 = theta2; T2 = theta3; T3 = theta1;
                        P1 = phi2;   P2 = phi3;   P3 = phi1;
                    end
                elseif theta3 < theta1 && theta3 < theta2
                    if theta1 < theta2
                        H1 = Hip2;   H2 = Hip3;   H3 = Hip1;
                        T1 = theta3; T2 = theta1; T3 = theta2;
                        P1 = phi3;   P2 = phi1;   P3 = phi2;
                    else
```

```
                        H1 = Hip3;   H2 = Hip2;   H3 = Hip1;
                        T1 = theta3; T2 = theta2; T3 = theta1;
                        P1 = phi3;   P2 = phi2;   P3 = phi1;
                    end
                end

                L = L + 1;

                feat(L).HipID1 = H1;
                feat(L).HipID2 = H2;
                feat(L).HipID3 = H3;
                feat(L).theta1 = T1;
                feat(L).theta2 = T2;
                feat(L).theta3 = T3;
                feat(L).phi1   = P1;
                feat(L).phi2   = P2;
                feat(L).phi3   = P3;

            end
        end
    end
    waitbar(i/N,handle,sprintf('Building Comp. Pyramid Star Table...%2.1f %%',i/N*100))
end

feat = feat(1:L);

delete(handle)

%Exporting Feature Table to Data file
save(['PyramidStar_M',num2str(magcut),'_F',num2str(FOV),'.mat'],'feat','-v6')

end

End of Program.

end
```

C. Body to ECI Rotation

```
function [q,R] = View_to_Quat_mod(ViewVec,rot)

%Function for obtaining a rotation matrix and rotation quaternion based on
%a given camera viewing vector. The Z-axis is in line with the bore-sight
%of the camera. This function to be used in conjunction with Parent
%Function: StarSimProgram.m
%
%Created by: Steven Bratt
%
% Input:
%       ViewVec - Vector of Right Ascension [Deg], and Declination [Deg]
%                 from horizon
%       rot     - Angle by which to rotate vectors (right hand rotation
%                 viewed from XY-plane) [Deg]
%
% Outputs:
```

```
%       q       - Quaternion Rotation for coordinate transformation. The
%                 quaternion is [q0 q1 q2 q3] with q0 being the angle of
%                 rotation about the quaternion axis.
%
%       R       - Rotation matrix from ECI to the Body Coordinate System.
%                 Rotation is a ZXZ rotation.
%
% Example:
%
%       ViewVec = [188 54]; %[Deg]
%       rot     = [90];     %[Deg]
%       [q,R]   = View_to_Quat(ViewVec,rot);
%
```

Calculations

```
%Input Check
if length(ViewVec) ~= 2
   error('Incorrect [ViewVec] variable length. Must be length 2.')
elseif isnumeric(ViewVec) ~= 1
   error('[ViewVec] must be a rational numeric vector variable.')
elseif isnumeric(rot) ~= 1 || length(rot) ~= 1
    error('[rot] must be a rational numeric variable of length 1')
end

%Convert to XYZ
xyz = [cosd(ViewVec(1))*cosd(ViewVec(2))... X [rad]
      sind(ViewVec(1))*cosd(ViewVec(2))... Y [rad]
      sind(ViewVec(2))];                   %Z [rad]

%Obtain Euler Angles
theta = atan2(xyz(1),-xyz(2));
phi   = acos(xyz(3));
psi   = -rot*pi/180;

%Construct Rotation Matrix
c1 = cos(theta); c2 = cos(phi); c3 = cos(psi);
s1 = sin(theta); s2 = sin(phi); s3 = sin(psi);

R = [c1*c3-c2*s1*s3  -c1*s3-c2*c3*s1   s1*s2
    c3*s1+c1*c2*s3   c1*c2*c3-s1*s3  -c1*s2
    s2*s3            c3*s2            c2];

%Construct Rotation Quaternion
q = [cos(theta/2)*cos(phi/2)*cos(psi/2)+sin(theta/2)*sin(phi/2)*sin(psi/2)
    sin(theta/2)*cos(phi/2)*cos(psi/2)-cos(theta/2)*sin(phi/2)*sin(psi/2)
    cos(theta/2)*sin(phi/2)*cos(psi/2)+sin(theta/2)*cos(phi/2)*sin(psi/2)
    cos(theta/2)*cos(phi/2)*sin(psi/2)-sin(theta/2)*sin(phi/2)*cos(psi/2)];

End of Program.

end
```

D. Star Field Generator

```
function [Sky] = FOV_star_generator(ViewVec,magcut,FOV)
```

```
%
% Function for generating a truncated star table based on an
% initial pointing vector, a magnitude threshold, and a desired field of
% view. Data is obtained from a modified HIPPARCOS Catalog. Use in
% conjunction with Parent Function: StarSimProgram.m
%
% Created by: Steven Bratt
%
%   Input:
%       ViewVec - Pointing vector in ECI [Deg]
%       magcut  - Magnitude threshold (Any numeric value between -2 and 13)
%       FOV     - Desired field of view [Deg]
%
%   Output:
%       Sky - Field of View (FOV) limited star table outputed as
%             a structured variable
%
%   Example:
%          [Sky] = FOV_star_generator([184 54],4,50)
%   Result:
%           Sky
%             HipID:
%             XYZ:
%

INPUTS
User Inputs

%Boundary Error Check
if FOV < 10;
   error('Field of View (FOV) must be at least 10 deg')
end

%Boundary Error Check
if -2 > magcut || magcut > 13
   error('The Magnitude Cutoff value is beyond the bounds of the Database. Input a new Magnitude Cutoff
value between -2 and 13.')
end

CALCULATIONS

%Convert to XYZ
IA = [cosd(ViewVec(1))*cosd(ViewVec(2))... X [rad]
     sind(ViewVec(1))*cosd(ViewVec(2))... Y [rad]
     sind(ViewVec(2))];                %Z [rad]

%Magnitude Thresholding Cutoff
load(['HIP_',num2str(magcut),'.mat'],'cat');
N = size(cat,2); %Number of stars in truncated catalog

Windowing Truncation

%Initial zero matrix for looping speed
Sky(100,1) = struct('HipID',[],'XYZ',[]);
```

```
%Dummy counting variable
L   = 0;
%Angle of allowance
ANG = (FOV/2)*(pi/180);

%Loop for duration of Catalog size
for i = 1:N

   %Calculate dot product angle from a star to IA
   theta = acos(dot(cat(i).XYZ,IA));

   %Acceptance criteria
   if theta <= ANG

      L = L + 1;                    %Update counter
      Sky(L).HipID = cat(i).HipID;   %Hip # of stars
      Sky(L).XYZ   = cat(i).XYZ;     %XYZ position of stars

   end
end

%Selected stars within FOV of ViewVec
Sky = Sky(1:L);

End of Function.

End

function V = rotateVector(phi,theta,psi,V,direction)
% Function to rotate coordinate systems given a state vector and euler
% angles.  (ZXZ Euler Sequence)
%
% Created by: David Fowler
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
%   input:
%      phi       - (radians)
%      theta     - Inclination (radians)
%      psi       - Argument of Perigee (radians)
%      V         - Vector (X,Y,Z components)
%      direction - If equal to 1 it transforms from local to global else
%                  visa-versa
%   return:
%      V         - Rotated Vector (X,Y,Z components)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%

cPhi = cos(phi);    cTheta = cos(theta);    cPsi = cos(psi);
sPhi = sin(phi);    sTheta = sin(theta);    sPsi = sin(psi);

Q_12 = [[cPhi*cPsi-sPhi*cTheta*sPsi, sPhi*cPsi+cPhi*cTheta*sPsi, sTheta*sPsi];...
   [-cPhi*sPsi-sPhi*cTheta*cPsi, -sPhi*sPsi+cPhi*cTheta*cPsi, sTheta*cPsi];...
   [sPhi*sTheta, -cPhi*sTheta, cTheta]];

if direction == 1
```

```
   V = Q_12' * V;
else
   V = Q_12 * V;
end

end
```

E. Camera and Error Distortion Program

```
function [spotlist] = Camera_Ref_Frame(Sky,R,Nfake,ecen)

% Function code for constructing ECI sky images and rotating them into the
% Camera Body reference frame. Receives 3D vectors and returns 3D vectors
% that have been rotated and introduces error based on Nfake and ecen.
%
% Created by: Steven Bratt
%
%   Inputs:
%       Sky      - Output from FOV_star_generator.m function
%       R        - Rotation matrix
%       Nfake    - Number of false spots to be added to image
%       ecen     - Error tolerance in star/spot Centroiding [rad]
%
%   Outputs:
%       spotlist - Structured array containing 3D vectors of 'spots'
%                  obtained from 'Sky' that have been randomly shifted
%

CALCULATIONS
Data Retrieval and Initial Parameters

%Initial Parameters
N = size(Sky,1);

Rotation Transform from ECI to Camera Frame

%Initial zero matrix for looping speed
Spots = zeros(N,4);

%Rotate and index spots
for i = 1:N

   Spots(i,:)= [i (R*Sky(i).XYZ')']; %Camera Reference Frame [rad]

end

Uniform Random Centroiding Error

%Computer precision limit
if ecen == 0
   ecen = 10^-15;
end

%Input random errors into Spots vectors
for i = 1:N
```

```
    a = -ecen; b = ecen;

    phi = a + (b-a).*rand(1);
    the = a + (b-a).*rand(1);
    psi = a + (b-a).*rand(1);

    V = rotateVector(phi,the,psi,Spots(i,2:4)',1);

    Spots(i,:) = [Spots(i,1) V'];

end
```

Guassian Random False Spots

```
%Randomized location based on given number of false spots
if Nfake == 0
else

    %Mean Value of Spots        %Standard Deviation of Spots
    mu_x = mean(Spots(:,2));    st_x = std(Spots(:,2));
    mu_y = mean(Spots(:,3));    st_y = std(Spots(:,3));
    mu_z = mean(Spots(:,4));    st_z = std(Spots(:,4));

    randspots(:,1) = mu_x+st_x*randn(Nfake,1);   %Random x position
    randspots(:,2) = mu_y+st_y*randn(Nfake,1);   %Random y postion
    randspots(:,3) = mu_z+st_z*randn(Nfake,1);   %Random z position

    %Indexing of random spots and placement in spotlist
    randspots = [(N+1:N+Nfake)' randspots];
    Spots     = [Spots;randspots];

end
```

OUTPUTS/RESULTS

```
%Initialize structured array
spotlist(N+Nfake,1) = struct('spot',[],'XYZ',[]);

%Create structured array
for i = 1:N+Nfake
    spotlist(i).spot = Spots(i,1);
    spotlist(i).XYZ  = Spots(i,2:end);
end
```

End of Function.

```
end
```

F. ID Accuracy Check

```
function [stats,Matrix] = IDAccuracy(starID,Sky,MRSS)

%MRSS: minimum required stars for solution

%Show what?: A full matrix of the spots, votes, Hips, XYZ
```

```
%           The number of false matches (ID with a neg vote)
%           Whether or not there was a false ID (ID with a pos vote)
%           Reliability/Confidence value and Quality of answer

n = length(Sky);
N = length(starID);

if N > n
    PAD = zeros(1,N-n);
else
    PAD = [];
end

for i = 1:n
    if isempty(Sky(i).HipID)
        Sky(i).HipID = 0;
    end
end

Skyz = [Sky.HipID PAD];

stats = struct('trueID',0,'falseID',0,'neutralID',0,'RCvalue',0,...
    'quality',0,'PercFalse',0,'NoSol',0,'EmptySol',0);

if sum([starID.votes]>0) == 0
    stats.trueID    = 0;
    stats.falseID   = 0;
    stats.neutralID = 0;
    stats.PercFalse = 0;
    stats.NoSol     = 0;
    stats.EmptySol  = 100;
elseif sum([starID.HipID]) ~= 0

    for i = 1:N

        %Compare HIP ID from starID to what is in the SKY
        if isequal( starID(i).HipID , Skyz(i) ) == 1

            stats.trueID  = stats.trueID + 1;

        else

            if starID(i).votes > 0

                stats.falseID = stats.falseID + 1;

            elseif starID(i).votes <= 0

                stats.neutralID = stats.neutralID + 1;

            end
        end

        if i == n

            if stats.trueID < MRSS || sum([stats.falseID]) > 0
```

```
            stats.NoSol = 100;
        else
            stats.NoSol = 0;
        end

    end
end

divider = sum([starID.HipID]~=0);
stats.PercFalse = stats.falseID/divider*100;

else
    stats.trueID    = 0;
    stats.falseID   = 0;
    stats.neutralID = 0;
    stats.PercFalse = 0;
    stats.NoSol     = 0;
    stats.EmptySol  = 100;
end

Reliability/Confidence and Quality of votes

%sum of votes / max abs vote / n true stars = Reliability/Confidence

maxabs = max(abs([starID(1:n).votes]));
if  maxabs == 0
    stats.RCvalue = -1;
else
    stats.RCvalue = sum([starID(1:n).votes])/(maxabs*n);
end

stats.quality = sum([starID.votes]);

Matrix.text = ['Votes ','Spot# ','HipID ','TrueH ','X ','Y ','Z '];

data = [[starID.votes]' [starID.spot]' [starID.HipID]' Skyz'...
    vertcat(starID.XYZ)];

Matrix.data = data;

End of Program.

end
```

G. Post Processing and Probability of Error

```
function [SimTime,Table] = SimPostProcess(Mag_Cut,ecat,ecen,Nfake)

%SimTime [total # days]
%Table.Time [average time in sec]

load(['SimAnsM',num2str(Mag_Cut),'run.mat']);

L  = length(ProfStats);
x  = ecat;
n1 = length(ecat);
```

```
n2 = length(ecen);
n3 = length(Nfake);

M = 0;
figure
for i = 1:L
    H = sum([ProfStats(i).data.FunctionTable.TotalTime]);
    M = H+M;
    X = ProfStats(i).View(1);
    Y = ProfStats(i).View(2);

    plot(X,Y,'ko','MarkerSize',40)
    hold on
    plot(X,Y,'k')

end
axis([0 360 -90 90])
set(gca,'XTick',0:30:360)
set(gca,'YTick',-90:15:90)

title(['Simulation Camera View Points in Sky - Mag. ',num2str(Mag_Cut)])
xlabel('Right Ascension, [Deg]')
ylabel('Declination, [Deg]')

fprintf('** Probability of Error Complete **\n')

SimTime = M/3600/24;

stop = 0;

for i = 1:8
    k = 0;

    while stop == 0

        k = k + 1;
        if k > L
            break
        end
        t1 = ProfStats(k).Method(i).Prob(:,6);
        t2 = ProfStats(k).Method(i).Prob(:,7);

        s1 = sum(t1);
        s2 = sum(t2);

        if s2 > 0 && s1 == 0 %|| s2 == 0 && s1 > 0 %1st half: True when error is present, 2nd half: True
when fewer than MRSS are ID'd
            disp(['Method ' num2str(i) ' Verification Failed'])
            disp(' ')
            index = find(t2~=0);
            location.ProfStats = k;
            location.Method    = i;
            location.ProbRow   = index ;
            location
            return
        end
```

```
    end
end

for i = 1:8
    s = zeros(n1*n2*n3,5);
    for j = 1:L

        t = ProfStats(j).Method(i).Prob(:,[1:2 6:8]);

        if j == 1
            s = t;
        else
            s(:,3:5) = s(:,3:5)+t(:,3:5);
        end

    end

    s(:,3:5) = s(:,3:5)/L;
    for k = 1:n1

        a = s(s(:,1)==x(k),:);
        y1(k,1) = sum(a(:,3))/size(a,1); %sol error %
        y2(k,1) = sum(a(:,4))/size(a,1); %False %
        y3(k,1) = sum(a(:,5))/size(a,1); %empty sol %

    end

    if i == 1
        Y1 = y1;
        Y2 = y2;
        Y3 = y3;
    else
        Y1 = [Y1 y1];
        Y2 = [Y2 y2];
        Y3 = [Y3 y3];
    end
end

%
%
strLine = {'-','-.','-','-.',':',':',':',':'};
strShape = {'o','*','s','<','.','*','s','v'};

% Plots based on ECAT ***************************************************
figure
for j = 4:8

    p=plot(x*10^3,Y1(:,j));
    set(p,'LineStyle',strLine{j},'Marker',strShape{j},'MarkerSize',8)
    hold on

end

set(findobj('Type','line'),'Color','k')
set(gca,'XGrid','off','YGrid','on')
set(gca,'XTick',ecat*10^3)
```

```
title(['Averaged Solutions of Sim. vs. Tolerance Data - Mag. ',num2str(Mag_Cut)])
% xlabel('Catalog Tolerance [mrad]')
ylabel('No Solution [%]')
% legend('Two','Liebe','LiebeVote','Bratt','Pyramid','CompPyr','ModPyr','PyrVote','Location','Best')
legend('Bratt','Pyramid','CompPyr','ModPyr','PyrVote','Location','Best')

figure
for j = 1:8

    p=plot(x*10^3,Y2(:,j));
    set(p,'LineStyle',strLine{j},'Marker',strShape{j},'MarkerSize',8)
    hold on

end

set(findobj('Type','line'),'Color','k')
set(gca,'XGrid','off','YGrid','on')
set(gca,'XTick',ecat*10^3)
title(['Avg. False Matches of Sim. Data - Mag. ',num2str(Mag_Cut)])
xlabel('Catalog Tolerance [mrad]')
ylabel('Failed Matches [%]')
legend('Two','Liebe','LiebeVote','Bratt','Pyramid','CompPyr','ModPyr','PyrVote','Location','Best')

figure
for j = 1:8

    p=plot(x*10^3,Y3(:,j));
    set(p,'LineStyle',strLine{j},'Marker',strShape{j},'MarkerSize',8)
    hold on

end

set(findobj('Type','line'),'Color','k')
set(gca,'XGrid','off','YGrid','on')
set(gca,'XTick',ecat*10^3)
title(['Avg. Empty Sol. of Sim. Data - Mag. ',num2str(Mag_Cut)])
xlabel('Catalog Tolerance [mrad]')
ylabel('Empty Sets [%]')
legend('Two','Liebe','LiebeVote','Bratt','Pyramid','CompPyr','ModPyr','PyrVote','Location','Best')

barY1 = sum(Y1,1)/length(ecat);
barY2 = sum(Y2,1)/length(ecat);
barY3 = sum(Y3,1)/length(ecat);

barY = [barY1;barY2;barY3];
bLegend = {'No Solution';'False Match';'Empty'};
figure
colormap(gray)
bar(1:8,barY')

set(gca,'XTickLabel',{'TwoStar','Liebe','LiebeVote','Bratt','Pyr','CompPyr','ModPyr','PyrVote'})

title(['Overall solution probability of Sim. data - Mag. ' num2str(Mag_Cut)])
xlabel('Identification Algorithm')
ylabel('Probability [%]')
legend(bLegend{:},'Location','Best')
```

```
Based on ECEN

clear y1 y2 y3 Y1 Y2 Y3 t a
x = ecen;

for i = 1:8
    s = zeros(n1*n2*n3,5);
    for j = 1:L

        t = ProfStats(j).Method(i).Prob(:,[1:2 6:8]);

        if j == 1
            s = t;
        else
            s(:,3:5) = s(:,3:5)+t(:,3:5);
        end

    end

    s(:,3:5) = s(:,3:5)/L;
    for k = 1:n2

        a = s(s(:,2)==x(k),:);
        y1(k,1) = sum(a(:,3))/size(a,1); %sol error %
        y2(k,1) = sum(a(:,4))/size(a,1); %False %
        y3(k,1) = sum(a(:,5))/size(a,1); %empty sol %

    end

    if i == 1
        Y1 = y1;
        Y2 = y2;
        Y3 = y3;
    else
        Y1 = [Y1 y1];
        Y2 = [Y2 y2];
        Y3 = [Y3 y3];
    end
end

% Plots based on ECEN ****************************************************

figure
subplot(3,1,1)
for j = 4:8

    p=plot(x*10^3,Y1(:,j));
    set(p,'LineStyle',strLine{j},'Marker',strShape{j},'MarkerSize',8)
    hold on

end

% xlabel('Centroid Tolerance [mrad]')

set(findobj('Type','line'),'Color','k')
```

```
set(gca,'XGrid','off','YGrid','on')
set(gca,'XTick',x*10^3)
ylabel('No Solution [%]')
% legend('Two','Liebe','LiebeVote','Bratt','Pyramid','CompPyr','ModPyr','PyrVote','Location','Best')
legend('Bratt','Pyramid','CompPyr','ModPyr','PyrVote','Location','Best')
title(['Averaged Failed Solutions of Sim. vs. Centroid Data - Mag. ',num2str(Mag_Cut)])

% figure
subplot(3,1,2)
for j = 4:8

    p=plot(x*10^3,Y2(:,j));
    set(p,'LineStyle',strLine{j},'Marker',strShape{j},'MarkerSize',8)
    hold on

end

% xlabel('Centroid Tolerance [mrad]')

set(findobj('Type','line'),'Color','k')
set(gca,'XGrid','off','YGrid','on')
set(gca,'XTick',x*10^3)
ylabel('Failed Matches [%]')
% legend('Two','Liebe','LiebeVote','Bratt','Pyramid','CompPyr','ModPyr','PyrVote','Location','Best')
% title(['Avg. % Failed ID''s of Sim. Data - Mag. ',num2str(Mag_Cut)])

figure

for j = 1:8

    p=plot(x*10^3,Y3(:,j));
    set(p,'LineStyle',strLine{j},'Marker',strShape{j},'MarkerSize',8)
    hold on

end

xlabel('Centroid Tolerance [mrad]')

set(findobj('Type','line'),'Color','k')
set(gca,'XGrid','off','YGrid','on')
set(gca,'XTick',x*10^3)
ylabel('Emtpy Solutions [%]')
legend('Two','Liebe','LiebeVote','Bratt','Pyramid','CompPyr','ModPyr','PyrVote','Location','Best')
title(['Avg. Empty Sol. of Sim. Data - Mag. ',num2str(Mag_Cut)])

clear j i k x y a b pot
L = 100;
strName = {'Two Star','Liebe','Liebe w/ Voting','Bratt','Pyramid',...
    'Comp. Pyramid','Mod. Pyramid','Pyramid w/ Voting'};

for j = 1:8

    b = 0;
    for i = 1:n3
        a = 0;
        for k = 1:L
```

```
            index = ProfStats(k).Method(j).Prob(:,3)==Nfake(i);
            pot   = ProfStats(k).Method(j).Prob(index,:);
            a     = pot(:,6:8)+a;
        end
        b = b + a/L;
    end
    b = b/n3;

    x     = unique(pot(:,1));
    y     = unique(pot(:,2));
    [X,Y] = meshgrid(x,y);
    Z1    = reshape(b(:,1),size(X));

    figure
    surf(X,Y,Z1)
    set(gca,'XTick',ecat,'YTick',ecen)
    xlabel('Catalog Tolerance [mrad]')
    ylabel('Centroid Tolerance [mrad]')
    zlabel('Percent [%] Error')
    title(['Solution Probability of Error of ' strName{j} ' method (Mag ' num2str(Mag_Cut) ')'])
    colormap(gray)

    Z2    = reshape(b(:,2),size(X));

    figure
    surf(X,Y,Z2)
    set(gca,'XTick',ecat,'YTick',ecen)
    xlabel('Catalog Tolerance [mrad]')
    ylabel('Centroid Tolerance [mrad]')
    zlabel('Percent [%] Error')
    title(['False ID Probability of Error of ' strName{j} ' method (Mag ' num2str(Mag_Cut) ')'])
    colormap(gray)

    Z3    = reshape(b(:,3),size(X));

    figure
    surf(X,Y,Z3)
    set(gca,'XTick',ecat,'YTick',ecen)
    xlabel('Catalog Tolerance [mrad]')
    ylabel('Centroid Tolerance [mrad]')
    zlabel('Percent [%] Error')
    title(['Empty Solution Probability of Error of ' strName{j} ' method (Mag ' num2str(Mag_Cut) ')'])
    colormap(gray)

end

%7 TwoStar
%8 Voting
%9 ThreeStar
%10 ThreeVote
%11 CompThree
%12 Pyramid
%13 Comp Pyr
%14 Pyr mod
%15 Pyr vote
q = 0;
```

```
for i = 7:15
    t = 0;
    for j = 1:L
        timing = ProfStats(j).data.FunctionTable(i).TotalTime/60;
        t = t+timing;
    end
    q = q + 1;
    t = sum(t)/L;

    Table(q,1).Time = t;
    Table(q,1).Name = ProfStats(j).data.FunctionTable(i).FunctionName;
end
```

II. Star Identification Program Codes

The complete star identification algorithms are given here. Variable names, function calls, mathematical usage, and formatting can all be seen.

A. Two Star with Voting Method

```
function [starID,starIDMod] = getTwoStar_ID(catalog,featurelist,spotlist,ecat)

% 2-star Voting Identification Algorithm
%
% Obtaines the dot-product angles of two stars and catalogs them with a
% vote if the angle is found to match within the field of the 'distlist'
% catalog. The assumption used is if the number of votes for a star is >=
% 75% of the length of 'spotlist' then it is a 'true' star.
%
%   Inputs:
%
%       featurelist - Sub-catalog created by the 'Dual_Feature_Extract.m'
%                     function
%
%       spotlist - Sub-catalog created by the 'Camera_Ref_Frame.m' function
%
%       ecat     - Displacement error tolerance in 'distlist' search
%
%       plots    - 'On'/'Off' command. 'On' will show all plots and
%                  intermediate comments
%
%   Outputs:
%
%       starID   - An N x 5 matrix, where N is the length of spotlist.
%                  Column 1: Entry number
%                  Column 2: HIP number found if catalog match is found
%                  Column 3-5: XYZ of spot
%
%       Outputs starID as a structured array.
```

Feature Extraction

```
N = length(spotlist);
L = 0;
S = N*(N-1)/2; %S = N! / ( 2 * (N-num_star_in_pattern)! )

pattern(S,1) = struct('spot1',[],'spot2',[],'theta1',[]);

for j = 1:N-1

   %Primary spot in rotation
   A      = spotlist(j).XYZ;
   spot1  = spotlist(j).spot;

   for i = j+1:N

      %Secondary spot in rotation
      B     = spotlist(i).XYZ;
      spot2 = spotlist(i).spot;

      %Dot-product angle between spots
      theta = acos(dot(A,B)/(norm(A)*norm(B)));

      %Incremented Feature Table
      L       = L + 1;

      pattern(L).spot1  = spot1;
      pattern(L).spot2  = spot2;
      pattern(L).theta1 = theta;

   end
end

Voting Sequence

[starID,starIDMod] = Voting_Algorithm(catalog,featurelist,spotlist,pattern,ecat,2);

End of Program.

end
```

B. Liebe's Three Star Method

```
function [starID,starIDMod] = getThreeStar_ID(~,featurelist,spotlist,ecat)

% Leibe's 3 star Triad Star Identification Algorithm
%
% Obtaines the dot-products and interior angles of the two closest stars to
% a particular star. This is limited to only the two closest stars
% adjacent to a the star in question.
%
%   Inputs:
%
%       featurelist - Sub-catalog created by the 'Triad_Feature_Extract.m'
%                     function
%
%       spotlist - Sub-catalog created by the 'Camera_Ref_Frame.m' function
```

```
%
%       ecat     - Displacement error tolerance in 'featurelist' search
%
%       Plot     - 'On'/'Off' command. 'On' will show all plots and
%                  intermediate comments
%
%   Outputs:
%
%       starID   - An N x 5 matrix, where N is the length of spotlist.
%                  Column 1: Entry number
%                  Column 2: HIP number found if catalog match is found
%                  Column 3-5: XYZ of spot
%
%       Outputs starID as a structured array.

Feature Extraction into 'pattern'

N = length(spotlist);

pattern(N,1) = struct('spot1',[],'spot2',[],'spot3',[],'theta1',[],...
    'theta2',[],'phi',[]);

for j = 1:N

    A      = spotlist(j).XYZ;
    B      = 0;
    C      = 0;
    spot1  = spotlist(j).spot;
    spot2  = 0;
    spot3  = 0;
    theta1 = 360;
    theta2 = 360;

    for i = 1:N

        New = spotlist(i).XYZ;

        if New ~= A;

            theta = acos(dot(A,New)/(norm(A)*norm(New)));

            if theta < theta2 && theta > theta1

                theta2 = theta;
                spot3  = spotlist(i).spot;
                C      = New;

            elseif theta < theta1

                theta2 = theta1;
                theta1 = theta;
                spot3  = spot2;
                spot2  = spotlist(i).spot;
                C      = B;
                B      = New;
```

```
        end
      end
   end

   %GET interior angle (phi)
   Vec1 = B-A;
   Vec2 = C-A;
   v1   = sqrt(Vec1(1)^2+Vec1(2)^2+Vec1(3)^2);
   v2   = sqrt(Vec2(1)^2+Vec2(2)^2+Vec2(3)^2);
   phi  = acos(dot(Vec1,Vec2)/(v1*v2));

   if theta1 > theta2
      ang1 = theta2;
      ang2 = theta1;
      Spot2 = spot3;
      Spot3 = spot2;
   else
      ang1 = theta1;
      ang2 = theta2;
      Spot2 = spot2;
      Spot3 = spot3;
   end

   %Incremented pattern structure
   pattern(j).spot1  = spot1;
   pattern(j).spot2  = Spot2;
   pattern(j).spot3  = Spot3;
   pattern(j).theta1 = ang1;
   pattern(j).theta2 = ang2;
   pattern(j).phi    = phi;

end

Search Featurelist GET matches

match(N,1) = struct('spot1',[],'spot2',[],'spot3',[],'theta1',[],...
   'theta2',[],'phi',[]);

fAng1 = [featurelist.feat.theta1];
fAng2 = [featurelist.feat.theta2];
fAng3 = [featurelist.feat.phi];

for i = 1:N

   high1 = pattern(i).theta1+ecat;
   low1  = pattern(i).theta1-ecat;
   high2 = pattern(i).theta2+ecat;
   low2  = pattern(i).theta2-ecat;
   high3 = pattern(i).phi+ecat;
   low3  = pattern(i).phi-ecat;

   ind1  = fAng1<=high1;
   ind2  = fAng1>=low1 ;
   ind3  = fAng2<=high2;
   ind4  = fAng2>=low2 ;
   ind5  = fAng3<=high3;
```

```
    ind6  = fAng3>=low3 ;
    index = (ind1 & ind2 & ind3 & ind4 & ind5 & ind6);

    if sum(index) == 0

        match(i).spot1  = 0;
        match(i).spot2  = 0;
        match(i).spot3  = 0;
        match(i).theta1 = 0;
        match(i).theta2 = 0;
        match(i).phi    = 0;

    else

        match(i).spot1  = featurelist.feat(index).HipID1;
        match(i).spot2  = featurelist.feat(index).HipID2;
        match(i).spot3  = featurelist.feat(index).HipID3;
        match(i).theta1 = featurelist.feat(index).theta1;
        match(i).theta2 = featurelist.feat(index).theta2;
        match(i).phi    = featurelist.feat(index).phi;

    end
end

Pairing of Spots to Stars (mini voting)

starID(N,1) = struct('votes',[],'spot',[],'HipID',[],'XYZ',[]);

ps1 = [pattern.spot1];
ps2 = [pattern.spot2];
ps3 = [pattern.spot3];
ms1 = [match.spot1];
ms2 = [match.spot2];
ms3 = [match.spot3];

for i = 1:N

    hip1 = ms1(ps1==i);
    hip2 = ms2(ps2==i);
    hip3 = ms3(ps3==i);

    hip = [hip1 hip2 hip3];
    hip = hip(hip ~= 0);

    uhip  = unique(hip); %Unique set of HIP# found from quick list 'hip'
    s     = size(uhip,2);
    votes = zeros(s,1);

    if ~isempty(uhip);

        for j = 1:s

            votes(j) = sum(hip == uhip(j));

        end
```

```
        [vote,index] = max(votes);
        starID(i).votes = vote;
        starID(i).HipID = uhip(index);

    else

        starID(i).votes = 0;
        starID(i).HipID = 0;

    end

    starID(i).spot  = i;
    starID(i).XYZ   = spotlist(i).XYZ;

end

starIDMod = starID;

End of Program.

end
```

C. Liebe's Method with Voting

```
function [starID,starIDMod] = getThreeStarVote_ID(catalog,featurelist,spotlist,ecat)

% Leibe's 3 star Triad Star Identification Algorithm w/ Voting
%
% Obtaines the dot-products and interior angles of the two closest stars to
% a particular star. This is limited to only the two closest stars
% adjacent to a the star in question.
%
%   Inputs:
%
%       featurelist - Sub-catalog created by the 'Triad_Feature_Extract.m'
%                     function
%
%       spotlist - Sub-catalog created by the 'Camera_Ref_Frame.m' function
%
%       ecat     - Displacement error tolerance in 'featurelist' search
%
%       Plot     - 'On'/'Off' command. 'On' will show all plots and
%                  intermediate comments
%
%   Outputs:
%
%       starID   - An N x 5 matrix, where N is the length of spotlist.
%                  Column 1: Entry number
%                  Column 2: HIP number found if catalog match is found
%                  Column 3-5: XYZ of spot
%
%       Outputs starID as a structured array.

Feature Extraction

N = length(spotlist);
```

```
pattern(N,1) = struct('spot1',[],'spot2',[],'spot3',[],'theta1',[],...
   'theta2',[],'phi',[]);

for j = 1:N

   A      = spotlist(j).XYZ;
   B      = 0;
   C      = 0;
   spot1  = spotlist(j).spot;
   spot2  = 0;
   spot3  = 0;
   theta1 = 360;
   theta2 = 360;

   for i = 1:N

      New = spotlist(i).XYZ;

      if New ~= A;

         theta = acos(dot(A,New)/(norm(A)*norm(New)));

         if theta < theta2 && theta > theta1

            theta2 = theta;
            spot3  = spotlist(i).spot;
            C      = New;

         elseif theta < theta1

            theta2 = theta1;
            theta1 = theta;
            spot3  = spot2;
            spot2  = spotlist(i).spot;
            C      = B;
            B      = New;

         end
      end
   end

   %GET interior angle (phi)
   Vec1 = B-A;
   Vec2 = C-A;
   v1   = sqrt(Vec1(1)^2+Vec1(2)^2+Vec1(3)^2);
   v2   = sqrt(Vec2(1)^2+Vec2(2)^2+Vec2(3)^2);
   phi  = acos(dot(Vec1,Vec2)/(v1*v2));

   if theta1 > theta2
       ang1 = theta2;
       ang2 = theta1;
       Spot2 = spot3;
       Spot3 = spot2;
   else
       ang1 = theta1;
```

```
        ang2 = theta2;
        Spot2 = spot2;
        Spot3 = spot3;
    end

    %Incremented pattern structure
    pattern(j).spot1  = spot1;
    pattern(j).spot2  = Spot2;
    pattern(j).spot3  = Spot3;
    pattern(j).theta1 = ang1;
    pattern(j).theta2 = ang2;
    pattern(j).phi    = phi;

end
```

Voting Algorithm

```
[starID,starIDMod] = Voting_Algorithm(catalog,featurelist,spotlist,pattern,ecat,3);
```

End of Program.

```
end
```

D. Brätt's Three Star Comprehensive with Voting

```
function [starID,starIDMod] = getCompThreeStar_ID(catalog,featurelist,spotlist,ecat)

% Leibe's 3 star Triad Star Identification Algorithm with Voting Method
%
% Obtaines the dot-products and interior angles of the two closest stars to
% a particular star. This is limited to only the two closest stars
% adjacent to a the star in question.
%
%   Inputs:
%
%       featurelist - Sub-catalog created by the 'Triad_Feature_Extract.m'
%                     function
%
%       spotlist - Sub-catalog created by the 'Camera_Ref_Frame.m' function
%
%       ecat     - Displacement error tolerance in 'featurelist' search
%
%       plots    - 'On'/'Off' command. 'On' will show all plots and
%                  intermediate comments
%
%   Outputs:
%
%       starID   - An N x 5 matrix, where N is the length of spotlist.
%                  Column 1: Entry number
%                  Column 2: HIP number found if catalog match is found
%                  Column 3-5: XYZ of spot
%
%       Outputs starID as a structured array.
```

Feature Extraction

```
N = length(spotlist);
L = 0;
S = N*(N-1)*(N-2)/2; %S = N! / ( 2 * (N-num_star_in_pattern)! )

pattern(S,1) = struct('spot1',[],'spot2',[],'spot3',[],'theta1',[],...
   'theta2',[],'phi',[]);

for i = 1:N

   spot1 = spotlist(i).spot;
   A     = spotlist(i).XYZ;


   for j = 1:N

      if i ~= j

         spot2 = spotlist(j).spot;
         B     = spotlist(j).XYZ;
         ang1  = acos(dot(A,B)/(norm(A)*norm(B)));

         for k = 1:N

            if k ~= i && k > j

               spot3 = spotlist(k).spot;
               C     = spotlist(k).XYZ;
               ang2  = acos(dot(A,C)/(norm(A)*norm(C)));

               Vec1   = B-A;
               Vec2   = C-A;
               v1     = sqrt(Vec1(1)^2+Vec1(2)^2+Vec1(3)^2);
               v2     = sqrt(Vec2(1)^2+Vec2(2)^2+Vec2(3)^2);

               phi    = acos(dot(Vec1,Vec2)/(v1*v2));

               if ang1 > ang2
                   theta1 = ang2;
                   theta2 = ang1;
                   Spot2  = spot3;
                   Spot3  = spot2;
               else
                   theta1 = ang1;
                   theta2 = ang2;
                   Spot2  = spot2;
                   Spot3  = spot3;
               end

               %Incremented Feature Table
               L = L + 1;

               pattern(L).spot1  = spot1;
               pattern(L).spot2  = Spot2;
               pattern(L).spot3  = Spot3;
               pattern(L).theta1 = theta1;
               pattern(L).theta2 = theta2;
```

```
                pattern(L).phi    = phi;
            end
          end
        end
    end
end
```

Voting Sequence

```
[starID,starIDMod] = Voting_Algorithm(catalog,featurelist,spotlist,pattern,ecat,3);
```

End of Program.

```
end
```

E. Constrained Pyramid Method

```
function [starID,starIDMod] = getPyramid_ID(~,featurelist,spotlist,ecat)

%Mortari's Pyramid Algorithm. Creates a list of patterns described by 6
%features each using 3 stars. These patterns are checked against a feature
%list, one pattern at a time, using a 4th star as a verification tool. If
%all 4 spots match to stars in the feature list, then the spots are marked,
%their HIP# recorded, and the output is a table of all spots in the image,
%their location, and only the 4 spots that were recognized will have a HIP
%ID.


%Length of spotlist
N = length(spotlist);

%Initialize structure
starID(N,1) = struct('votes',[],'spot',[],'HipID',[],'XYZ',[]);

%Pad with zeros and with known info
[starID.votes] = deal(0);
[starID.spot]  = spotlist.spot;
[starID.HipID] = deal(0);
[starID.XYZ]   = spotlist.XYZ;

%Preallocated variable
starIDMod = starID;

Algorithm Model

if N < 4 % Early Failure Detection, requires min. 4 spots in image to proc.

else %Process Image and Analyze

Pattern Creation

    %Pattern size: N! / ( 6 * [N-3]! )
    S = N*(N-1)*(N-2)/6;

    %Initialize structured array
    pattern(S,1) = struct('spot1',[],'spot2',[],'spot3',[],...
```

```
                'theta1',[],'theta2',[],'theta3',[],...
                'phi1',[],'phi2',[],'phi3',[]);
%Pattern Counter
L = 0;

%Create Patterns
for i = 1:N-2

    %Find 1st spot and vector
    spot1 = spotlist(i).spot;
    A     = spotlist(i).XYZ;

    for j = i+1:N-1

        %Find 2nd spot and vector
        spot2  = spotlist(j).spot;
        B      = spotlist(j).XYZ;

        %Find 1st Feature
        theta1 = acos(dot(A,B));

        for k = j+1:N

            %Find 3rd spot and vector
            spot3  = spotlist(k).spot;
            C      = spotlist(k).XYZ;

            %Find 2nd and 3rd Features
            theta2 = acos(dot(A,C)/(norm(A)*norm(C)));
            theta3 = acos(dot(B,C)/(norm(B)*norm(C)));

            %Calculate Interior Angles
            V12 = B-A;
            V13 = C-A;
            V23 = C-B;
            v12 = sqrt(V12(1)^2+V12(2)^2+V12(3)^2);
            v13 = sqrt(V13(1)^2+V13(2)^2+V13(3)^2);
            v23 = sqrt(V23(1)^2+V23(2)^2+V23(3)^2);

            %Interior angles: Features 4->6
            phi1 = acos(dot(V12,V13)/(v12*v13));
            phi2 = acos(dot(V12,V23)/(v12*v23));
            phi3 = acos(dot(V13,V23)/(v13*v23));

            %Sort Features based on smallest theta angle
            if theta1 < theta2 && theta1 < theta3
                if theta2 < theta3
                    S1 = spot1;  S2 = spot2;  S3 = spot3;
                    T1 = theta1; T2 = theta2; T3 = theta3;
                    P1 = phi1;   P2 = phi2;   P3 = phi3;
                else
                    S1 = spot2;  S2 = spot1;  S3 = spot3;
                    T1 = theta1; T2 = theta3; T3 = theta2;
                    P1 = phi1;   P2 = phi3;   P3 = phi2;
                end
            elseif theta2 < theta1 && theta2 < theta3
```

```
                if theta1 < theta3
                    S1 = spot1;  S2 = spot3;  S3 = spot2;
                    T1 = theta2; T2 = theta1; T3 = theta3;
                    P1 = phi2;   P2 = phi1;   P3 = phi3;
                else

                    S1 = spot3;  S2 = spot1;  S3 = spot2;
                    T1 = theta2; T2 = theta3; T3 = theta1;
                    P1 = phi2;   P2 = phi3;   P3 = phi1;
                end
            elseif theta3 < theta1 && theta3 < theta2
                if theta1 < theta2
                    S1 = spot2;  S2 = spot3;  S3 = spot1;
                    T1 = theta3; T2 = theta1; T3 = theta2;
                    P1 = phi3;   P2 = phi1;   P3 = phi2;
                else
                    S1 = spot3;  S2 = spot2;  S3 = spot1;
                    T1 = theta3; T2 = theta2; T3 = theta1;
                    P1 = phi3;   P2 = phi2;   P3 = phi1;
                end
            end

            %Update Pattern Counter
            L = L + 1;

            %Input Pattern
            pattern(L).spot1 = S1;
            pattern(L).spot2 = S2;
            pattern(L).spot3 = S3;

            pattern(L).theta1 = T1;
            pattern(L).theta2 = T2;
            pattern(L).theta3 = T3;

            pattern(L).phi1 = P1;
            pattern(L).phi2 = P2;
            pattern(L).phi3 = P3;

        end
    end
end
```

Pattern Identification

```
%Pre-allocate for increased index search speed
fAng1 = [featurelist.feat.theta1];
fAng2 = [featurelist.feat.theta2];
fAng3 = [featurelist.feat.theta3];

fPhi1 = [featurelist.feat.phi1];
fPhi2 = [featurelist.feat.phi2];
fPhi3 = [featurelist.feat.phi3];

patsp1 = [pattern.spot1];
patsp2 = [pattern.spot2];
patsp3 = [pattern.spot3];
```

```
%Array of spots
ns = 1:N;

%
for i = 1:S

    %New variables for ease in coding
    starnum(1) = pattern(i).spot1;
    starnum(2) = pattern(i).spot2;
    starnum(3) = pattern(i).spot3;

    %Search for next 'spots' to build future triads
    index    = ( ns>starnum(1) & ns>starnum(2) & ns>starnum(3) );
    star4set = ns( index ~= 0 );

    %
    for j = 1:length(star4set)

        %New 4th spot chosen
        starnum(4) = star4set(j);

        %Search for new triads using all 4 spots using indexing
        i1 = (patsp1 == starnum(1) |...
              patsp1 == starnum(2) |...
              patsp1 == starnum(3) |...
              patsp1 == starnum(4));
        i2 = (patsp2 == starnum(1) |...
              patsp2 == starnum(2) |...
              patsp2 == starnum(3) |...
              patsp2 == starnum(4));
        i3 = (patsp3 == starnum(1) |...
              patsp3 == starnum(2) |...
              patsp3 == starnum(3) |...
              patsp3 == starnum(4));

        %Create new Image Pyramid
        Pyramid  = pattern(i1&i2&i3);

        %Initialize Featurelist Pyramid
        FPyramid = cell(4,1);

        %Add tolerances and search Featurelist
        for k = 1:4

            %Search tolerance added to image
            H1 = Pyramid(k).theta1+ecat; L1 = Pyramid(k).theta1-ecat;
            H2 = Pyramid(k).theta2+ecat; L2 = Pyramid(k).theta2-ecat;
            H3 = Pyramid(k).theta3+ecat; L3 = Pyramid(k).theta3-ecat;

            H4 = Pyramid(k).phi1+ecat;   L4 = Pyramid(k).phi1-ecat;
            H5 = Pyramid(k).phi2+ecat;   L5 = Pyramid(k).phi2-ecat;
            H6 = Pyramid(k).phi3+ecat;   L6 = Pyramid(k).phi3-ecat;

            %Indexing of tolerances
            ind1  = fAng1 <= H1; ind2  = fAng1 >= L1;
```

```
    ind3  = fAng2 <= H2; ind4  = fAng2 >= L2;
    ind5  = fAng3 <= H3; ind6  = fAng3 >= L3;
    ind7  = fPhi1 <= H4; ind8  = fPhi1 >= L4;
    ind9  = fPhi2 <= H5; ind10 = fPhi2 >= L5;
    ind11 = fPhi3 <= H6; ind12 = fPhi3 >= L6;

    %Location in Featurelist for match
    Findex = (ind1 & ind2 & ind3 & ind4  & ind5  & ind6 &...
            ind7 & ind8 & ind9 & ind10 & ind11 & ind12);

    %New Featurelist Pyramid
    FPyramid(k) = {featurelist.feat(Findex)};

end

%Check if F.Pyramid is empty
L1 = length(FPyramid{1}); L2 = length(FPyramid{2});
L3 = length(FPyramid{3}); L4 = length(FPyramid{4});

%Verify if F.Pyramid is valid for use, else use new 4th spot
if L1 == 0 || L2 == 0 || L3 == 0 || L4 == 0

else

    list = zeros(12,2);
    n    = 0;

    %First Identification Process
    for k = 1:4

        h1  = [FPyramid{k}.HipID1];   %HIP's found
        u   = unique(h1);             %Unique values of HIP
        lu  = length(u);              %Number of unique values
        tag = zeros(lu,1);            %Pre-allocated matrix for loop speed

        for m = 1:lu
            index1 = h1 == u(m);     %index: all locations of value u(m) in HIP
            tag(m) = sum(index1);    %TAG: number of times value u(m) is found in HIP
        end
        [~,index1] = max(tag);       %Location in TAG for max similar entries of u

        h2  = [FPyramid{k}.HipID2];   %HIP's found
        u   = unique(h2);             %Unique values of HIP
        lu  = length(u);              %Number of unique values
        tag = zeros(lu,1);            %Pre-allocated matrix for loop speed

        for m = 1:lu
            index2 = h2 == u(m);     %index: all locations of value u(m) in HIP
            tag(m) = sum(index2);    %TAG: number of times value u(m) is found in HIP
        end
        [~,index2] = max(tag);       %Location in TAG for max similar entries of u

        h3  = [FPyramid{k}.HipID3];   %HIP's found
        u   = unique(h3);             %Unique values of HIP
        lu  = length(u);              %Number of unique values
        tag = zeros(lu,1);            %Pre-allocated matrix for loop speed
```

```
            for m = 1:lu
                index3 = h3 == u(m);     %index: all locations of value u(m) in HIP
                tag(m) = sum(index3);    %TAG: number of times value u(m) is found in HIP
            end
            [~,index3] = max(tag);       %Location in TAG for max similar entries of u

            %Update LIST entry and counter
            n = n(end)+1:n(end)+3;

            list(n,:) = [Pyramid(k).spot1 h1(index1)
                         Pyramid(k).spot2 h2(index2)
                         Pyramid(k).spot3 h3(index3)];

        end

        %Final Identification and Output
        for k = 1:N

            %Find all HIP's for spot 'k'
            H   = list( list(:,1) == k ,2);

            if ~isempty(H)

                u   = unique(H);             %Unique values of HIP
                lu  = length(u);             %Number of unique values
                tag = zeros(lu,1);           %Pre-allocated matrix for loop speed

                for m = 1:lu
                    index = H == u(m);       %index: all locations of value u(m) in HIP
                    tag(m) = sum(index);     %TAG: number of times value u(m) is found in HIP
                end

                [ntags,index] = max(tag);    %Location in TAG for max similar entries of u

                %Input results to Structured output
                starID(k).votes = ntags;
                starID(k).spot  = k;
                starID(k).HipID = u(index);
                starID(k).XYZ   = spotlist(k).XYZ;

                %[Variable for use in DAVID FOWLER codes]
                starIDMod = starID;

            else
            end
        end

        %Identification has completed and results recorded,
        %terminate further need to identify image
        return

    end
  end
end
```

```
end

End of Program.

end
```

F. Comprehensive Pyramid Method

```
function [starID,starIDMod] = getCompPyramid_ID(~,featurelist,spotlist,ecat)

%Mortari's Pyramid Algorithm. Creates a list of patterns described by 6
%features each using 3 stars. These patterns are checked against a feature
%list, one pattern at a time, using a 4th star as a verification tool. If
%all 4 spots match to stars in the feature list, then the spots are marked,
%their HIP# recorded, and the output is a table of all spots in the image,
%their location, and only the 4 spots that were recognized will have a HIP
%ID.


%Length of spotlist
N = length(spotlist);

%Initialize structure
starID(N,1) = struct('votes',[],'spot',[],'HipID',[],'XYZ',[]);

%Pad with zeros and with known info
[starID.votes] = deal(0);
[starID.spot]  = spotlist.spot;
[starID.HipID] = deal(0);
[starID.XYZ]   = spotlist.XYZ;

%Preallocated variable
starIDMod = starID;

Algorithm Model

if N < 4 % Early Failure Detection, requires min. 4 spots in image to proc.

else %Process Image and Analyze

Pattern Creation

    %Pattern size: N! / ( 6 * [N-3]! )
    S = N*(N-1)*(N-2)/6;

    %Initialize structured array
    pattern(S,1) = struct('spot1',[],'spot2',[],'spot3',[],...
                          'theta1',[],'theta2',[],'theta3',[],...
                          'phi1',[],'phi2',[],'phi3',[]);
    %Pattern Counter
    L = 0;

    %Create Patterns
    for i = 1:N-2

        %Find 1st spot and vector
```

```
spot1 = spotlist(i).spot;
A     = spotlist(i).XYZ;

for j = i+1:N-1

    %Find 2nd spot and vector
    spot2  = spotlist(j).spot;
    B      = spotlist(j).XYZ;

    %Find 1st Feature
    theta1 = acos(dot(A,B));

    for k = j+1:N

        %Find 3rd spot and vector
        spot3  = spotlist(k).spot;
        C      = spotlist(k).XYZ;

        %Find 2nd and 3rd Features
        theta2 = acos(dot(A,C)/(norm(A)*norm(C)));
        theta3 = acos(dot(B,C)/(norm(B)*norm(C)));

        %Calculate Interior Angles
        V12 = B-A;
        V13 = C-A;
        V23 = C-B;
        v12 = sqrt(V12(1)^2+V12(2)^2+V12(3)^2);
        v13 = sqrt(V13(1)^2+V13(2)^2+V13(3)^2);
        v23 = sqrt(V23(1)^2+V23(2)^2+V23(3)^2);

        %Interior angles: Features 4->6
        phi1 = acos(dot(V12,V13)/(v12*v13));
        phi2 = acos(dot(V12,V23)/(v12*v23));
        phi3 = acos(dot(V13,V23)/(v13*v23));

        %Sort Features based on smallest theta angle
        if theta1 < theta2 && theta1 < theta3
            if theta2 < theta3
                S1 = spot1;  S2 = spot2;  S3 = spot3;
                T1 = theta1; T2 = theta2; T3 = theta3;
                P1 = phi1;   P2 = phi2;   P3 = phi3;
            else
                S1 = spot2;  S2 = spot1;  S3 = spot3;
                T1 = theta1; T2 = theta3; T3 = theta2;
                P1 = phi1;   P2 = phi3;   P3 = phi2;
            end
        elseif theta2 < theta1 && theta2 < theta3
            if theta1 < theta3
                S1 = spot1;  S2 = spot3;  S3 = spot2;
                T1 = theta2; T2 = theta1; T3 = theta3;
                P1 = phi2;   P2 = phi1;   P3 = phi3;
            else

                S1 = spot3;  S2 = spot1;  S3 = spot2;
                T1 = theta2; T2 = theta3; T3 = theta1;
                P1 = phi2;   P2 = phi3;   P3 = phi1;
```

```
                end
            elseif theta3 < theta1 && theta3 < theta2
                if theta1 < theta2
                    S1 = spot2;  S2 = spot3;  S3 = spot1;
                    T1 = theta3; T2 = theta1; T3 = theta2;
                    P1 = phi3;   P2 = phi1;   P3 = phi2;
                else
                    S1 = spot3;  S2 = spot2;  S3 = spot1;
                    T1 = theta3; T2 = theta2; T3 = theta1;
                    P1 = phi3;   P2 = phi2;   P3 = phi1;
                end
            end

            %Update Pattern Counter
            L = L + 1;

            %Input Pattern
            pattern(L).spot1 = S1;
            pattern(L).spot2 = S2;
            pattern(L).spot3 = S3;

            pattern(L).theta1 = T1;
            pattern(L).theta2 = T2;
            pattern(L).theta3 = T3;

            pattern(L).phi1 = P1;
            pattern(L).phi2 = P2;
            pattern(L).phi3 = P3;

        end
    end
end
```

Pattern Identification

```
%Pre-allocate for increased index search speed
fAng1 = [featurelist.feat.theta1];
fAng2 = [featurelist.feat.theta2];
fAng3 = [featurelist.feat.theta3];

fPhi1 = [featurelist.feat.phi1];
fPhi2 = [featurelist.feat.phi2];
fPhi3 = [featurelist.feat.phi3];

patsp1 = [pattern.spot1];
patsp2 = [pattern.spot2];
patsp3 = [pattern.spot3];

%Array of spots
ns = 1:N;

%
for i = 1:S

    %New variables for ease in coding
    starnum(1) = pattern(i).spot1;
```

```
starnum(2) = pattern(i).spot2;
starnum(3) = pattern(i).spot3;

%Search for next 'spots' to build future triads
index    = ( ns>starnum(1) & ns>starnum(2) & ns>starnum(3) );
star4set = ns( index ~= 0 );

%
for j = 1:length(star4set)

    %New 4th spot chosen
    starnum(4) = star4set(j);

    %Search for new triads using all 4 spots using indexing
    i1 = (patsp1 == starnum(1) |...
          patsp1 == starnum(2) |...
          patsp1 == starnum(3) |...
          patsp1 == starnum(4));
    i2 = (patsp2 == starnum(1) |...
          patsp2 == starnum(2) |...
          patsp2 == starnum(3) |...
          patsp2 == starnum(4));
    i3 = (patsp3 == starnum(1) |...
          patsp3 == starnum(2) |...
          patsp3 == starnum(3) |...
          patsp3 == starnum(4));

    %Create new Image Pyramid
    Pyramid  = pattern(i1&i2&i3);

    %Initialize Featurelist Pyramid
    FPyramid = cell(4,1);

    if length(Pyramid) < 4
       dbstop getCompPyramid_ID.m at 201
    end

    %Add tolerances and search Featurelist
    for k = 1:4

        %Search tolerance added to image
        H1 = Pyramid(k).theta1+ecat; L1 = Pyramid(k).theta1-ecat;
        H2 = Pyramid(k).theta2+ecat; L2 = Pyramid(k).theta2-ecat;
        H3 = Pyramid(k).theta3+ecat; L3 = Pyramid(k).theta3-ecat;

        H4 = Pyramid(k).phi1+ecat;   L4 = Pyramid(k).phi1-ecat;
        H5 = Pyramid(k).phi2+ecat;   L5 = Pyramid(k).phi2-ecat;
        H6 = Pyramid(k).phi3+ecat;   L6 = Pyramid(k).phi3-ecat;

        %Indexing of tolerances
        ind1  = fAng1 <= H1; ind2  = fAng1 >= L1;
        ind3  = fAng2 <= H2; ind4  = fAng2 >= L2;
        ind5  = fAng3 <= H3; ind6  = fAng3 >= L3;
        ind7  = fPhi1 <= H4; ind8  = fPhi1 >= L4;
        ind9  = fPhi2 <= H5; ind10 = fPhi2 >= L5;
        ind11 = fPhi3 <= H6; ind12 = fPhi3 >= L6;
```

```
    %Location in Featurelist for match
    Findex = (ind1 & ind2 & ind3 & ind4  & ind5  & ind6 &...
            ind7 & ind8 & ind9 & ind10 & ind11 & ind12);

    %New Featurelist Pyramid
    FPyramid(k) = {featurelist.feat(Findex)};

end

%Check if F.Pyramid is empty
L1 = length(FPyramid{1}); L2 = length(FPyramid{2});
L3 = length(FPyramid{3}); L4 = length(FPyramid{4});

%Verify if F.Pyramid is valid for use, else use new 4th spot
if L1 == 0 || L2 == 0 || L3 == 0 || L4 == 0

else

    list = zeros(12,2);
    n    = 0;

    %First Identification Process
    for k = 1:4

        h1  = [FPyramid{k}.HipID1];   %HIP's found
        u   = unique(h1);             %Unique values of HIP
        lu  = length(u);              %Number of unique values
        tag = zeros(lu,1);            %Pre-allocated matrix for loop speed

        for m = 1:lu
            index1 = h1 == u(m);     %index: all locations of value u(m) in HIP
            tag(m) = sum(index1);    %TAG: number of times value u(m) is found in HIP
        end
        [~,index1] = max(tag);       %Location in TAG for max similar entries of u

        h2  = [FPyramid{k}.HipID2];   %HIP's found
        u   = unique(h2);             %Unique values of HIP
        lu  = length(u);              %Number of unique values
        tag = zeros(lu,1);            %Pre-allocated matrix for loop speed

        for m = 1:lu
            index2 = h2 == u(m);     %index: all locations of value u(m) in HIP
            tag(m) = sum(index2);    %TAG: number of times value u(m) is found in HIP
        end
        [~,index2] = max(tag);       %Location in TAG for max similar entries of u

        h3  = [FPyramid{k}.HipID3];   %HIP's found
        u   = unique(h3);             %Unique values of HIP
        lu  = length(u);              %Number of unique values
        tag = zeros(lu,1);            %Pre-allocated matrix for loop speed

        for m = 1:lu
            index3 = h3 == u(m);     %index: all locations of value u(m) in HIP
            tag(m) = sum(index3);    %TAG: number of times value u(m) is found in HIP
        end
```

```
                [~,index3] = max(tag);        %Location in TAG for max similar entries of u

                %Update LIST entry and counter
                n = n(end)+1:n(end)+3;

                list(n,:) = [Pyramid(k).spot1 h1(index1)
                             Pyramid(k).spot2 h2(index2)
                             Pyramid(k).spot3 h3(index3)];

            end

            %Final Identification and Output
            for k = 1:N

                %Find all HIP's for spot 'k'
                H   = list( list(:,1) == k ,2);

                if ~isempty(H)

                    u   = unique(H);             %Unique values of HIP
                    lu  = length(u);             %Number of unique values
                    tag = zeros(lu,1);           %Pre-allocated matrix for loop speed

                    for m = 1:lu
                        index = H == u(m);       %index: all locations of value u(m) in HIP
                        tag(m) = sum(index);     %TAG: number of times value u(m) is found in HIP
                    end

                    [ntags,index] = max(tag);     %Location in TAG for max similar entries of u

                    %Input results to Structured output
                    starID(k).votes = ntags;
                    starID(k).spot  = k;
                    starID(k).HipID = u(index);
                    starID(k).XYZ   = spotlist(k).XYZ;

                    %[Variable for use in DAVID FOWLER codes]
                    starIDMod = starID;

                else
                end
            end

            %Identification has completed and results recorded,
            %terminate further need to identify image
%               return

        end
    end
end

end

End of Program.

end
```

G. Modified Pyramid Method

```
function [starID,starIDMod] = getPyramid_ID_mod(~,featurelist,spotlist,ecat)

%Mortari's Pyramid Algorithm. Creates a list of patterns described by 6
%features each using 3 stars. These patterns are checked against a feature
%list, one pattern at a time, using a 4th star as a verification tool. If
%all 4 spots match to stars in the feature list, then the spots are marked,
%their HIP# recorded, and the output is a table of all spots in the image,
%their location, and only the 4 spots that were recognized will have a HIP
%ID.

%Length of spotlist
N = length(spotlist);

%Initialize structure
starID(N,1) = struct('votes',[],'spot',[],'HipID',[],'XYZ',[]);

%Pad with zeros and with known info
[starID.votes] = deal(0);
[starID.spot]  = spotlist.spot;
[starID.HipID] = deal(0);
[starID.XYZ]   = spotlist.XYZ;

%Preallocated variable
starIDMod = starID;

Algorithm Model

if N < 4 % Early Failure Detection, requires min. 4 spots in image to proc.

else %Process Image and Analyze

Pattern Creation

    %Pattern size: N! / ( 6 * [N-3]! )
    S = N*(N-1)*(N-2)/6;

    %Initialize structured array
    pattern(S,1) = struct('spot1',[],'spot2',[],'spot3',[],...
                          'theta1',[],'theta2',[],'theta3',[],...
                          'phi1',[],'phi2',[],'phi3',[]);
    %Pattern Counter
    L = 0;

    %Create Patterns
    for i = 1:N-2

        %Find 1st spot and vector
        spot1 = spotlist(i).spot;
        A     = spotlist(i).XYZ;

        for j = i+1:N-1

            %Find 2nd spot and vector
            spot2  = spotlist(j).spot;
```

```
B     = spotlist(j).XYZ;

%Find 1st Feature
theta1 = acos(dot(A,B));

for k = j+1:N

    %Find 3rd spot and vector
    spot3  = spotlist(k).spot;
    C      = spotlist(k).XYZ;

    %Find 2nd and 3rd Features
    theta2 = acos(dot(A,C)/(norm(A)*norm(C)));
    theta3 = acos(dot(B,C)/(norm(B)*norm(C)));

    if ~isreal(theta2)
        theta2 = 0;
    elseif ~isreal(theta3)
        theta3 = 0;
    end

    %Calculate Interior Angles
    V12 = B-A;
    V13 = C-A;
    V23 = C-B;
    v12 = sqrt(V12(1)^2+V12(2)^2+V12(3)^2);
    v13 = sqrt(V13(1)^2+V13(2)^2+V13(3)^2);
    v23 = sqrt(V23(1)^2+V23(2)^2+V23(3)^2);

    %Interior angles: Features 4->6
    phi1 = acos(dot(V12,V13)/(v12*v13));
    phi2 = acos(dot(V12,V23)/(v12*v23));
    phi3 = acos(dot(V13,V23)/(v13*v23));

    %Sort Features based on smallest theta angle
    if theta1 < theta2 && theta1 < theta3
        if theta2 < theta3
            S1 = spot1;  S2 = spot2;  S3 = spot3;
            T1 = theta1; T2 = theta2; T3 = theta3;
            P1 = phi1;   P2 = phi2;   P3 = phi3;
        else
            S1 = spot2;  S2 = spot1;  S3 = spot3;
            T1 = theta1; T2 = theta3; T3 = theta2;
            P1 = phi1;   P2 = phi3;   P3 = phi2;
        end
    elseif theta2 < theta1 && theta2 < theta3
        if theta1 < theta3
            S1 = spot1;  S2 = spot3;  S3 = spot2;
            T1 = theta2; T2 = theta1; T3 = theta3;
            P1 = phi2;   P2 = phi1;   P3 = phi3;
        else

            S1 = spot3;  S2 = spot1;  S3 = spot2;
            T1 = theta2; T2 = theta3; T3 = theta1;
            P1 = phi2;   P2 = phi3;   P3 = phi1;
        end
```

```
            elseif theta3 < theta1 && theta3 < theta2
                if theta1 < theta2
                    S1 = spot2;  S2 = spot3;  S3 = spot1;
                    T1 = theta3; T2 = theta1; T3 = theta2;
                    P1 = phi3;   P2 = phi1;   P3 = phi2;
                else
                    S1 = spot3;  S2 = spot2;  S3 = spot1;
                    T1 = theta3; T2 = theta2; T3 = theta1;
                    P1 = phi3;   P2 = phi2;   P3 = phi1;
                end
            end

            %Update Pattern Counter
            L = L + 1;

            %Input Pattern
            pattern(L).spot1 = S1;
            pattern(L).spot2 = S2;
            pattern(L).spot3 = S3;

            pattern(L).theta1 = T1;
            pattern(L).theta2 = T2;
            pattern(L).theta3 = T3;

            pattern(L).phi1 = P1;
            pattern(L).phi2 = P2;
            pattern(L).phi3 = P3;

        end
    end
end

Pattern Identification

    %Pre-allocate for increased index search speed
    fAng1 = [featurelist.feat.theta1];
    fAng2 = [featurelist.feat.theta2];
    fAng3 = [featurelist.feat.theta3];

    fPhi1 = [featurelist.feat.phi1];
    fPhi2 = [featurelist.feat.phi2];
    fPhi3 = [featurelist.feat.phi3];

    patsp1 = [pattern.spot1];
    patsp2 = [pattern.spot2];
    patsp3 = [pattern.spot3];

    %Array of spots
    ns = 1:N;

    %
    for i = 1:S

        %New variables for ease in coding
        starnum(1) = pattern(i).spot1;
        starnum(2) = pattern(i).spot2;
```

```
starnum(3) = pattern(i).spot3;

%Search for next 'spots' to build future triads
index    = ( ns>starnum(1) & ns>starnum(2) & ns>starnum(3) );
star4set = ns( index ~= 0 );

%
for j = 1:length(star4set)

    %New 4th spot chosen
    starnum(4) = star4set(j);

    %Search for new triads using all 4 spots using indexing
    i1 = (patsp1 == starnum(1) |...
          patsp1 == starnum(2) |...
          patsp1 == starnum(3) |...
          patsp1 == starnum(4));
    i2 = (patsp2 == starnum(1) |...
          patsp2 == starnum(2) |...
          patsp2 == starnum(3) |...
          patsp2 == starnum(4));
    i3 = (patsp3 == starnum(1) |...
          patsp3 == starnum(2) |...
          patsp3 == starnum(3) |...
          patsp3 == starnum(4));

    %Create new Image Pyramid
    Pyramid  = pattern(i1&i2&i3);

    %Initialize Featurelist Pyramid
    FPyramid = cell(4,1);

    %Add tolerances and search Featurelist
    for k = 1:4

        %Search tolerance added to image
        H1 = Pyramid(k).theta1+ecat; L1 = Pyramid(k).theta1-ecat;
        H2 = Pyramid(k).theta2+ecat; L2 = Pyramid(k).theta2-ecat;
        H3 = Pyramid(k).theta3+ecat; L3 = Pyramid(k).theta3-ecat;

        H4 = Pyramid(k).phi1+ecat;   L4 = Pyramid(k).phi1-ecat;
        H5 = Pyramid(k).phi2+ecat;   L5 = Pyramid(k).phi2-ecat;
        H6 = Pyramid(k).phi3+ecat;   L6 = Pyramid(k).phi3-ecat;

        %Indexing of tolerances
        ind1  = fAng1 <= H1; ind2  = fAng1 >= L1;
        ind3  = fAng2 <= H2; ind4  = fAng2 >= L2;
        ind5  = fAng3 <= H3; ind6  = fAng3 >= L3;
        ind7  = fPhi1 <= H4; ind8  = fPhi1 >= L4;
        ind9  = fPhi2 <= H5; ind10 = fPhi2 >= L5;
        ind11 = fPhi3 <= H6; ind12 = fPhi3 >= L6;

        %Location in Featurelist for match
        Findex = (ind1 & ind2 & ind3 & ind4  & ind5  & ind6 &...
                  ind7 & ind8 & ind9 & ind10 & ind11 & ind12);
```

```
    %New Featurelist Pyramid
    FPyramid(k) = {featurelist.feat(Findex)};

end

%Check if F.Pyramid is empty
L1 = length(FPyramid{1}); L2 = length(FPyramid{2});
L3 = length(FPyramid{3}); L4 = length(FPyramid{4});

%Verify if F.Pyramid is valid for use, else use new 4th spot
if L1 == 0 || L2 == 0 || L3 == 0 || L4 == 0

else

    list = zeros(12,2);
    n    = 0;

    %First Identification Process
    for k = 1:4

        h1  = [FPyramid{k}.HipID1];   %HIP's found
        u   = unique(h1);             %Unique values of HIP
        lu  = length(u);              %Number of unique values
        tag = zeros(lu,1);            %Pre-allocated matrix for loop speed

        for m = 1:lu
            index1 = h1 == u(m);     %index: all locations of value u(m) in HIP
            tag(m) = sum(index1);    %TAG: number of times value u(m) is found in HIP
        end
        [~,index1] = max(tag);       %Location in TAG for max similar entries of u

        h2  = [FPyramid{k}.HipID2];   %HIP's found
        u   = unique(h2);             %Unique values of HIP
        lu  = length(u);              %Number of unique values
        tag = zeros(lu,1);            %Pre-allocated matrix for loop speed

        for m = 1:lu
            index2 = h2 == u(m);     %index: all locations of value u(m) in HIP
            tag(m) = sum(index2);    %TAG: number of times value u(m) is found in HIP
        end
        [~,index2] = max(tag);       %Location in TAG for max similar entries of u

        h3  = [FPyramid{k}.HipID3];   %HIP's found
        u   = unique(h3);             %Unique values of HIP
        lu  = length(u);              %Number of unique values
        tag = zeros(lu,1);            %Pre-allocated matrix for loop speed

        for m = 1:lu
            index3 = h3 == u(m);     %index: all locations of value u(m) in HIP
            tag(m) = sum(index3);    %TAG: number of times value u(m) is found in HIP
        end
        [~,index3] = max(tag);       %Location in TAG for max similar entries of u

        %Update LIST entry and counter
        n = n(end)+1:n(end)+3;
```

```
                list(n,:) = [Pyramid(k).spot1 h1(index1)
                             Pyramid(k).spot2 h2(index2)
                             Pyramid(k).spot3 h3(index3)];

            end

            %Final Identification and Output
            for k = 1:N

                %Find all HIP's for spot 'k'
                H   = list( list(:,1) == k ,2);

                if ~isempty(H)

                    u   = unique(H);            %Unique values of HIP
                    lu  = length(u);            %Number of unique values
                    tag = zeros(lu,1);          %Pre-allocated matrix for loop speed

                    for m = 1:lu
                        index = H == u(m);      %index: all locations of value u(m) in HIP
                        tag(m) = sum(index);    %TAG: number of times value u(m) is found in HIP
                    end

                    [ntags,index] = max(tag);   %Location in TAG for max similar entries of u

                    %Input results to Structured output
                    starID(k).votes = ntags;
                    starID(k).spot  = k;
                    starID(k).HipID = u(index);
                    starID(k).XYZ   = spotlist(k).XYZ;

                    %[Variable for use in DAVID FOWLER codes]
                    starIDMod = starID;

                else
                end
            end

            newPyramid;

            %Identification has completed and results recorded,
            %terminate further need to identify image
            return
        end
    end
end

    function newPyramid

        newspot = (~ismember(ns,starnum));
        newspot = ns(newspot~=0);

%           h1 = starID(starnum(1)).HipID; h2 = starID(starnum(2)).HipID;
%           h3 = starID(starnum(3)).HipID; h4 = starID(starnum(4)).HipID;
```

```
for i = 1:length(newspot)

    I1 = (patsp1 == starnum(1) |...
        patsp1 == starnum(2) |...
        patsp1 == starnum(3) |...
        patsp1 == starnum(4) |...
        patsp1 == newspot(i));
    I2 = (patsp2 == starnum(1) |...
        patsp2 == starnum(2) |...
        patsp2 == starnum(3) |...
        patsp2 == starnum(4) |...
        patsp2 == newspot(i));
    I3 = (patsp3 == starnum(1) |...
        patsp3 == starnum(2) |...
        patsp3 == starnum(3) |...
        patsp3 == starnum(4) |...
        patsp3 == newspot(i));

    newPyr = pattern(I1 & I2 & I3);

    S1 = [newPyr.spot1] == newspot(i);
    S2 = [newPyr.spot2] == newspot(i);
    S3 = [newPyr.spot3] == newspot(i);

    newPyr = newPyr(S1 | S2 | S3);

    for j = 1:6

        H1 = newPyr(j).theta1+ecat; L1 = newPyr(j).theta1-ecat;
        H2 = newPyr(j).theta2+ecat; L2 = newPyr(j).theta2-ecat;
        H3 = newPyr(j).theta3+ecat; L3 = newPyr(j).theta3-ecat;

        H4 = newPyr(j).phi1+ecat;   L4 = newPyr(j).phi1-ecat;
        H5 = newPyr(j).phi2+ecat;   L5 = newPyr(j).phi2-ecat;
        H6 = newPyr(j).phi3+ecat;   L6 = newPyr(j).phi3-ecat;

        %Indexing of tolerances
        ind1  = fAng1 <= H1; ind2  = fAng1 >= L1;
        ind3  = fAng2 <= H2; ind4  = fAng2 >= L2;
        ind5  = fAng3 <= H3; ind6  = fAng3 >= L3;
        ind7  = fPhi1 <= H4; ind8  = fPhi1 >= L4;
        ind9  = fPhi2 <= H5; ind10 = fPhi2 >= L5;
        ind11 = fPhi3 <= H6; ind12 = fPhi3 >= L6;

        %Location in Featurelist for match
        Findex = (ind1 & ind2 & ind3 & ind4  & ind5  & ind6 &...
            ind7 & ind8 & ind9 & ind10 & ind11 & ind12);

        %New Featurelist Pyramid
        FPyramid(j) = {featurelist.feat(Findex)};

    end

    %Check if F.Pyramid is empty
    L1 = length(FPyramid{1}); L2 = length(FPyramid{2});
```

```
L3 = length(FPyramid{3}); L4 = length(FPyramid{4});
L5 = length(FPyramid{5}); L6 = length(FPyramid{6});
Ltable = [L1 L2 L3 L4 L5 L6];

%Verify if F.Pyramid is valid for use, else use new 4th spot
if sum(Ltable) == 0

else

    h = 0;
    for j = 1:6
        for k = 1:Ltable(j)
            if newPyr(j).spot1 == newspot(i)

                s2 = newPyr(j).spot2;
                s3 = newPyr(j).spot3;

                is1 = FPyramid{j}(k).HipID2 == starID(s2).HipID;
                is2 = FPyramid{j}(k).HipID3 == starID(s3).HipID;

                if sum([is1 is2]) == 2
                    h(k) = FPyramid{j}(k).HipID1;
                end

            elseif newPyr(j).spot2 == newspot(i)

                s1 = newPyr(j).spot1;
                s3 = newPyr(j).spot3;

                is1 = FPyramid{j}(k).HipID1 == starID(s1).HipID;
                is2 = FPyramid{j}(k).HipID3 == starID(s3).HipID;

                if sum([is1 is2]) == 2
                    h(k) = FPyramid{j}(k).HipID2;
                end

            elseif newPyr(j).spot3 == newspot(i)

                s1 = newPyr(j).spot1;
                s2 = newPyr(j).spot2;

                is1 = FPyramid{j}(k).HipID1 == starID(s1).HipID;
                is2 = FPyramid{j}(k).HipID2 == starID(s2).HipID;

                if sum([is1 is2]) == 2
                    h(k) = FPyramid{j}(k).HipID3;
                end

            end
        end
    end

    u = unique(h);
    lu = length(u);
    tag = zeros(lu,1);
```

```
            for k = 1:lu
                indexnew = h == u(k);     %index: all locations of value u(m) in HIP
                tag(k) = sum(indexnew);    %TAG: number of times value u(m) is found in HIP
            end
            [ntags,index] = max(tag);       %Location in TAG for max similar entries of u

            if u(index) == 0
                ntags = 0;
            end

            %Input results to Structured output
            starID(newspot(i)).votes = ntags;
            starID(newspot(i)).spot  = newspot(i);
            starID(newspot(i)).HipID = u(index);
            starID(newspot(i)).XYZ   = spotlist([spotlist.spot]==newspot(i)).XYZ;

            %[Variable for use in DAVID FOWLER codes]
            starIDMod = starID;
        end
    end
end
```

End of Program.

end

H. Pyramid with Voting Method

```
function [starID,starIDMod] = getPyramidVote_ID(catalog,featurelist,spotlist,ecat)

%Mortari's Pyramid Algorithm. Creates a list of patterns described by 6
%features each using 3 stars. These patterns are checked against a feature
%list, one pattern at a time, using a 4th star as a verification tool. If
%all 4 spots match to stars in the feature list, then the spots are marked,
%their HIP# recorded, and the output is a table of all spots in the image,
%their location, and only the 4 spots that were recognized will have a HIP
%ID.

%Length of spotlist
N = length(spotlist);

%Initialize structure
starID(N,1) = struct('votes',[],'spot',[],'HipID',[],'XYZ',[]);

%Pad with zeros and with known info
[starID.votes] = deal(0);
[starID.spot]  = spotlist.spot;
[starID.HipID] = deal(0);
[starID.XYZ]   = spotlist.XYZ;

%Preallocated variable
starIDMod = starID;

Algorithm Model

if N < 4 % Early Failure Detection, requires min. 4 spots in image to proc.
```

```
else %Process Image and Analyze

Pattern Creation

    %Pattern size: N! / ( 6 * [N-3]! )
    S = N*(N-1)*(N-2)/6;

    %Initialize structured array
    pattern(S,1) = struct('spot1',[],'spot2',[],'spot3',[],...
        'theta1',[],'theta2',[],'theta3',[],...
        'phi1',[],'phi2',[],'phi3',[]);
    %Pattern Counter
    L = 0;

    %Create Patterns
    for i = 1:N-2

        %Find 1st spot and vector
        spot1 = spotlist(i).spot;
        A     = spotlist(i).XYZ;

        for j = i+1:N-1

            %Find 2nd spot and vector
            spot2  = spotlist(j).spot;
            B      = spotlist(j).XYZ;

            %Find 1st Feature
            theta1 = acos(dot(A,B));

            for k = j+1:N

                %Find 3rd spot and vector
                spot3  = spotlist(k).spot;
                C      = spotlist(k).XYZ;

                %Find 2nd and 3rd Features
                theta2 = acos(dot(A,C)/(norm(A)*norm(C)));
                theta3 = acos(dot(B,C)/(norm(B)*norm(C)));

                %Calculate Interior Angles
                V12 = B-A;
                V13 = C-A;
                V23 = C-B;
                v12 = sqrt(V12(1)^2+V12(2)^2+V12(3)^2);
                v13 = sqrt(V13(1)^2+V13(2)^2+V13(3)^2);
                v23 = sqrt(V23(1)^2+V23(2)^2+V23(3)^2);

                %Interior angles: Features 4->6
                phi1 = acos(dot(V12,V13)/(v12*v13));
                phi2 = acos(dot(V12,V23)/(v12*v23));
                phi3 = acos(dot(V13,V23)/(v13*v23));

                %Sort Features based on smallest theta angle
                if theta1 < theta2 && theta1 < theta3
```

```
                if theta2 < theta3
                    S1 = spot1;  S2 = spot2;  S3 = spot3;
                    T1 = theta1; T2 = theta2; T3 = theta3;
                    P1 = phi1;   P2 = phi2;   P3 = phi3;
                else
                    S1 = spot2;  S2 = spot1;  S3 = spot3;
                    T1 = theta1; T2 = theta3; T3 = theta2;
                    P1 = phi1;   P2 = phi3;   P3 = phi2;
                end
            elseif theta2 < theta1 && theta2 < theta3
                if theta1 < theta3
                    S1 = spot1;  S2 = spot3;  S3 = spot2;
                    T1 = theta2; T2 = theta1; T3 = theta3;
                    P1 = phi2;   P2 = phi1;   P3 = phi3;
                else

                    S1 = spot3;  S2 = spot1;  S3 = spot2;
                    T1 = theta2; T2 = theta3; T3 = theta1;
                    P1 = phi2;   P2 = phi3;   P3 = phi1;
                end
            elseif theta3 < theta1 && theta3 < theta2
                if theta1 < theta2
                    S1 = spot2;  S2 = spot3;  S3 = spot1;
                    T1 = theta3; T2 = theta1; T3 = theta2;
                    P1 = phi3;   P2 = phi1;   P3 = phi2;
                else
                    S1 = spot3;  S2 = spot2;  S3 = spot1;
                    T1 = theta3; T2 = theta2; T3 = theta1;
                    P1 = phi3;   P2 = phi2;   P3 = phi1;
                end
            end

            %Update Pattern Counter
            L = L + 1;

            %Input Pattern
            pattern(L).spot1 = S1;
            pattern(L).spot2 = S2;
            pattern(L).spot3 = S3;

            pattern(L).theta1 = T1;
            pattern(L).theta2 = T2;
            pattern(L).theta3 = T3;

            pattern(L).phi1 = P1;
            pattern(L).phi2 = P2;
            pattern(L).phi3 = P3;

        end
    end
end
```

Pattern Identification

```
%Pre-allocate for increased index search speed
fAng1 = [featurelist.feat.theta1];
```

```
fAng2 = [featurelist.feat.theta2];
fAng3 = [featurelist.feat.theta3];

fPhi1 = [featurelist.feat.phi1];
fPhi2 = [featurelist.feat.phi2];
fPhi3 = [featurelist.feat.phi3];

patsp1 = [pattern.spot1];
patsp2 = [pattern.spot2];
patsp3 = [pattern.spot3];

%Array of spots
ns = 1:N;

%
for i = 1:S

    %New variables for ease in coding
    starnum(1) = pattern(i).spot1;
    starnum(2) = pattern(i).spot2;
    starnum(3) = pattern(i).spot3;

    %Search for next 'spots' to build future triads
    index    = ( ns>starnum(1) & ns>starnum(2) & ns>starnum(3) );
    star4set = ns( index ~= 0 );

    %
    for j = 1:length(star4set)

        %New 4th spot chosen
        starnum(4) = star4set(j);

        %Search for new triads using all 4 spots using indexing
        i1 = (patsp1 == starnum(1) |...
            patsp1 == starnum(2) |...
            patsp1 == starnum(3) |...
            patsp1 == starnum(4));
        i2 = (patsp2 == starnum(1) |...
            patsp2 == starnum(2) |...
            patsp2 == starnum(3) |...
            patsp2 == starnum(4));
        i3 = (patsp3 == starnum(1) |...
            patsp3 == starnum(2) |...
            patsp3 == starnum(3) |...
            patsp3 == starnum(4));

        %Create new Image Pyramid
        Pyramid  = pattern(i1&i2&i3);

        %Initialize Featurelist Pyramid
        FPyramid = cell(4,1);

        %Add tolerances and search Featurelist
        for k = 1:4

            %Search tolerance added to image
```

```
    H1 = Pyramid(k).theta1+ecat; L1 = Pyramid(k).theta1-ecat;
    H2 = Pyramid(k).theta2+ecat; L2 = Pyramid(k).theta2-ecat;
    H3 = Pyramid(k).theta3+ecat; L3 = Pyramid(k).theta3-ecat;

    H4 = Pyramid(k).phi1+ecat;   L4 = Pyramid(k).phi1-ecat;
    H5 = Pyramid(k).phi2+ecat;   L5 = Pyramid(k).phi2-ecat;
    H6 = Pyramid(k).phi3+ecat;   L6 = Pyramid(k).phi3-ecat;

    %Indexing of tolerances
    ind1  = fAng1 <= H1; ind2  = fAng1 >= L1;
    ind3  = fAng2 <= H2; ind4  = fAng2 >= L2;
    ind5  = fAng3 <= H3; ind6  = fAng3 >= L3;
    ind7  = fPhi1 <= H4; ind8  = fPhi1 >= L4;
    ind9  = fPhi2 <= H5; ind10 = fPhi2 >= L5;
    ind11 = fPhi3 <= H6; ind12 = fPhi3 >= L6;

    %Location in Featurelist for match
    Findex = (ind1 & ind2 & ind3 & ind4  & ind5  & ind6 &...
        ind7 & ind8 & ind9 & ind10 & ind11 & ind12);

    %New Featurelist Pyramid
    FPyramid(k) = {featurelist.feat(Findex)};

end

%Check if F.Pyramid is empty
L1 = length(FPyramid{1}); L2 = length(FPyramid{2});
L3 = length(FPyramid{3}); L4 = length(FPyramid{4});

%Verify if F.Pyramid is valid for use, else use new 4th spot
if L1 == 0 || L2 == 0 || L3 == 0 || L4 == 0

else

    list = zeros(12,2);
    n    = 0;

    %First Identification Process
    for k = 1:4

        h1  = [FPyramid{k}.HipID1];   %HIP's found
        u   = unique(h1);             %Unique values of HIP
        lu  = length(u);              %Number of unique values
        tag = zeros(lu,1);            %Pre-allocated matrix for loop speed

        for m = 1:lu
            index1 = h1 == u(m);     %index: all locations of value u(m) in HIP
            tag(m) = sum(index1);    %TAG: number of times value u(m) is found in HIP
        end
        [~,index1] = max(tag);       %Location in TAG for max similar entries of u

        h2  = [FPyramid{k}.HipID2];   %HIP's found
        u   = unique(h2);             %Unique values of HIP
        lu  = length(u);              %Number of unique values
        tag = zeros(lu,1);            %Pre-allocated matrix for loop speed
```

```
    for m = 1:lu
        index2 = h2 == u(m);     %index: all locations of value u(m) in HIP
        tag(m) = sum(index2);    %TAG: number of times value u(m) is found in HIP
    end
    [~,index2] = max(tag);       %Location in TAG for max similar entries of u

    h3  = [FPyramid{k}.HipID3];   %HIP's found
    u   = unique(h3);             %Unique values of HIP
    lu  = length(u);              %Number of unique values
    tag = zeros(lu,1);            %Pre-allocated matrix for loop speed

    for m = 1:lu
        index3 = h3 == u(m);     %index: all locations of value u(m) in HIP
        tag(m) = sum(index3);    %TAG: number of times value u(m) is found in HIP
    end
    [~,index3] = max(tag);       %Location in TAG for max similar entries of u

    %Update LIST entry and counter
    n = n(end)+1:n(end)+3;

    list(n,:) = [Pyramid(k).spot1 h1(index1)
        Pyramid(k).spot2 h2(index2)
        Pyramid(k).spot3 h3(index3)];

end

%Final Identification and Output
for k = 1:N

    %Find all HIP's for spot 'k'
    H   = list( list(:,1) == k ,2);

    if ~isempty(H)

        u   = unique(H);             %Unique values of HIP
        lu  = length(u);             %Number of unique values
        tag = zeros(lu,1);           %Pre-allocated matrix for loop speed

        for m = 1:lu
            index = H == u(m);       %index: all locations of value u(m) in HIP
            tag(m) = sum(index);     %TAG: number of times value u(m) is found in HIP
        end

        [ntags,index] = max(tag);    %Location in TAG for max similar entries of u

        %Input results to Structured output
        starID(k).votes = ntags;
        starID(k).spot  = k;
        starID(k).HipID = u(index);
        starID(k).XYZ   = spotlist(k).XYZ;

        %[Variable for use in DAVID FOWLER codes]
        starIDMod = starID;

    else
    end
```

```
                end

                SVT = starID;
                Validation

                %Identification has completed and results recorded,
                %terminate further need to identify image
                return

            end
        end
    end

end

    function Validation

Validation Procedure

        starID(N,1) = struct('votes',[],'spot',[],'HipID',[],'XYZ',[]);

        catHip = [catalog.cat.HipID];
        pos = 2;
        neg = 1;

        multID = unique([SVT.HipID]);
        LM = length(multID);

        for i = 1:LM
            index = [SVT.HipID]==multID(i);
            marks = sum(index);
            if marks > 1
                [SVT(index).HipID] = deal(0);
            end
        end

        for i = 1:N

            if SVT(i).HipID ~= 0

                index1 = catHip == SVT(i).HipID;
                XYZ1   = catalog.cat(index1).XYZ;
                xyz1   = spotlist(i).XYZ;

                if isempty(starID(i).votes)
                    starID(i).votes = 0;
                end

                for j = 1:N

                    if j ~= i

                        if SVT(j).HipID ~= 0

                            if isempty(starID(j).votes)
                                starID(j).votes = 0;
```

```
                end

                index2 = catHip == SVT(j).HipID;
                XYZ2   = catalog.cat(index2).XYZ;
                xyz2   = spotlist(j).XYZ;

                angle = acos(dot(XYZ1,XYZ2));
                theta = acos(dot(xyz1,xyz2));

                if ~isreal(theta) || theta <= 0
                    Lower = 0;
                    Upper = ecat;
                else
                    Upper = theta+ecat;
                    Lower = theta-ecat;
                end

                if Upper >= angle && angle >= Lower

                    starID(i).votes = starID(i).votes+pos;
                    starID(j).votes = starID(j).votes+pos;

                else

                    %starID(i).votes = starID(i).votes-neg;
                    starID(j).votes = starID(j).votes-neg;

                end

            else

                %starID(i).votes = starID(i).votes-neg;
                starID(j).votes = starID(j).votes-neg;

            end
        end
    end

elseif SVT(i).HipID == 0

    if isempty([starID(i).votes])
        starID(i).votes = 0;
    end

    starID(i).votes = starID(i).votes-neg;
    starID(j).votes = starID(j).votes-neg;

end

starID(i).spot = i;
starID(i).HipID = SVT(i).HipID;
starID(i).XYZ = spotlist(i).XYZ;

end

starIDMod = starID;
```

```
        index = [starID.votes] <= 0;
        [starIDMod(index).HipID] = deal(0);

    end

End of Program.

end
```

I. Voting Algorithm

```
function [starID,starIDMod] = Voting_Algorithm(catalog,featurelist,spotlist,pattern,ecat,ns)

Voting Code for 2 stars and 3 stars
Initial Pass Voting - First Stage

IVT(1000000,1) = struct('votes',[],'HipID',[],'spot',[]); %Initial Voting Table
n = length(pattern);
N = length(spotlist);
L = 0;
% dbstop Voting_Algorithm.m at 11
% Strip Angles from feature list only once outside loop
fAng1 = [featurelist.feat.theta1];

if ns == 3
    fAng2 = [featurelist.feat.theta2];
    fAng3 = [featurelist.feat.phi];
end

for i = 1:n %Run based on number of features in 'pattern'

    %featurelist being truncated based on pattern angles and catalog search
    %error tolerance
    high1 = pattern(i).theta1 + ecat; %high and low based on first angle
    low1  = pattern(i).theta1 - ecat;

    if ns == 3 %3-star only truncation

        high2 = pattern(i).theta2 + ecat; %high and low based on second angle
        low2  = pattern(i).theta2 - ecat;

        high3 = pattern(i).phi + ecat; %high and low based on interior angle
        low3  = pattern(i).phi - ecat;

        %Indices that are found to match
        ind1  = fAng1 <= high1;
        ind2  = fAng1 >= low1 ;
        ind3  = fAng2 <= high2;
        ind4  = fAng2 >= low2 ;
        ind5  = fAng3 <= high3;
        ind6  = fAng3 >= low3 ;
        index = (ind1 & ind2 & ind3 & ind4 & ind5 & ind6);

    else %2-star only truncation

        ind1  = fAng1 <= high1;
```

```
        ind2  = fAng1 >= low1 ;
        index = (ind1 & ind2);

    end

    %featurelist truncated and size recorded
    FLcheck  = featurelist.feat(index);
    numMatch = length(FLcheck); %Number of matches between pattern and featurelist
    L        = L + 1;

    if numMatch ~= 0 %Found a possible match

        nFound = L - 1 + numMatch;
        iFill  = (L:nFound);

        [IVT(iFill).votes] = deal( 1 );
        [IVT(iFill).HipID] = FLcheck(:).HipID1;
        [IVT(iFill).spot]  = deal( pattern(i).spot1 );

        if ns ~= 2

            iFill = iFill + numMatch;

            [IVT(iFill).votes] = deal( 1 );
            [IVT(iFill).HipID] = FLcheck(:).HipID2;
            [IVT(iFill).spot]  = deal( pattern(i).spot2 );

            iFill = iFill + numMatch;
            [IVT(iFill).votes] = deal( 1 );
            [IVT(iFill).HipID] = FLcheck(:).HipID3;
            [IVT(iFill).spot]  = deal( pattern(i).spot3 );

        end

        L = nFound+2*(ns-2)*numMatch;

    else %If no feature match is found

        IVT(L).votes   = 0;
        IVT(L).HipID   = 0;
        IVT(L).spot    = pattern(i).spot1;

        if ns ~= 2
            IVT(L+1).votes = 0;
            IVT(L+1).HipID = 0;
            IVT(L+1).spot  = pattern(i).spot2;
            IVT(L+2).votes = 0;
            IVT(L+2).HipID = 0;
            IVT(L+2).spot  = pattern(i).spot3;
        end

        L = L + ns-1;

    end
end
```

```
IVT = IVT(1:L);

Second Pass Voting - Second Stage

SVT(N,1) = struct('votes',[],'spot',[],'HipID',[]);
ivtSpot  = [IVT.spot];
ivtHip   = [IVT.HipID];

for j = 1:N

    %Index and cut IVT based on 'j'
    iS    = ivtSpot == j;
    iH    = ivtHip ~= 0;
    index = (iS & iH);

    hip   = ivtHip(index);

    if ~isempty(hip)

        uhip  = unique(hip);
        s     = size(uhip,2);
        votes = zeros(s,1);

        for i = 1:s

            votes(i) = sum(hip == uhip(i));

        end

        if sum(votes==max(votes)) > 1

            SVT(j).votes = 0;
            SVT(j).spot  = j;
            SVT(j).HipID = 0;

        else

            [vote,index] = max(votes);
            %Secondary Voting matrix
            SVT(j).votes = vote;
            SVT(j).spot  = j;
            SVT(j).HipID = uhip(index);

        end

    else

        SVT(j).votes = 0;
        SVT(j).spot  = j;
        SVT(j).HipID = 0;

    end
end

Validation Procedure - Third Stage
```

```
starID(N,1) = struct('votes',[],'spot',[],'HipID',[],'XYZ',[]);

catHip = [catalog.cat.HipID];
pos = 1;
neg = 1;

multID = unique([SVT.HipID]);
LM = length(multID);

for i = 1:LM
    index = [SVT.HipID]==multID(i);
    marks = sum(index);
    if marks > 1
        [SVT(index).HipID] = deal(0);
    end
end

for i = 1:N

    if SVT(i).HipID ~= 0

        index1 = catHip == SVT(i).HipID;
        XYZ1   = catalog.cat(index1).XYZ;
        xyz1   = spotlist(i).XYZ;

        if isempty(starID(i).votes)
            starID(i).votes = 0;
        end

        for j = 1:N

            if j ~= i

                if SVT(j).HipID ~= 0

                    if isempty(starID(j).votes)
                        starID(j).votes = 0;
                    end

                    index2 = catHip == SVT(j).HipID;
                    XYZ2   = catalog.cat(index2).XYZ;
                    xyz2   = spotlist(j).XYZ;

                    angle = acos(dot(XYZ1,XYZ2));
                    theta = acos(dot(xyz1,xyz2));

                    if ~isreal(theta) || theta <= 0
                        Lower = 0;
                        Upper = ecat;
                    else
                        Upper = theta+ecat;
                        Lower = theta-ecat;
                    end

                    if Upper >= angle && angle >= Lower
```

```
                    starID(i).votes = starID(i).votes+pos;
                    starID(j).votes = starID(j).votes+pos;

                else

                    %starID(i).votes = starID(i).votes-neg;
                    starID(j).votes = starID(j).votes-neg;

                end

            else

                %starID(i).votes = starID(i).votes-neg;
                starID(j).votes = starID(j).votes-neg;

            end
        end
    end

elseif SVT(i).HipID == 0

    if isempty([starID(i).votes])
        starID(i).votes = 0;
    end

    starID(i).votes = starID(i).votes-neg;
    starID(j).votes = starID(j).votes-neg;

end

starID(i).spot = i;
starID(i).HipID = SVT(i).HipID;
starID(i).XYZ = spotlist(i).XYZ;

end
starIDMod = starID;
index = [starID.votes] <= 0;
[starIDMod(index).HipID] = deal(0);

end
```

APPENDIX B

ADDITIONAL FIGURES

Additional figures that were not shown in the main body of the text are provided here to further illustrate the results obtained from simulation and experimental testing.

I. Simulations

1. Magnitude 3 Threshold

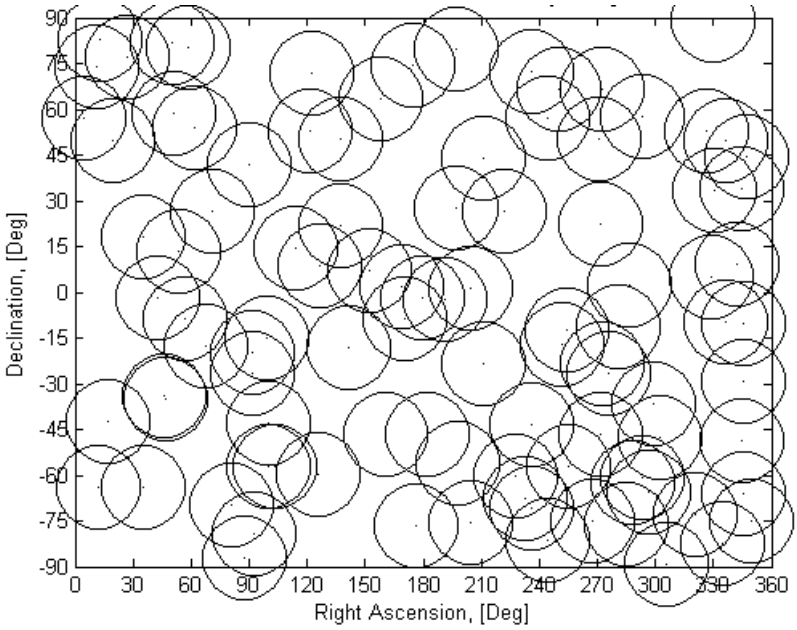

Figure B.1 Location of camera view point for 100 simulated images with approximate FOV area for magnitude 3 star fields in Miller cylindrical projection

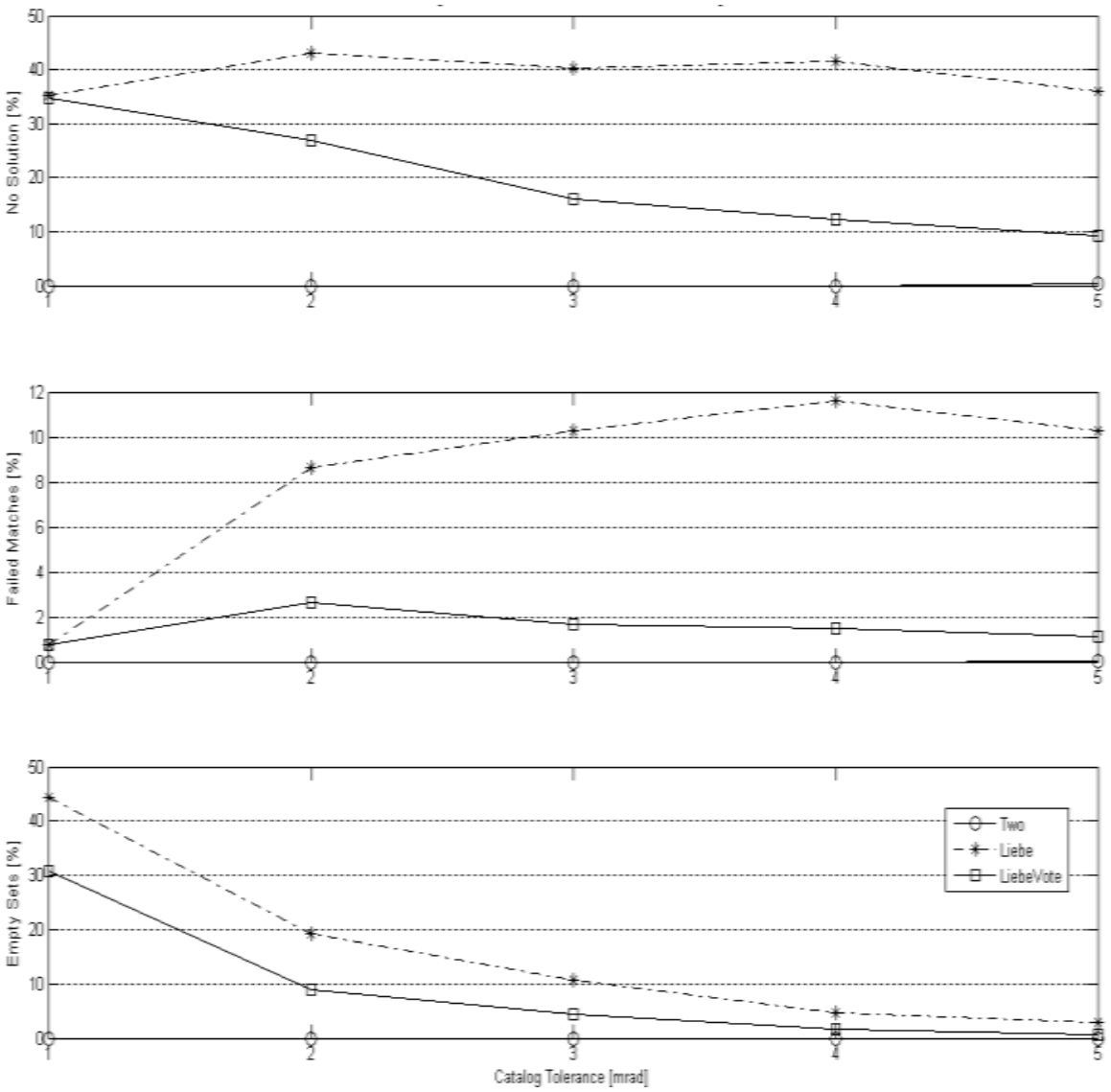

Figure B.2 Solution failures for 3 unacceptable simulated ID algorithms at magnitude 3 as a function of catalog tolerance

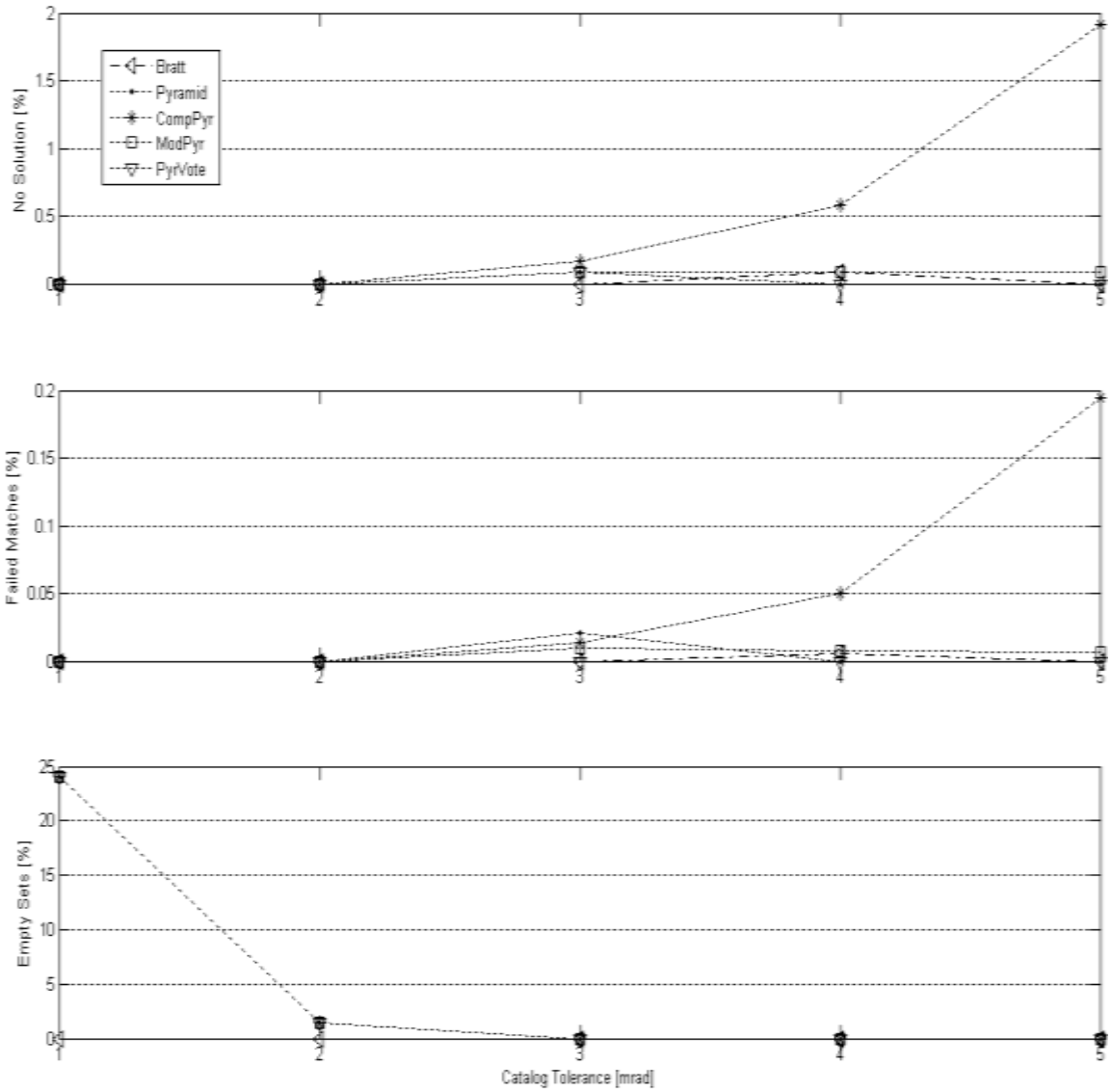

Figure B.3 Solution failures for 5 acceptable simulated algorithms at magnitude 3 as a function of catalog tolerance

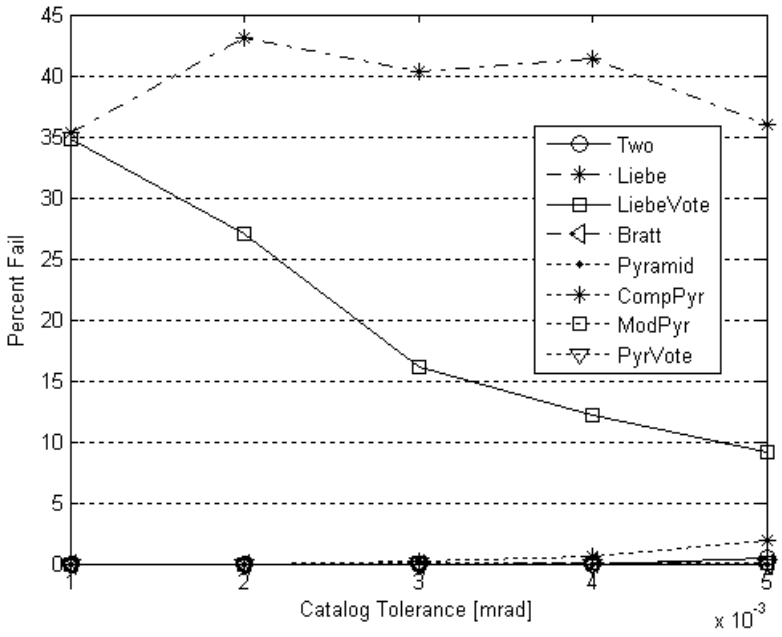

Figure B.4 Image solution failures of all simulated algorithms at magnitude 3 as a function of catalog tolerance

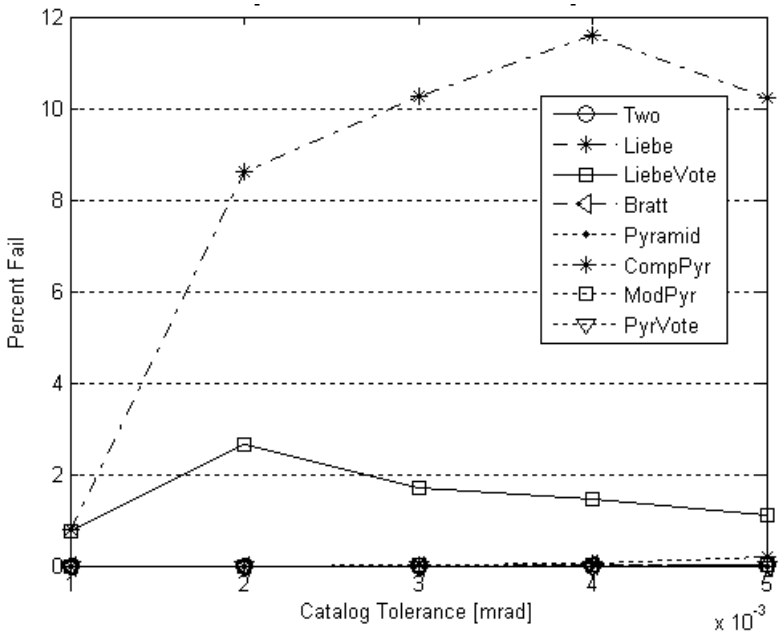

Figure B.5 Spot to star matching failures of all simulated algorithms at magnitude 3 as a function of catalog tolerance

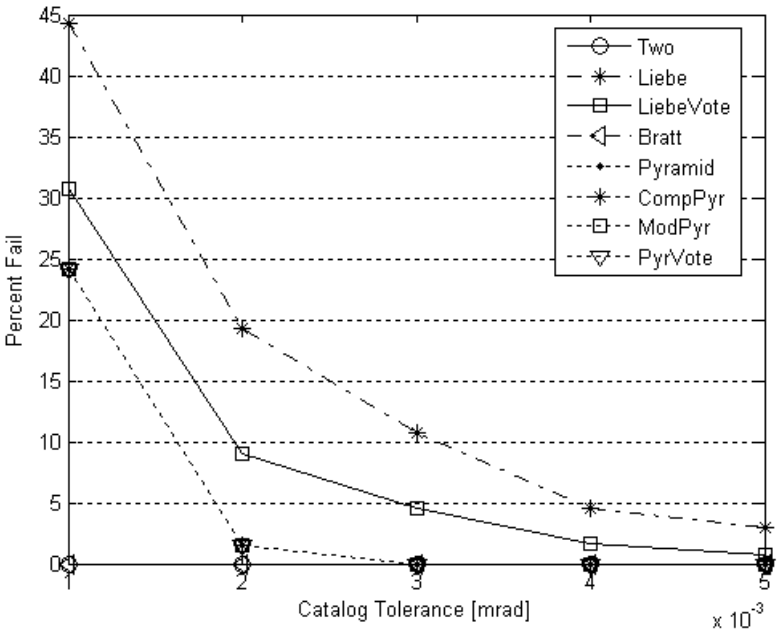

Figure B.6 Average empty sets of all simulated algorithms at magnitude 3 as a function of catalog tolerance

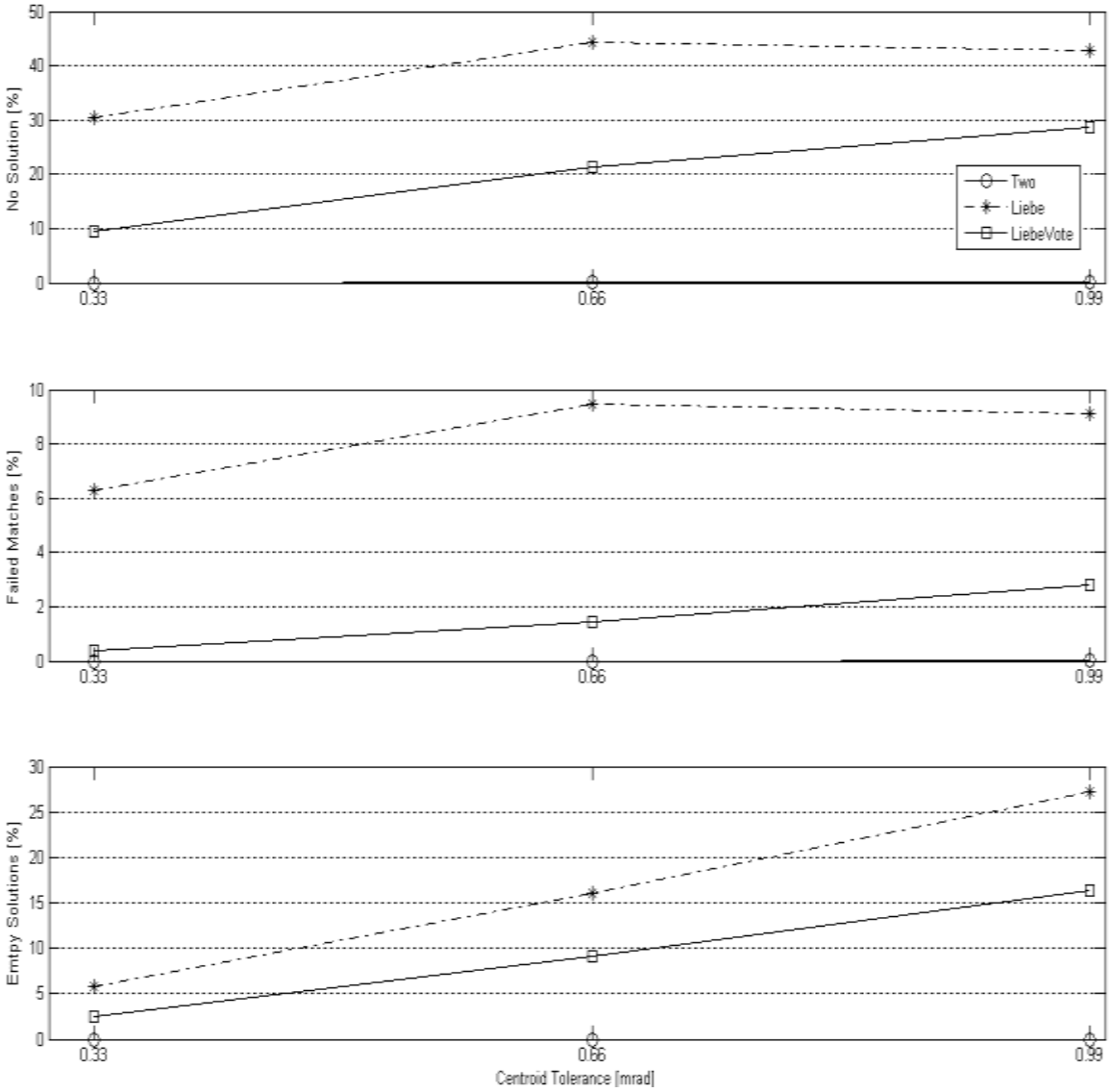

Figure B.7 Solution failures of 3 unacceptable simulated algorithms at magnitude 3 as a function of centroiding

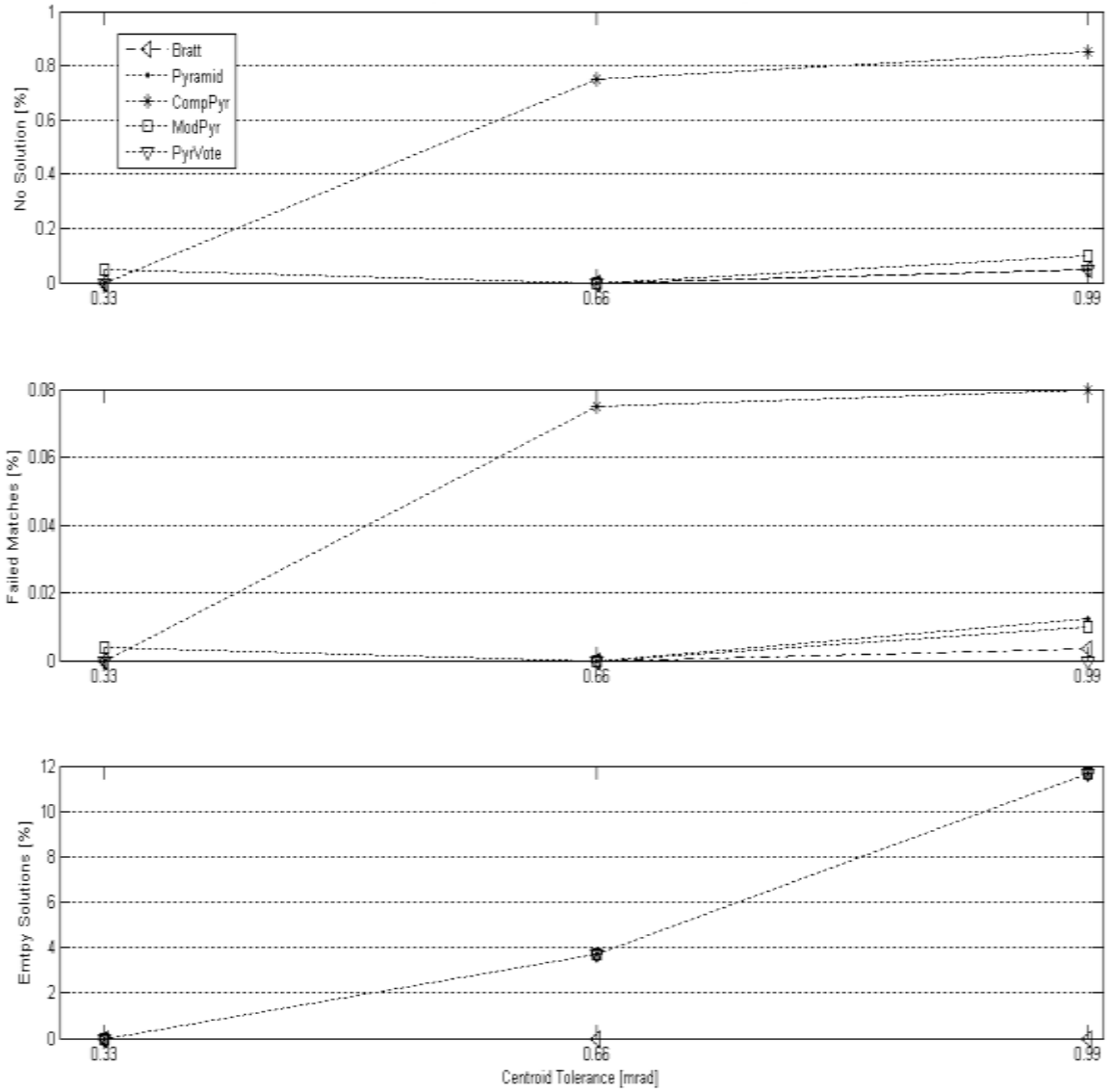

Figure B.8 Solution failures of 5 acceptable simulated algorithms at magnitude 3 as a function of centroiding

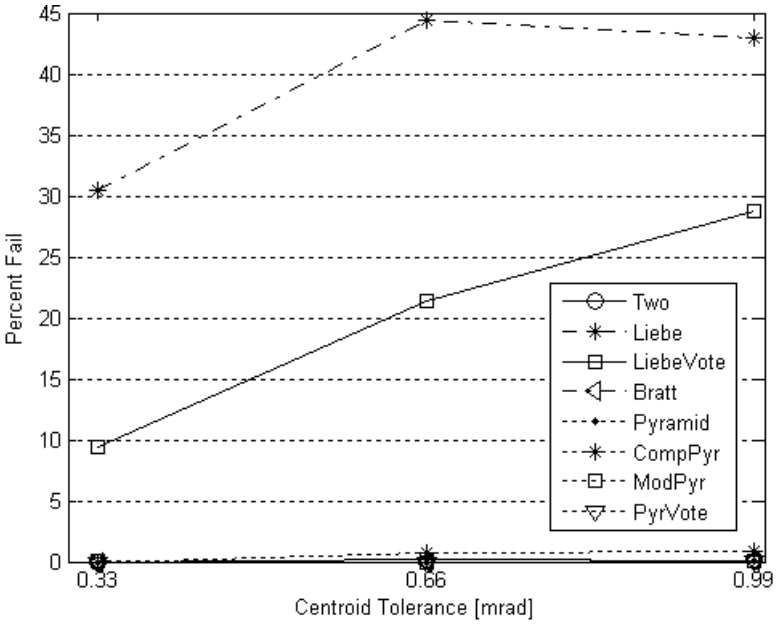

Figure B.9 Image solution failure of all simulated algorithms at magnitude 3 as a function of centroiding

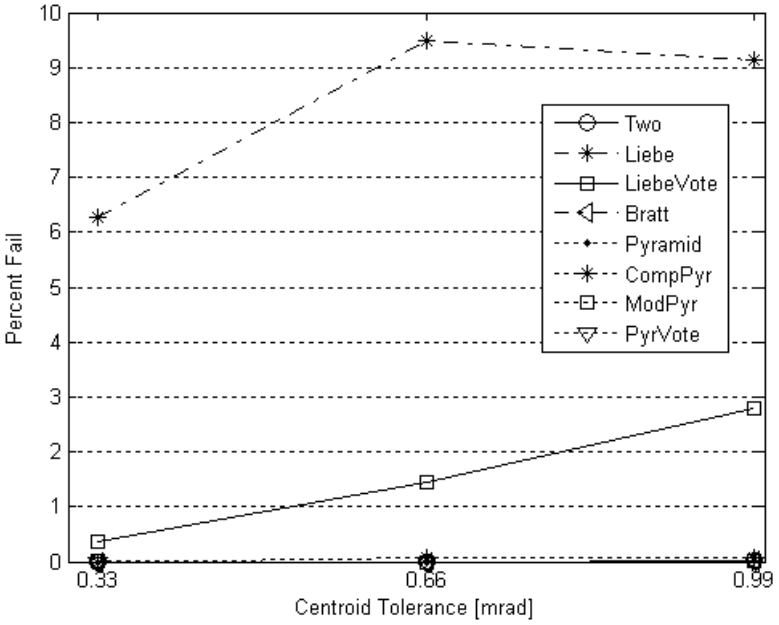

Figure B.10 Average failed matches of all simulated algorithms at magnitude 3 as a function of centroiding

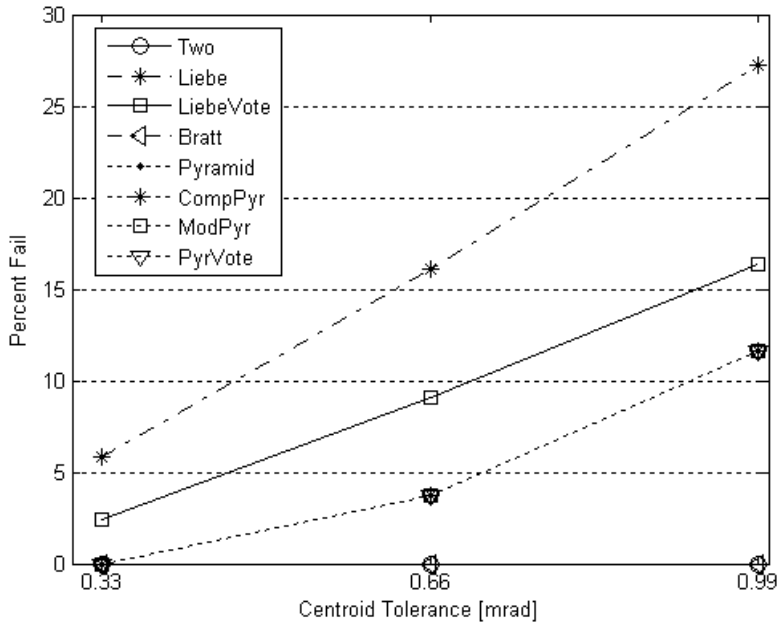

Figure B.11 Average empty sets of all simulated algorithms at magnitude 3 as a function of centroiding

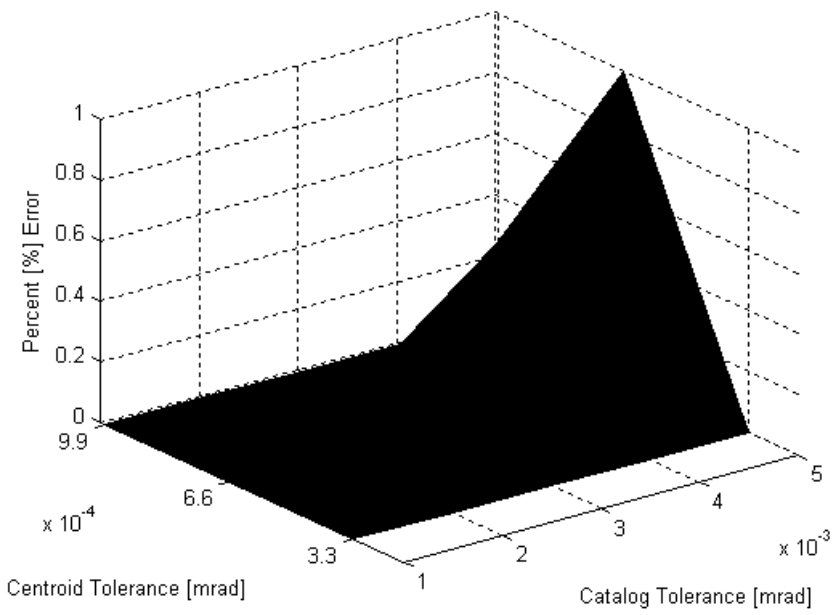

Figure B.12 3-D image solution failure of simulated Two Star method as functions of catalog tolerance and centroiding

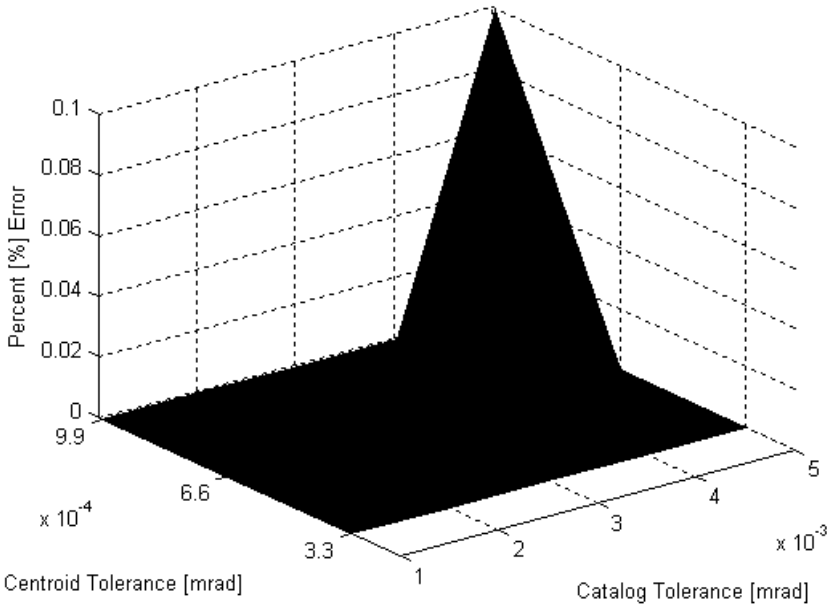

Figure B.13 3-D image match failure of simulated Two Star method as functions of catalog tolerance and centroiding

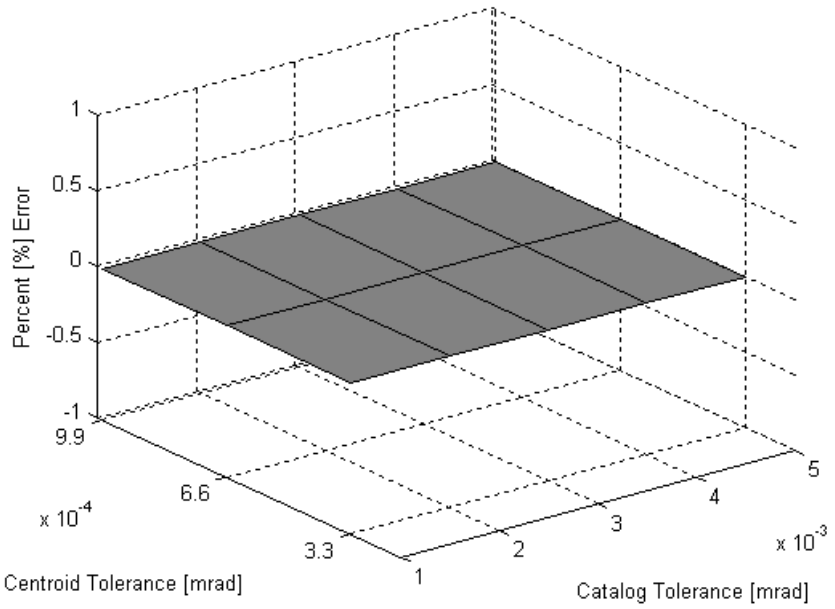

Figure B.14 3-D image empty sets of simulated Two Star method as functions of catalog tolerance and centroiding

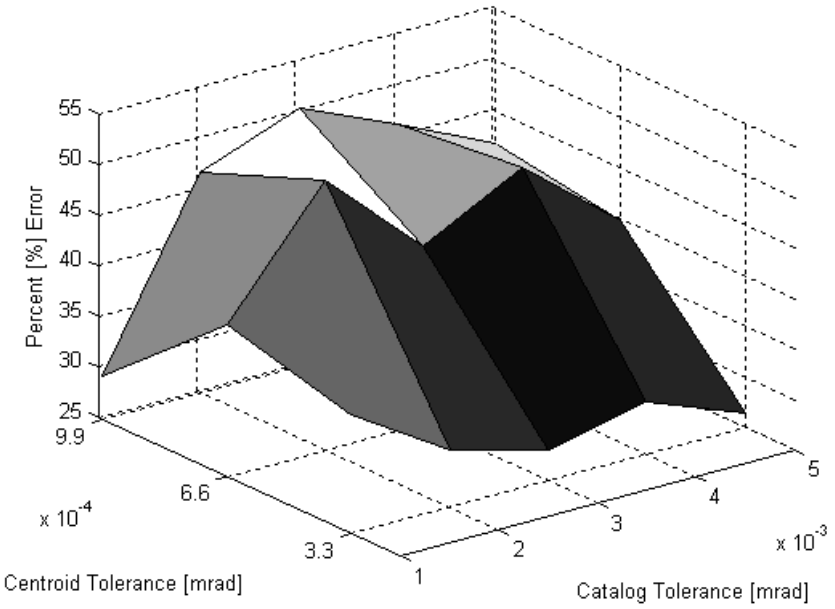

Figure B.15 3-D image solution failure of simulated Liebe method as functions of catalog tolerance and centroiding

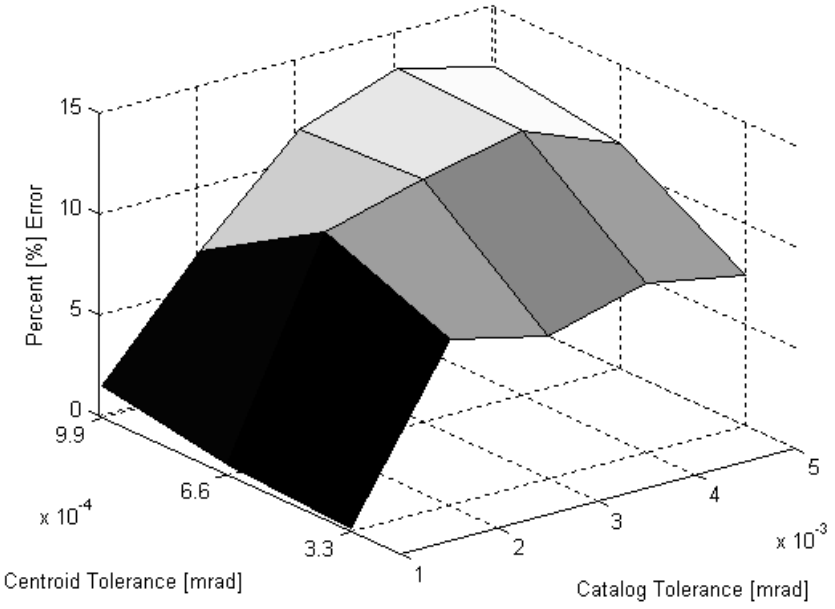

Figure B.16 3-D image match failure of simulated Liebe method as functions of catalog tolerance and centroiding

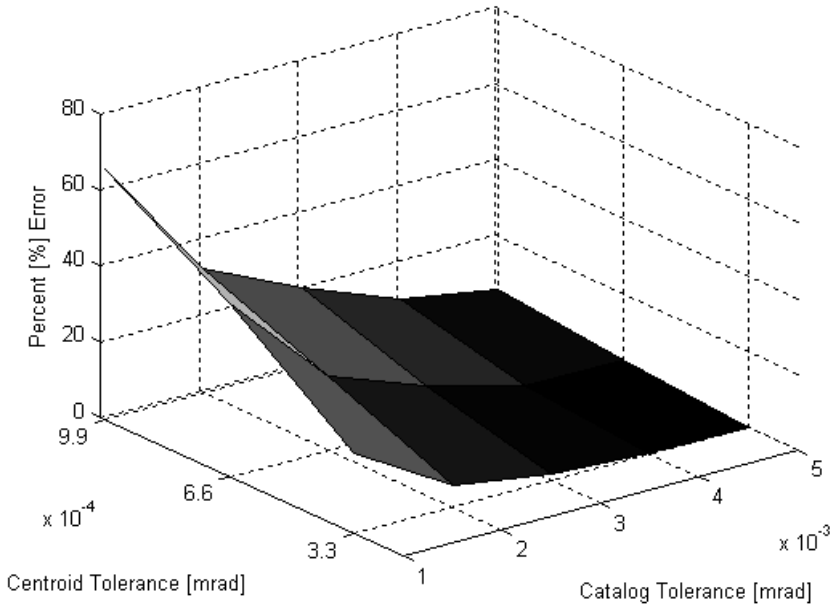

Figure B.17 3-D image empty set of simulated Liebe method as functions of catalog tolerance and centroiding

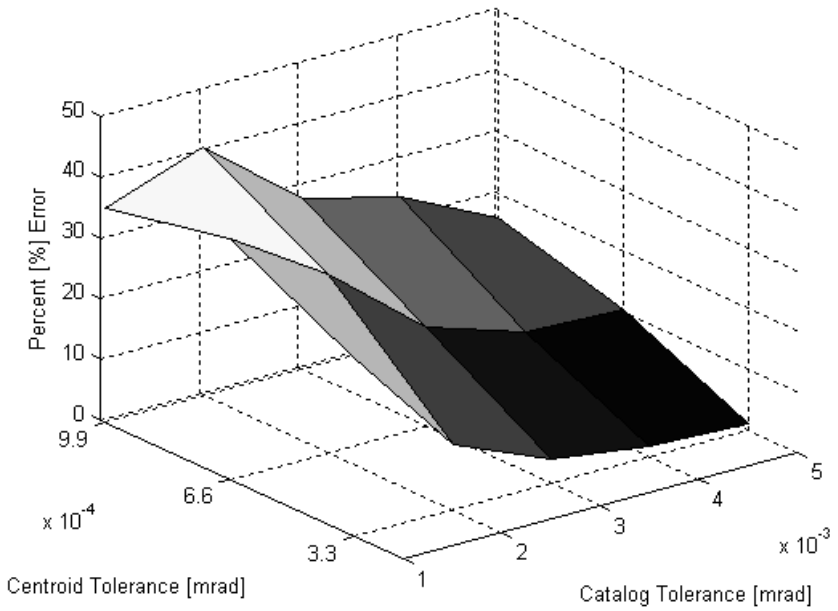

Figure B.18 3-D image solution failure of simulated Liebe with Voting method as functions of catalog tolerance and centroiding

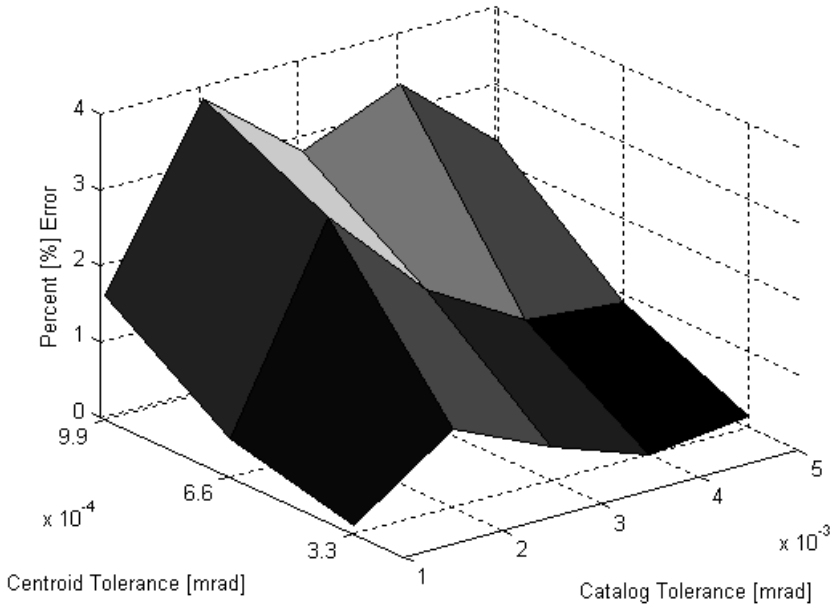

Figure B.19 3-D image match failure of simulated Liebe with Voting method as functions of catalog tolerance and centroiding

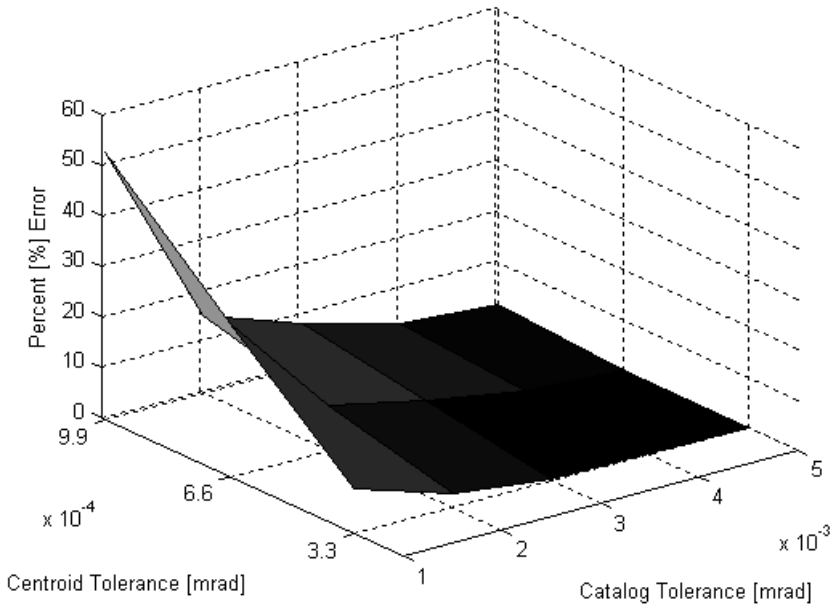

Figure B.20 3-D image empty set of simulated Liebe with Voting method as functions of catalog tolerance and centroiding

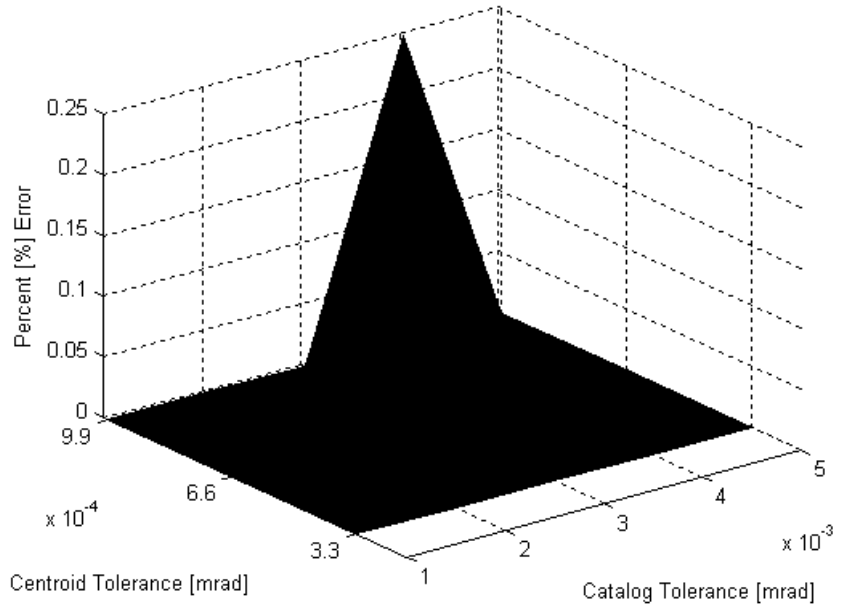

Figure B.21 3-D image solution failure of simulated Brätt method as functions of catalog tolerance and centroiding

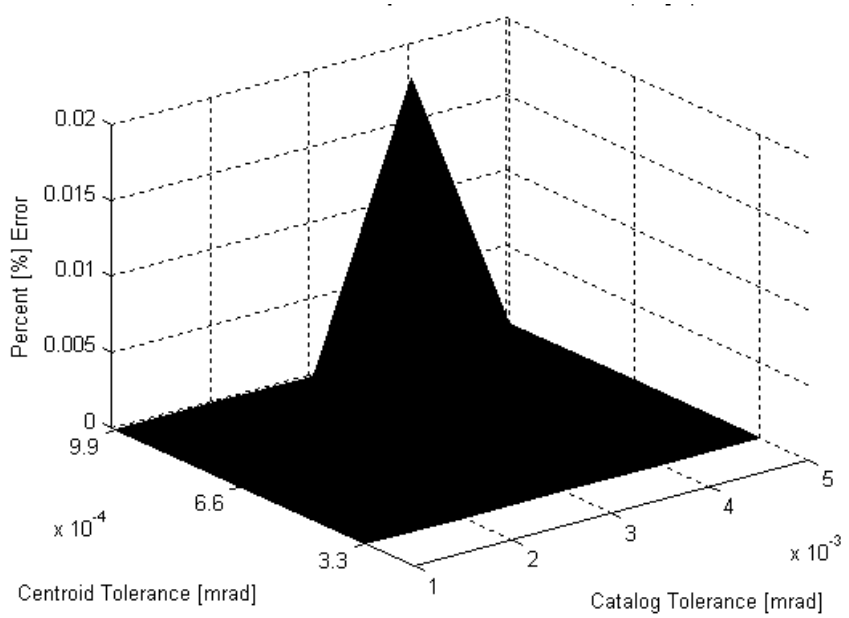

Figure B.22 3-D image match failure of simulated Brätt method as functions of catalog tolerance and centroiding

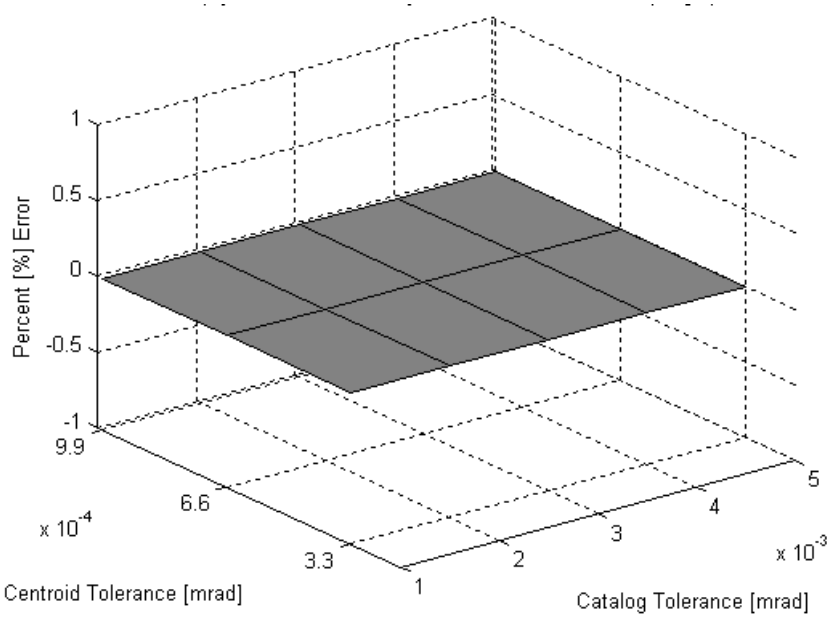

Figure B.23 3-D image empty set of simulated Brätt method as functions of catalog tolerance and centroiding

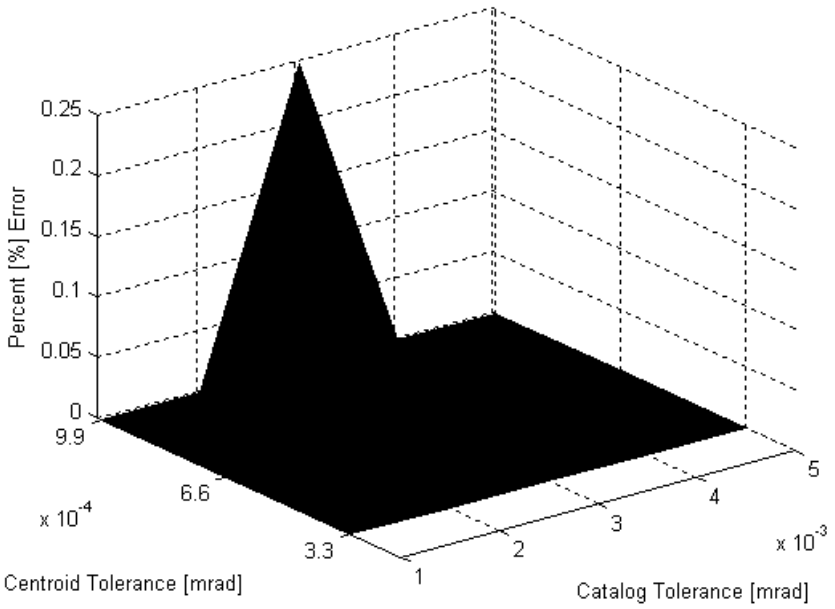

Figure B.24 3-D image solution failure of simulated Constrained Pyramid method as functions of catalog tolerance and centroiding

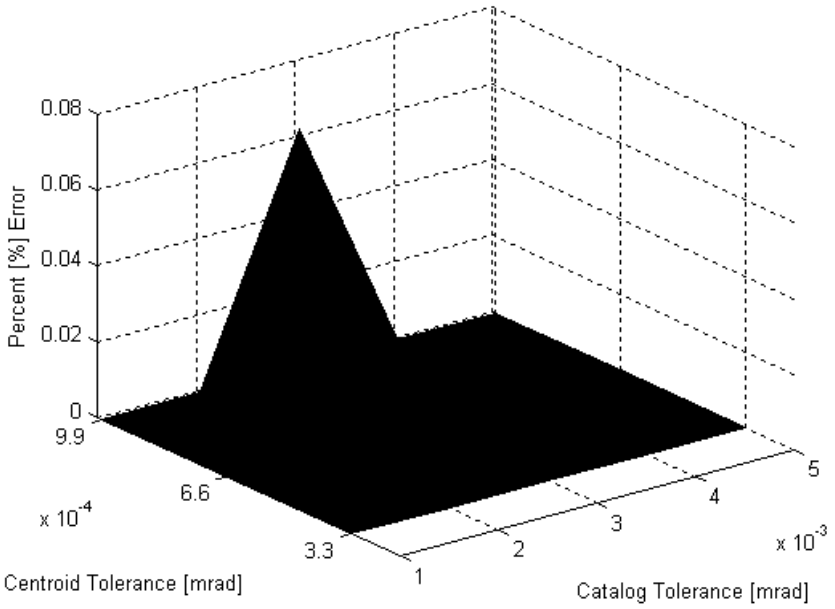

Figure B.25 3-D image match failure of simulated Constrained Pyarmid method as functions of catalog tolerance and centroiding

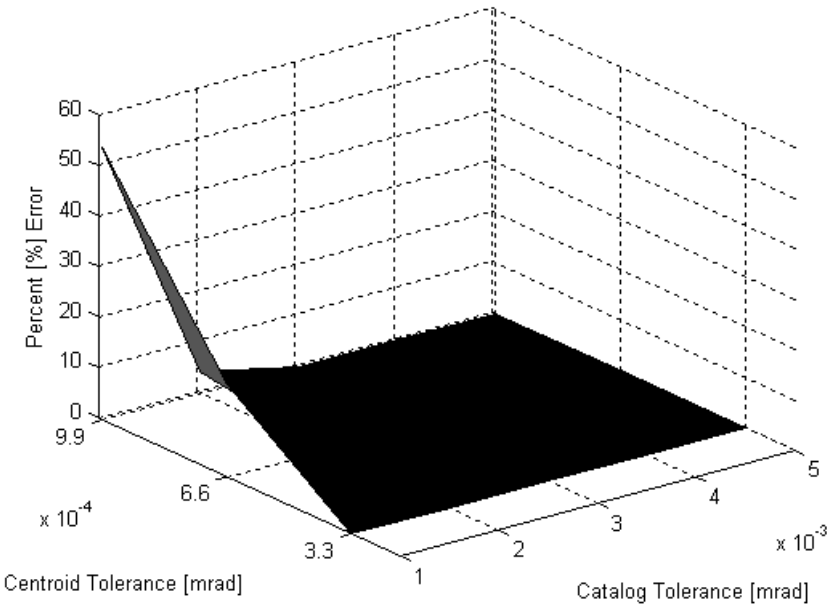

Figure B.26 3-D image empty set of simulated Constrained Pyarmid method as functions of catalog tolerance and centroiding

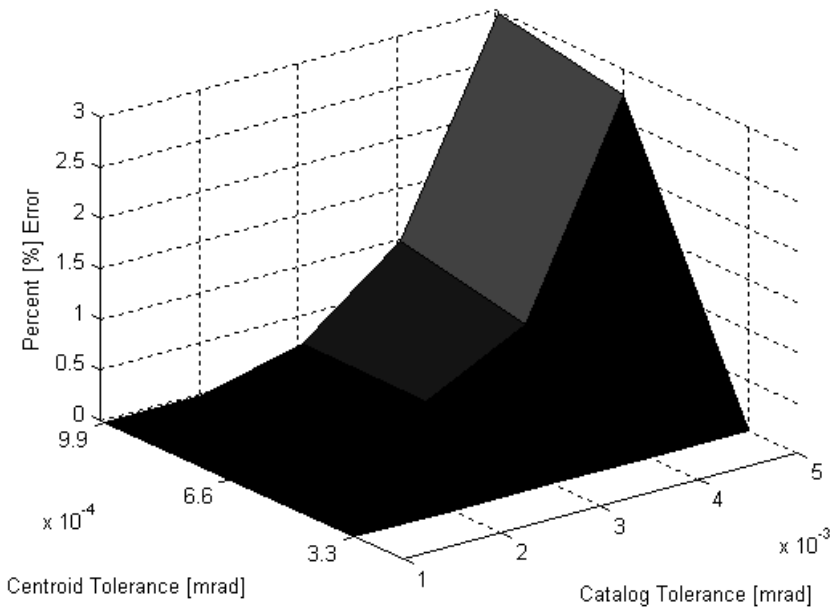

Figure B.27 3-D image solution failure of simulated Comprehensive Pyramid method as functions of catalog tolerance and centroiding

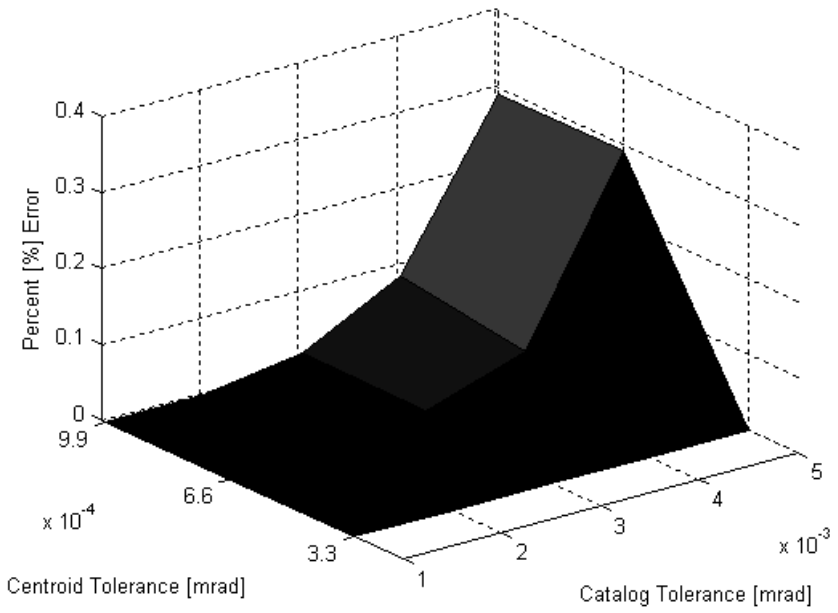

Figure B.28 3-D image match failure of simulated Comprehensive Pyramid method as functions of catalog tolerance and centroiding

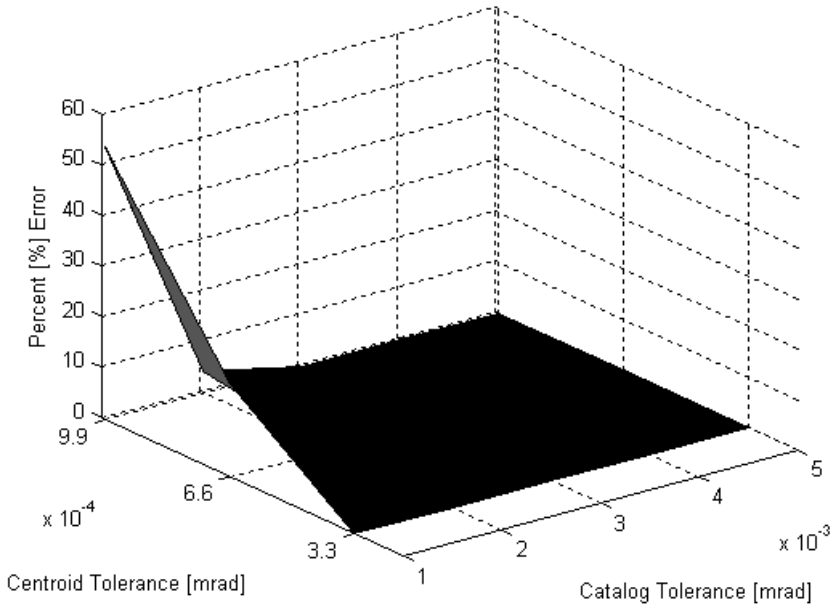

Figure B.29 3-D image empty set of simulated Comprehensive Pyramid method as functions of catalog tolerance and centroiding

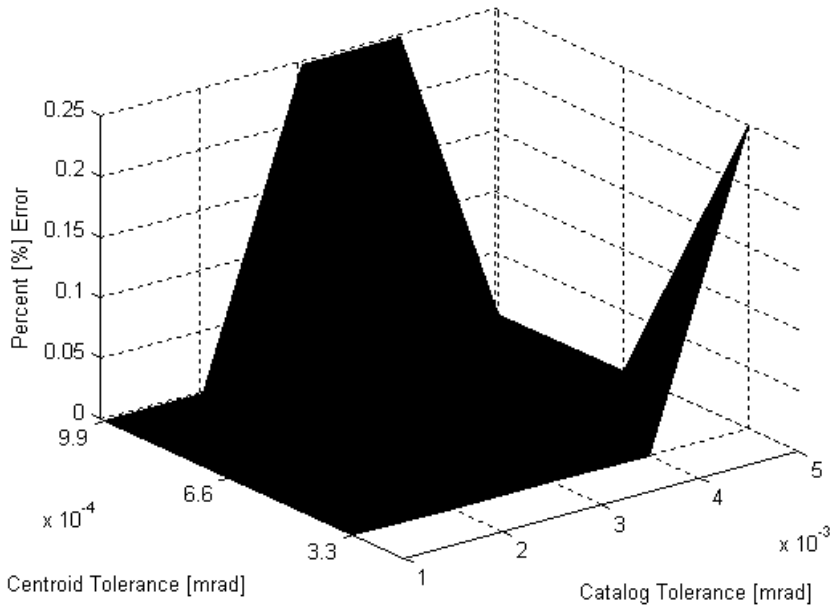

Figure B.30 3-D image solution failure of simulated Modified Pyramid method as functions of catalog tolerance and centroiding

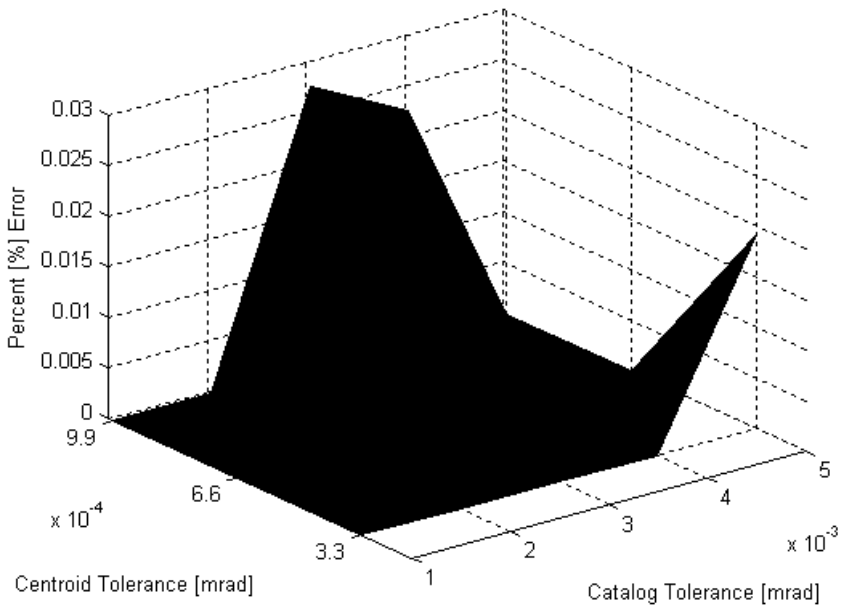

Figure B.31 3-D image match failure of simulated Modified Pyramid method as functions of catalog tolerance and centroiding

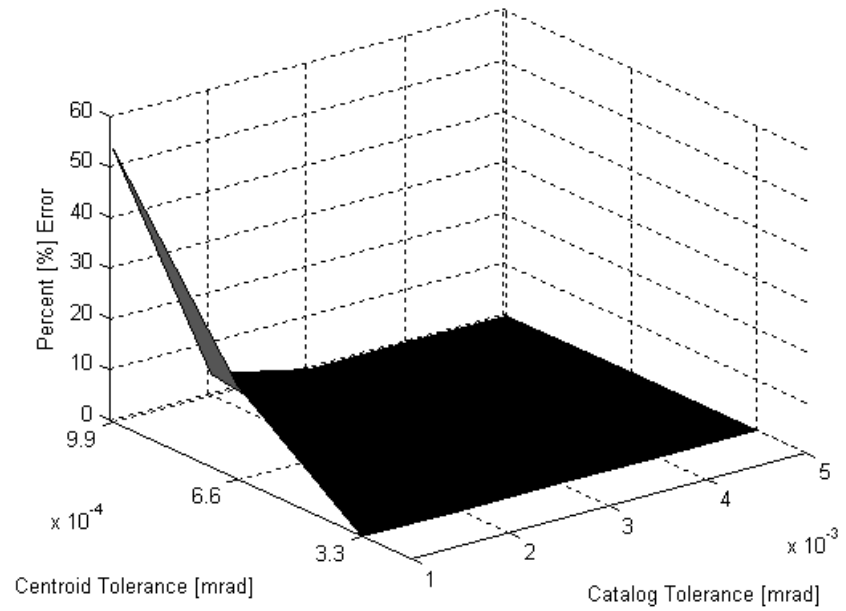

Figure B.32 3-D image empty set of simulated Modified Pyramid method as functions of catalog tolerance and centroiding

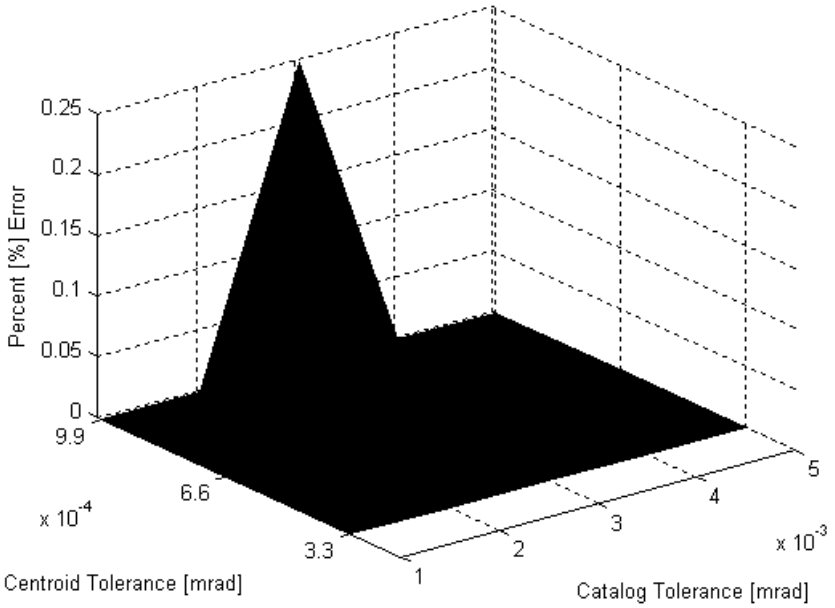

Figure B.33 3-D image solution failure of simulated Pyramid with Voting method as functions of catalog tolerance and centroiding

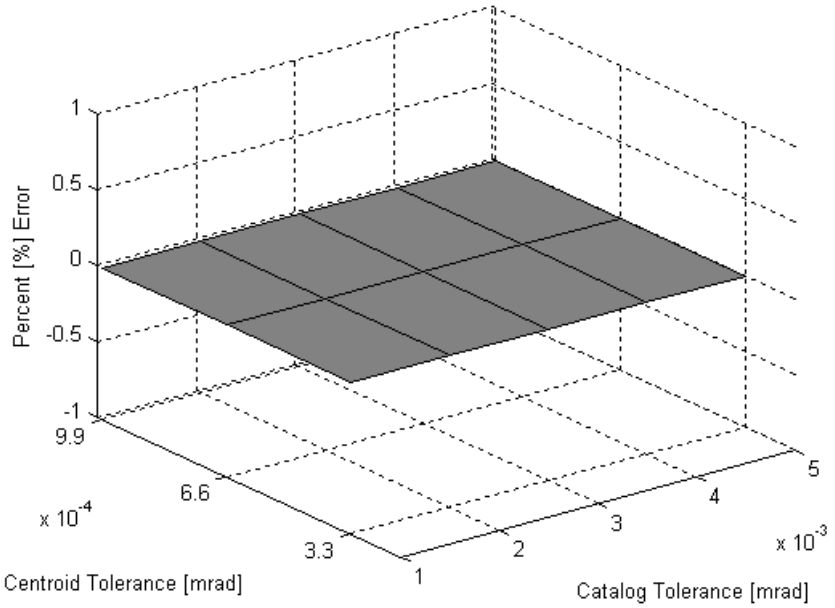

Figure B.34 3-D image match failure of simulated Pyramid with Voting method as functions of catalog tolerance and centroiding

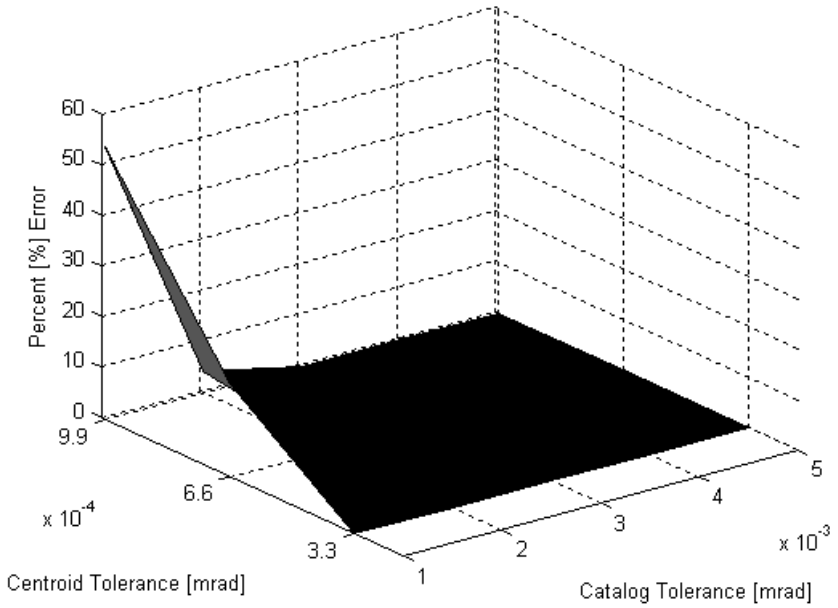

Figure B.35 3-D image empty set of simulated Pyramid with Voting method as functions of catalog tolerance and centroiding

2. **Magnitude 3.5 Threshold**

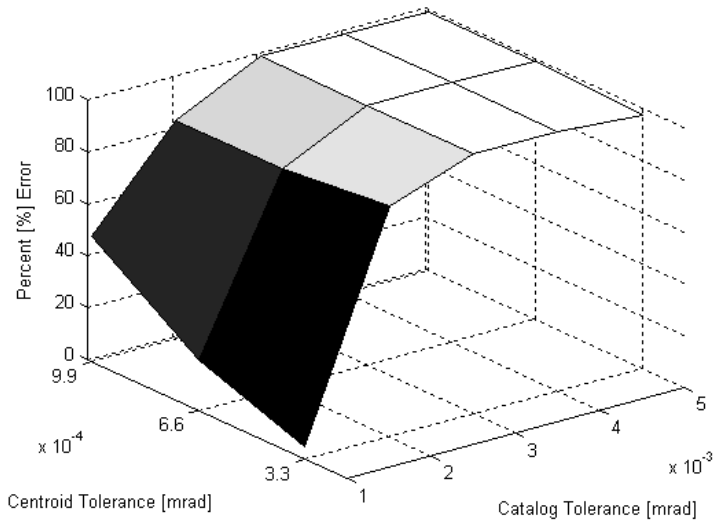

Figure B.36 3-D image solution failure of simulated Two Star method as functions of catalog tolerance and centroiding

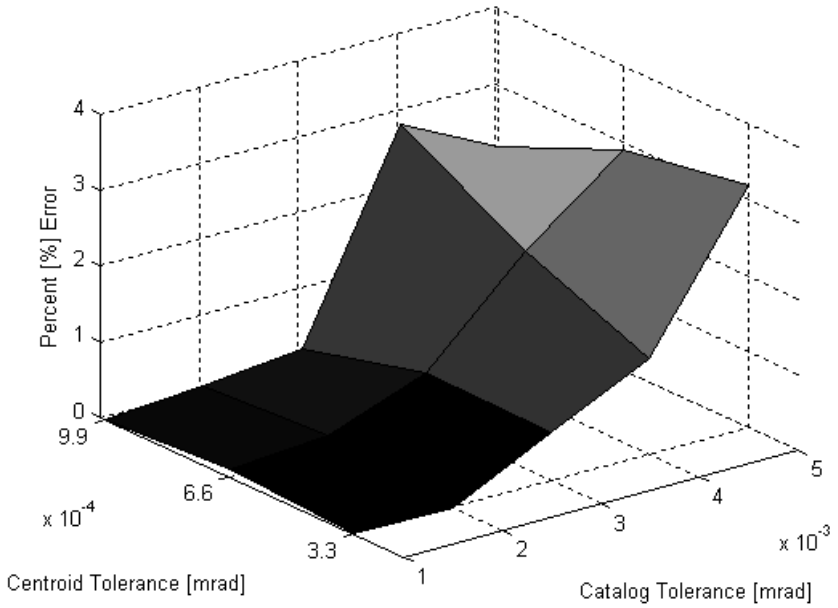

Figure B.37 3-D image match failure of simulated Two Star method as functions of catalog tolerance and centroiding

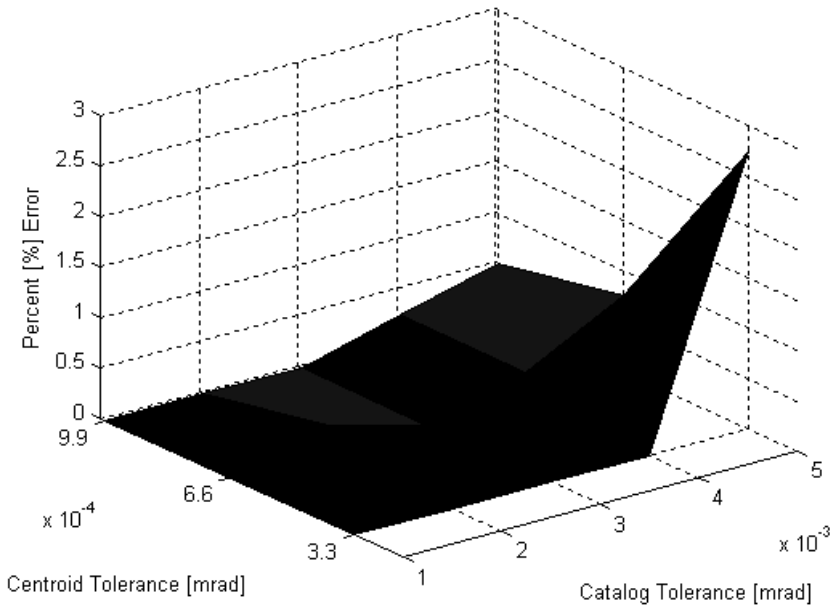

Figure B.38 3-D image empty set of simulated Two Star method as functions of catalog tolerance and centroiding

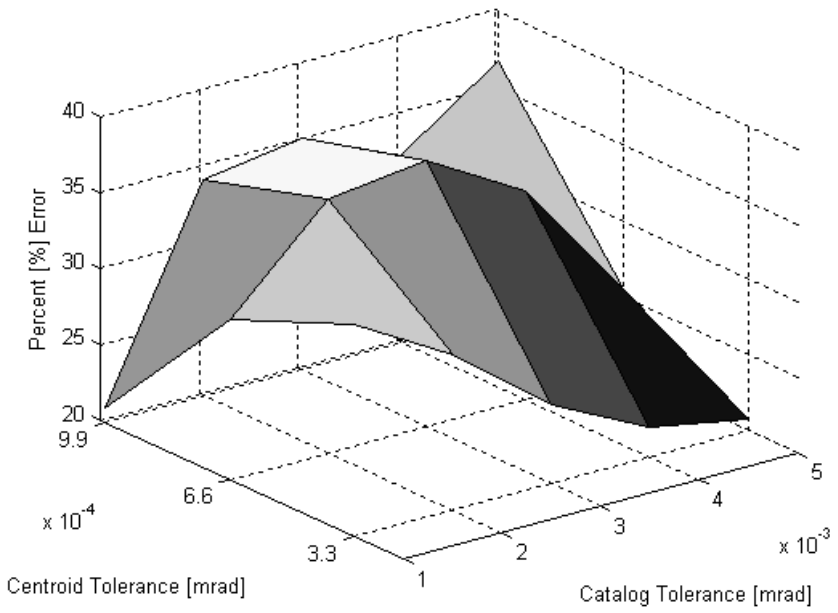

Figure B.39 3-D image solution failure of simulated Liebe method as functions of catalog tolerance and centroiding

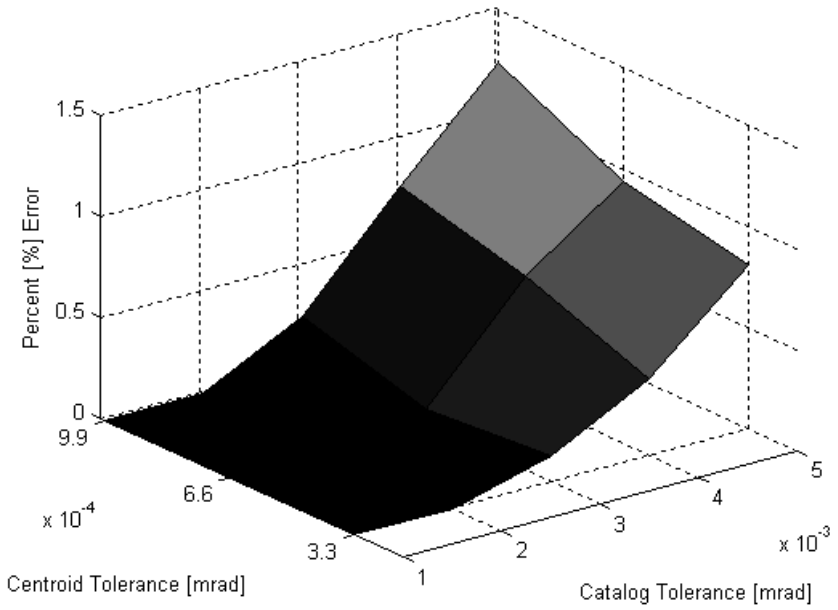

Figure B.40 3-D image match failure of simulated Liebe method as functions of catalog tolerance and centroiding

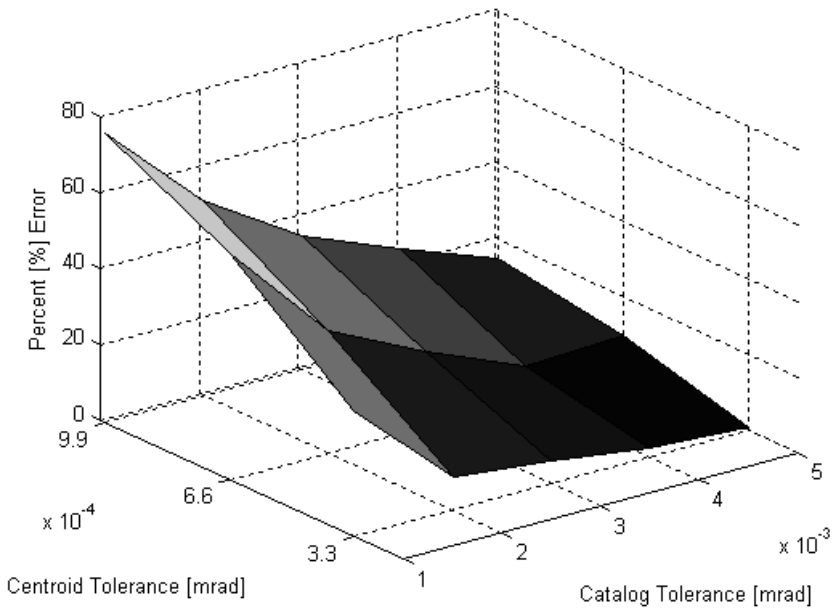

Figure B.41 3-D image empty set of simulated Liebe method as functions of catalog tolerance and centroiding

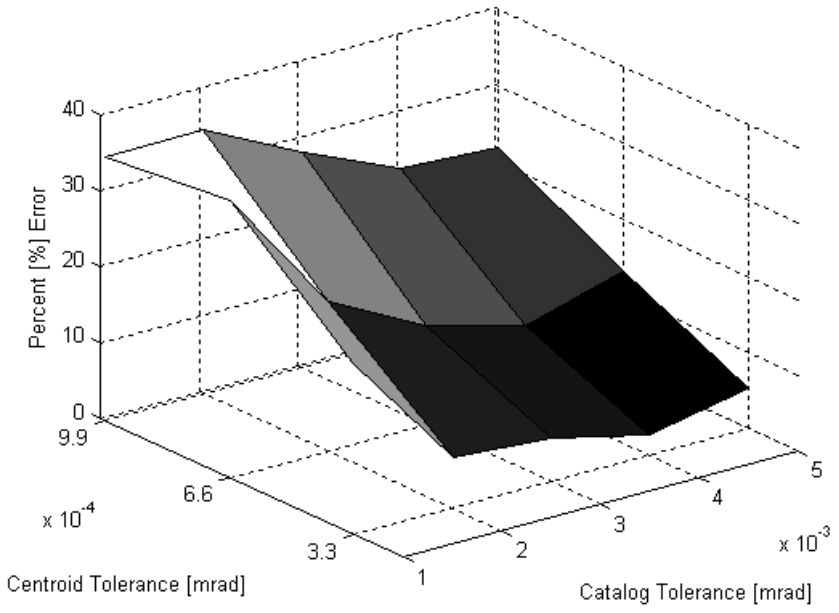

Figure B.42 3-D image solution failure of simulated Liebe with Voting method as functions of catalog tolerance and centroiding

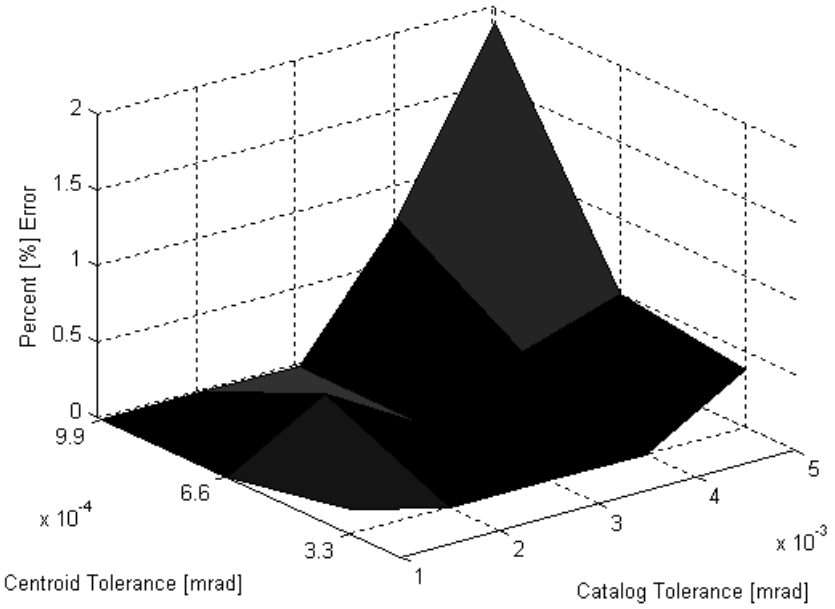

Figure B.43 3-D image match failure of simulated Liebe with Voting method as functions of catalog tolerance and centroiding

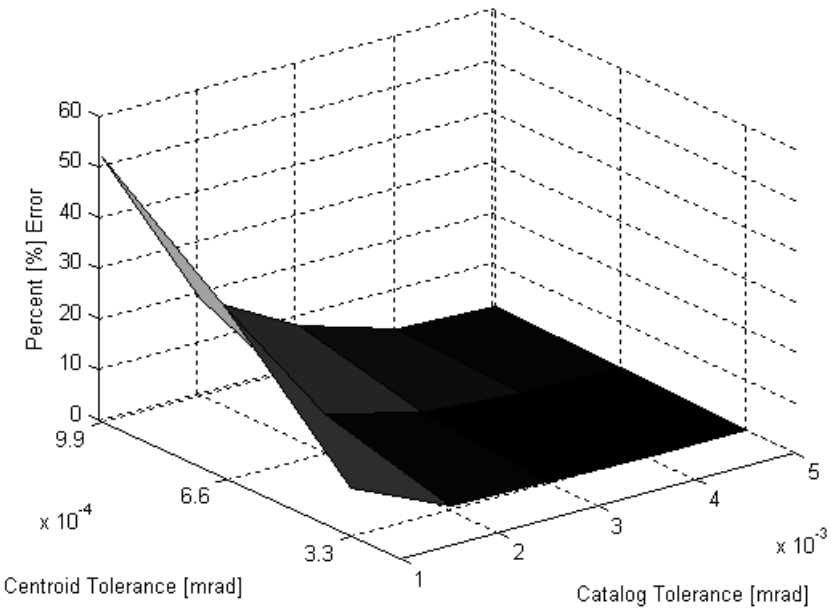

Figure B.44 3-D image empty set of simulated Liebe with Voting method as functions of catalog tolerance and centroiding

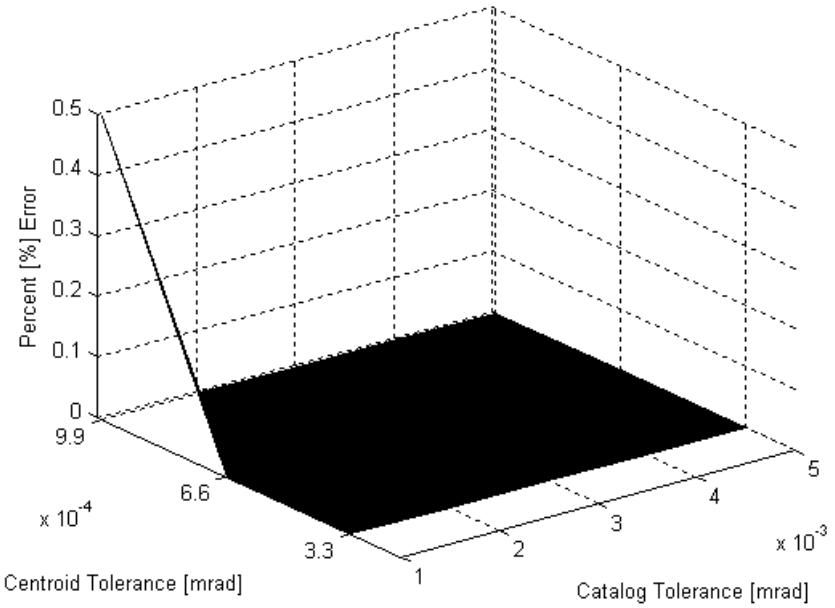

Figure B.45 3-D image solution failure of simulated Brätt method as functions of catalog tolerance and centroiding

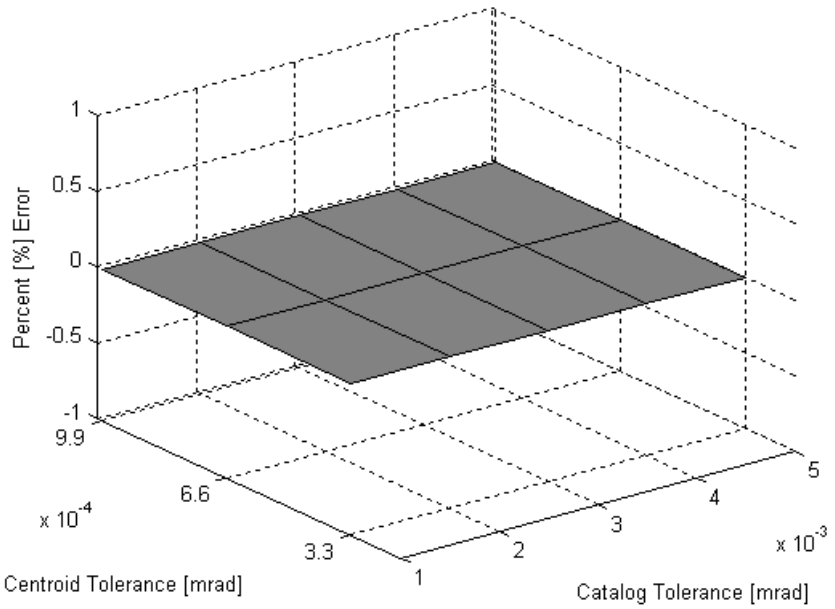

Figure B.46 3-D image match failure of simulated Brätt method as functions of catalog tolerance and centroiding

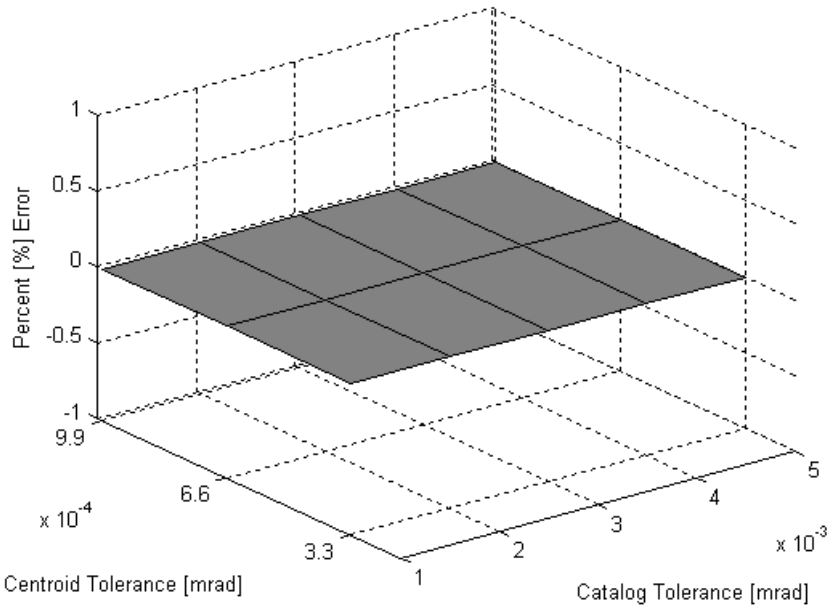

Figure B.47 3-D image empty set of simulated Brätt method as functions of catalog tolerance and centroiding

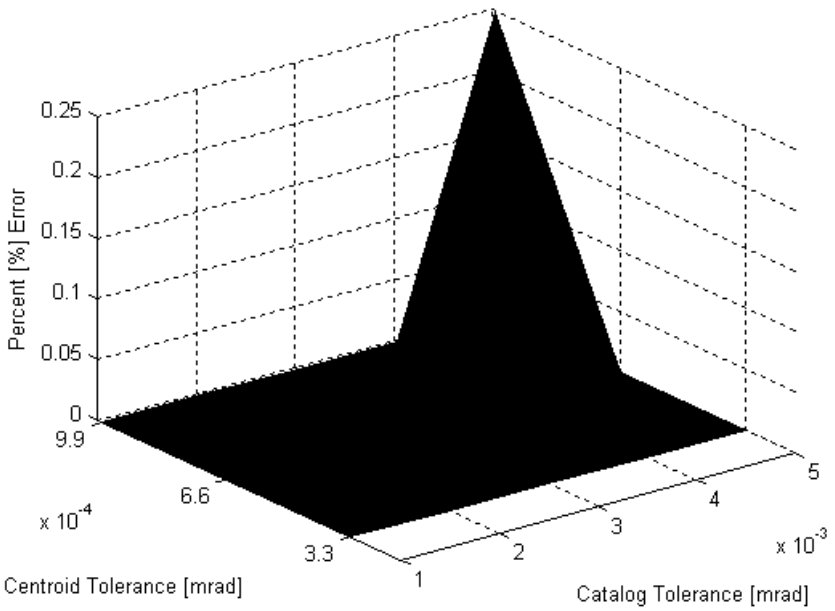

Figure B.48 3-D image solution failure of simulated Constrained Pyramid method as functions of catalog tolerance and centroiding

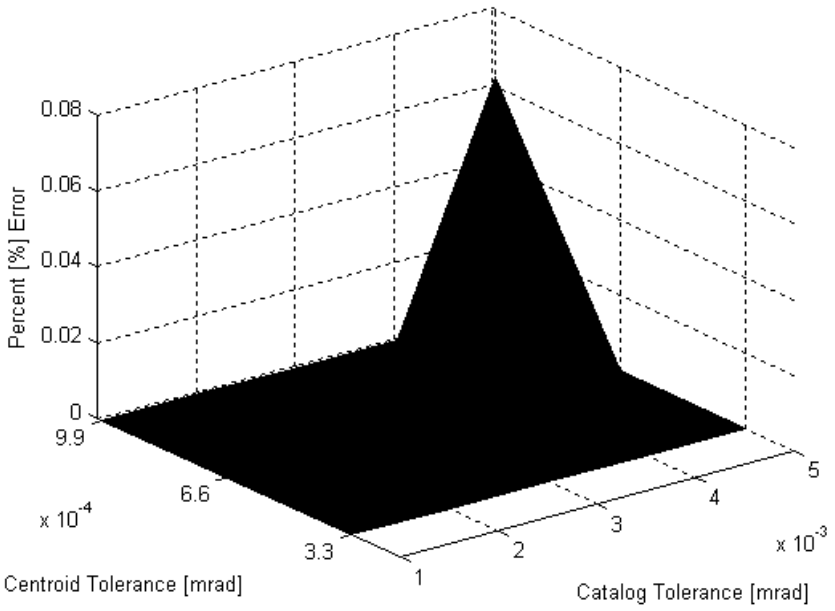

Figure B.49 3-D image match failure of simulated Constrained Pyramid method as functions of catalog tolerance and centroiding

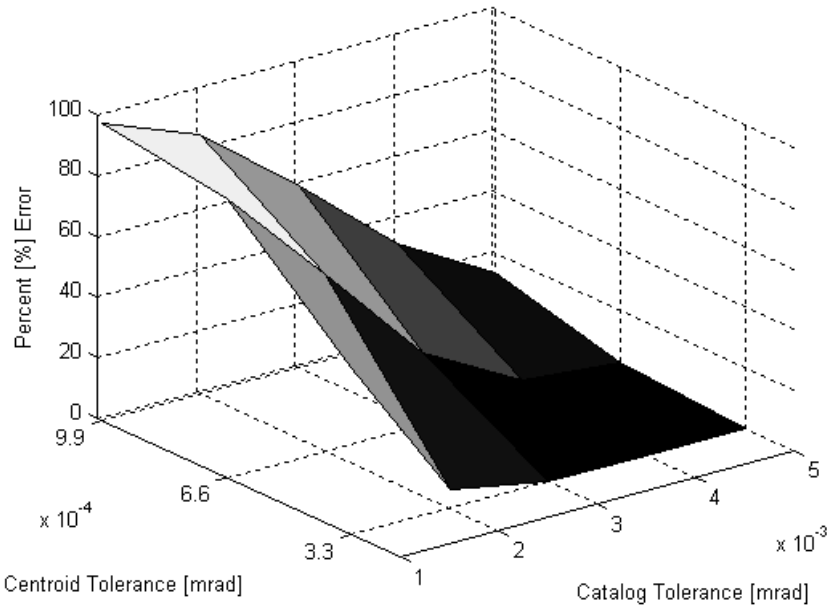

Figure B.50 3-D image empty set of simulated Constrained Pyramid method as functions of catalog tolerance and centroiding

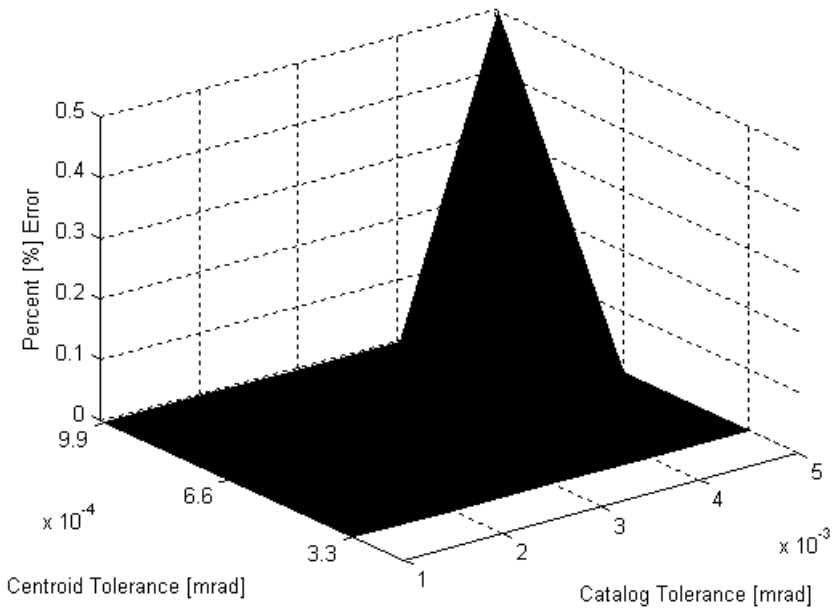

Figure B.51 3-D image solution failure of simulated Comprehensive Pyramid method as functions of catalog tolerance and centroiding

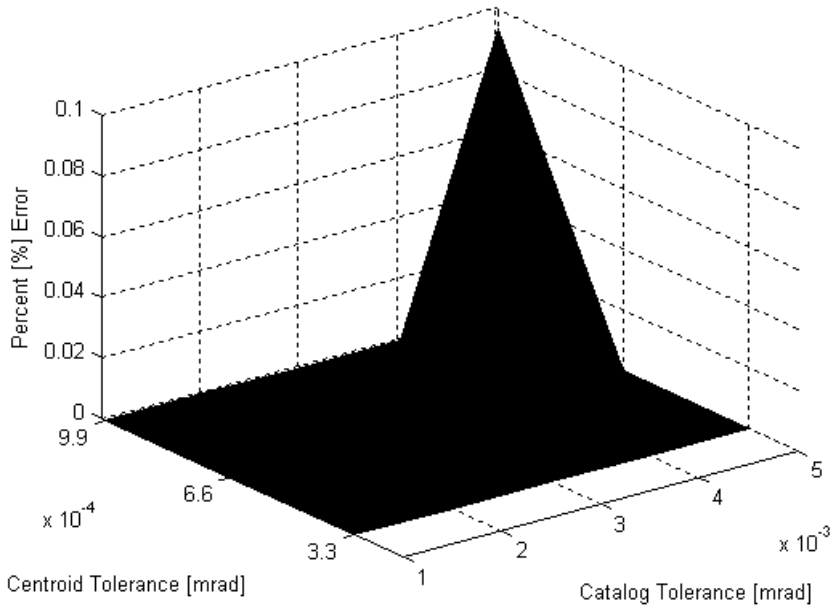

Figure B.52 3-D image match failure of simulated Comprehensive Pyramid method as functions of catalog tolerance and centroiding

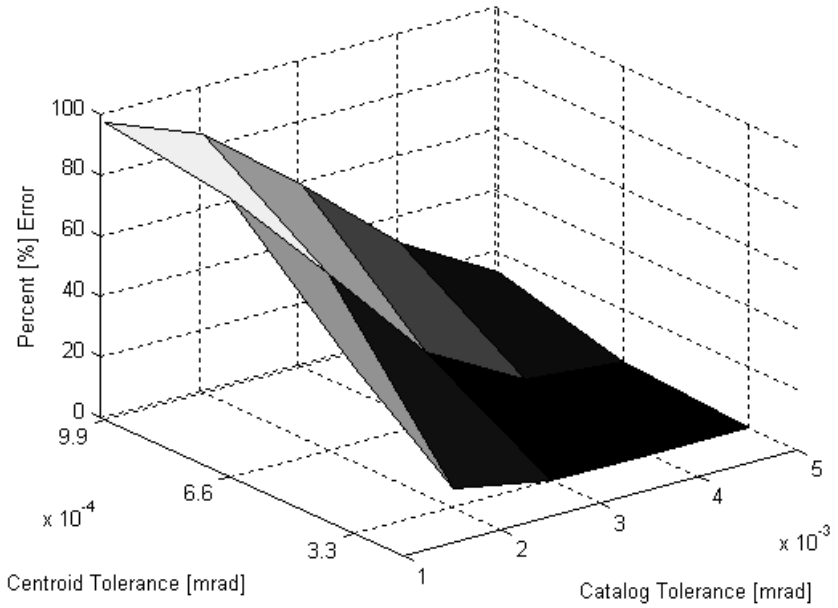

Figure B.53 3-D image empty set of simulated Comprehensive Pyramid method as functions of catalog tolerance and centroiding

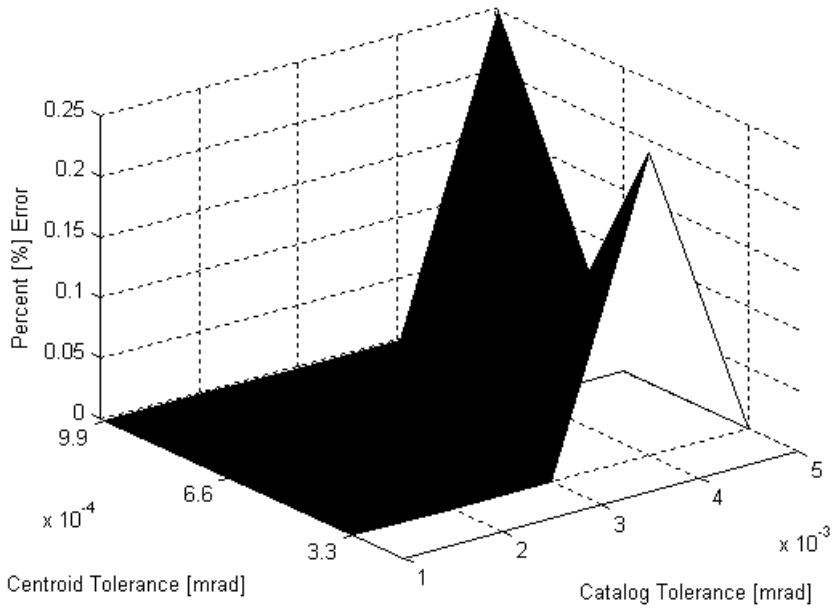

Figure B.54 3-D image solution failure of simulated Modified Pyramid method as functions of catalog tolerance and centroiding

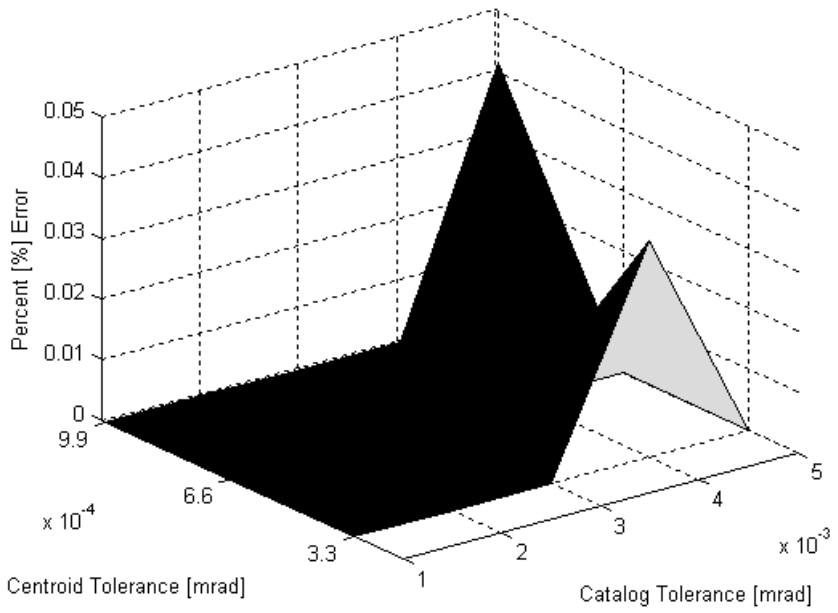

Figure B.55 3-D image match failure of simulated Modified Pyramid method as functions of catalog tolerance and centroiding

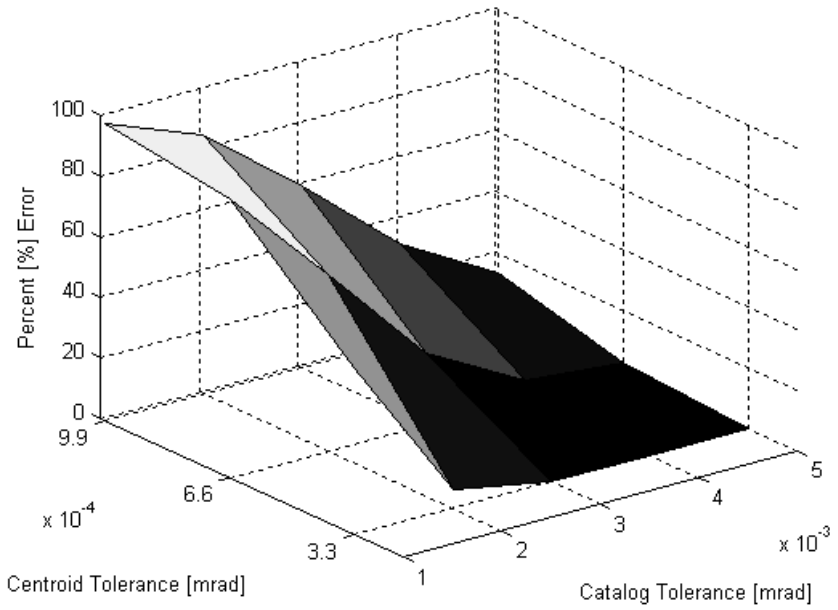

Figure B.56 3-D image empty set of simulated Modified Pyramid method as functions of catalog tolerance and centroiding

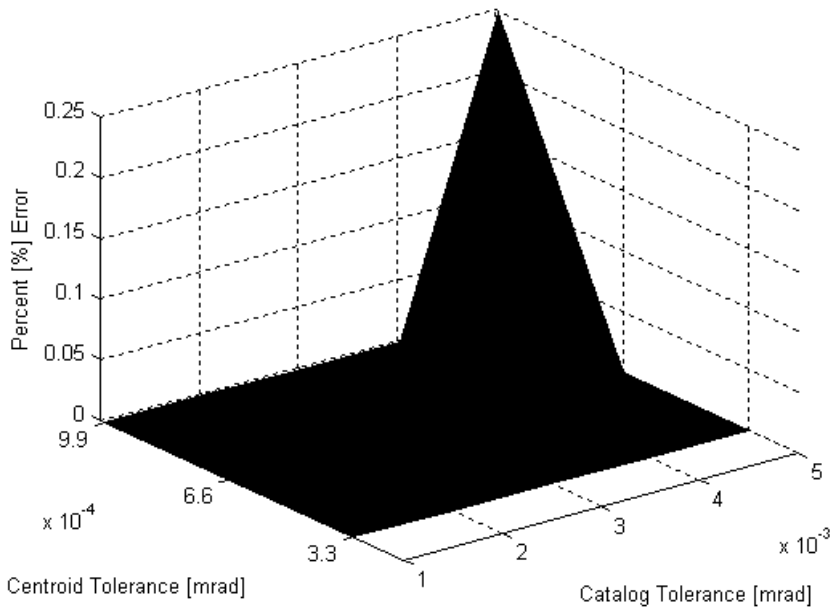

Figure B.57 3-D image solution failure of simulated Pyramid with Voting method as functions of catalog tolerance and centroiding

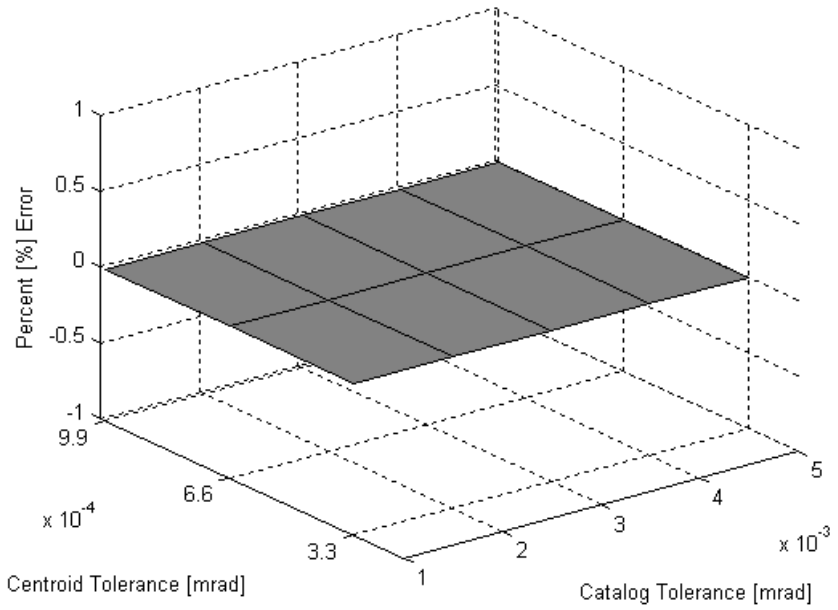

Figure B.58 3-D image match failure of simulated Pyramid with Voting method as functions of catalog tolerance and centroiding

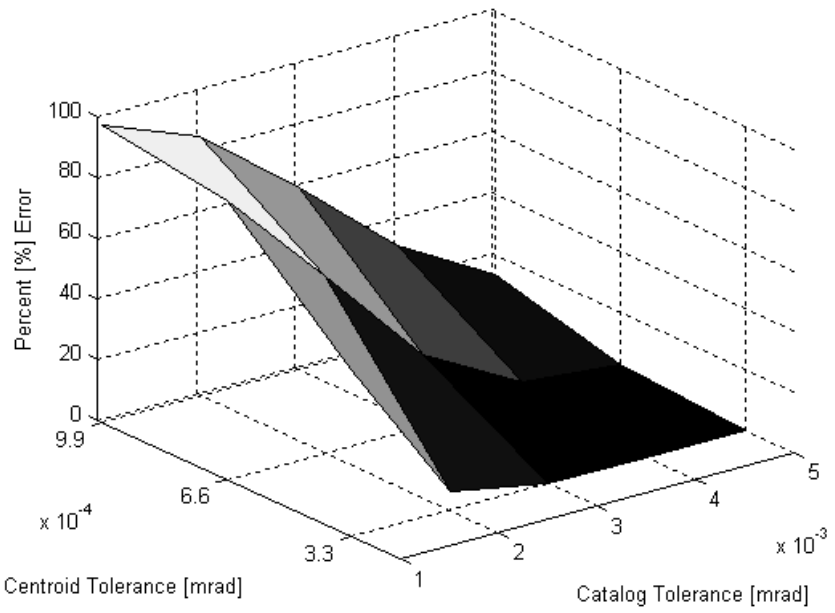

Figure B.59 3-D image empty set of simulated Pyramid with Voting method as functions of catalog tolerance and centroiding

II. Additional Experimental Data Figures

1. Magnitude 3 Threshold – OCT

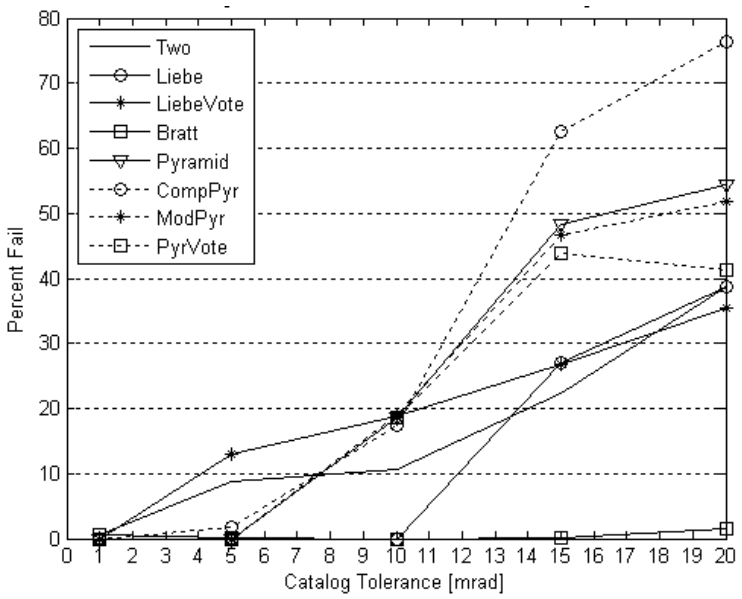

Figure B.60 Oct data at mag. 3 showing average false matches for all algorithms

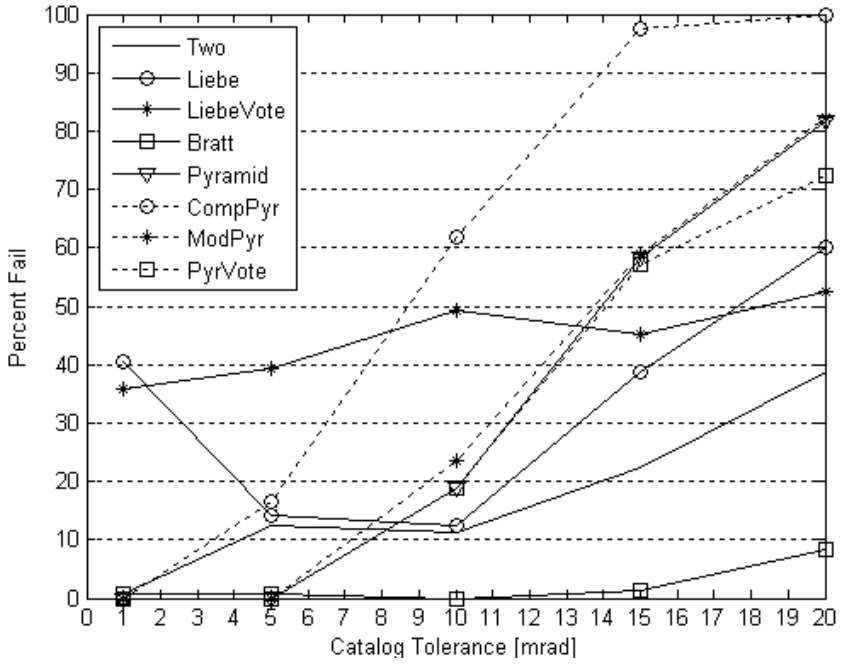

Figure B.61 Oct data at mag. 3 showing average solution failures for all algorithms

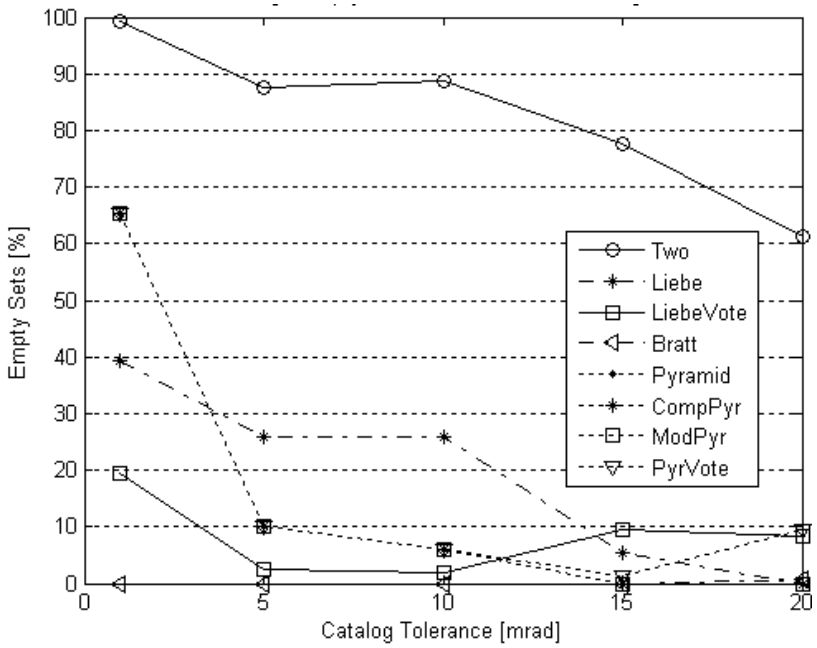

Figure B.62 Oct data at mag. 3 showing average empty set for all algorithms

2. **Magnitude 3.5 Threshold - OCT**

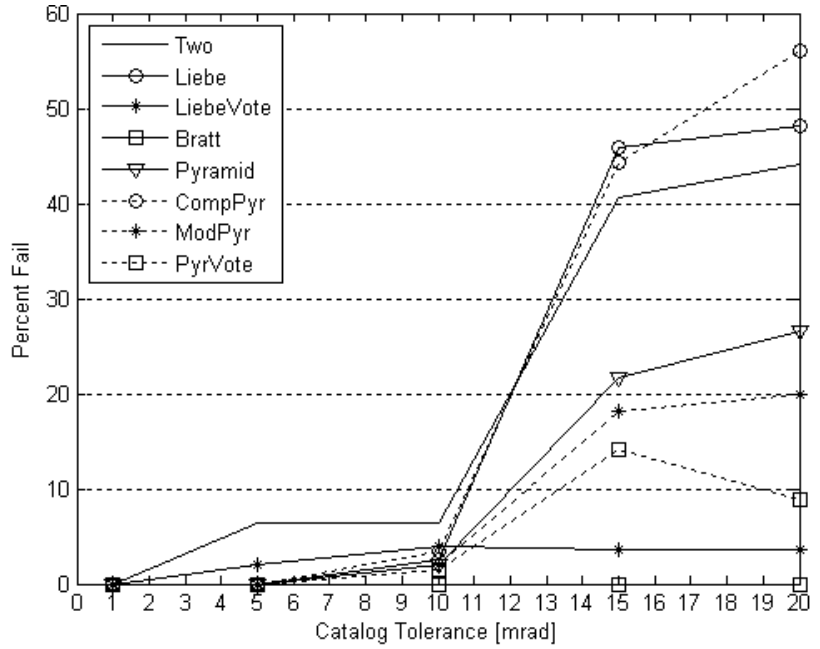

Figure B.63 Oct data at mag. 3.5 showing average false matches for all algorithms

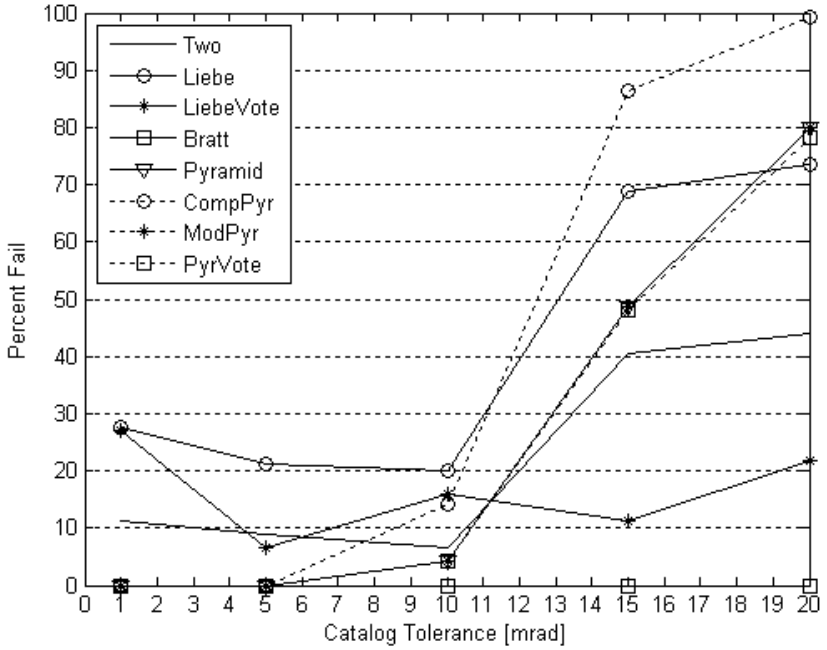

Figure B.64 Oct data at mag. 3.5 showing average false solutions for all algorithms

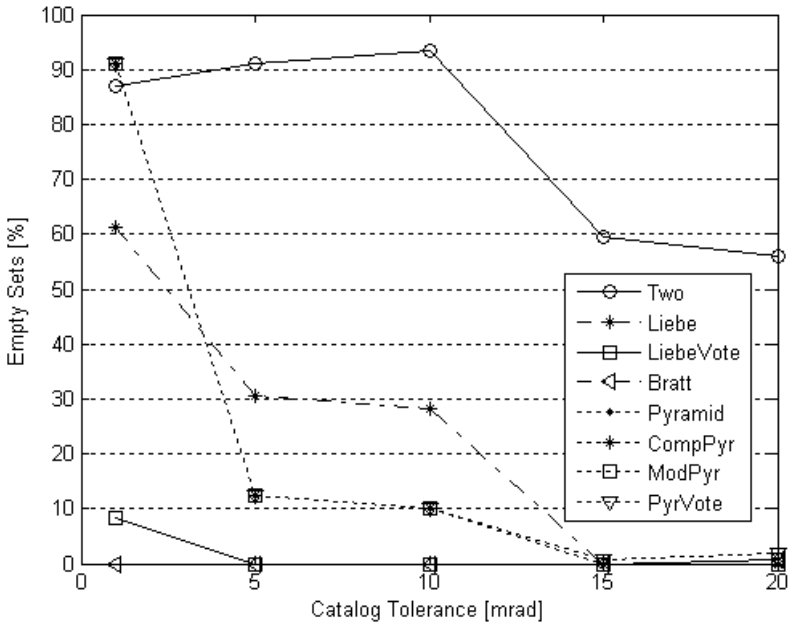

Figure B.65 Oct data at mag. 3.5 showing average empty set for all algorithms

3. **Magnitude 4 Threshold - OCT**

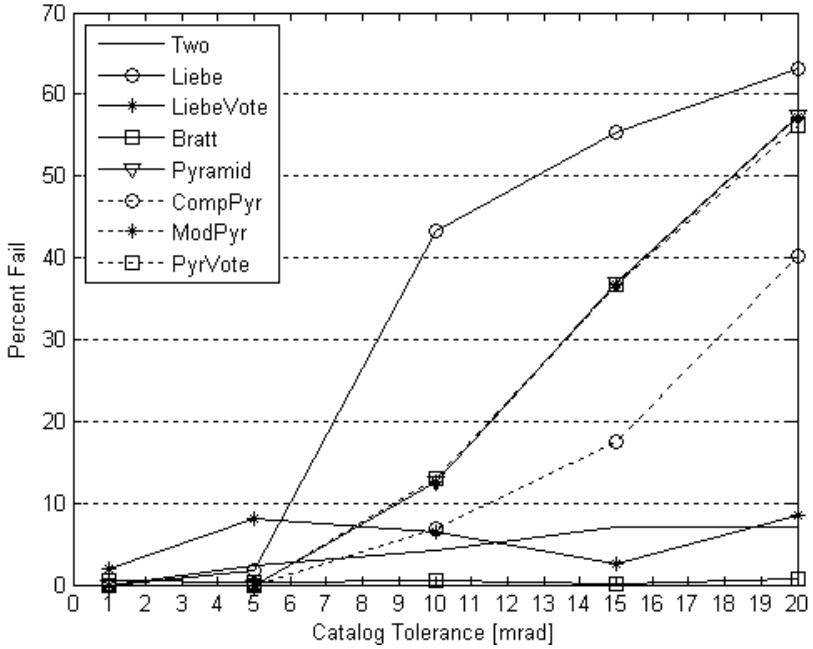

Figure B.66 Oct data at mag. 4 showing average false matches for all algorithms

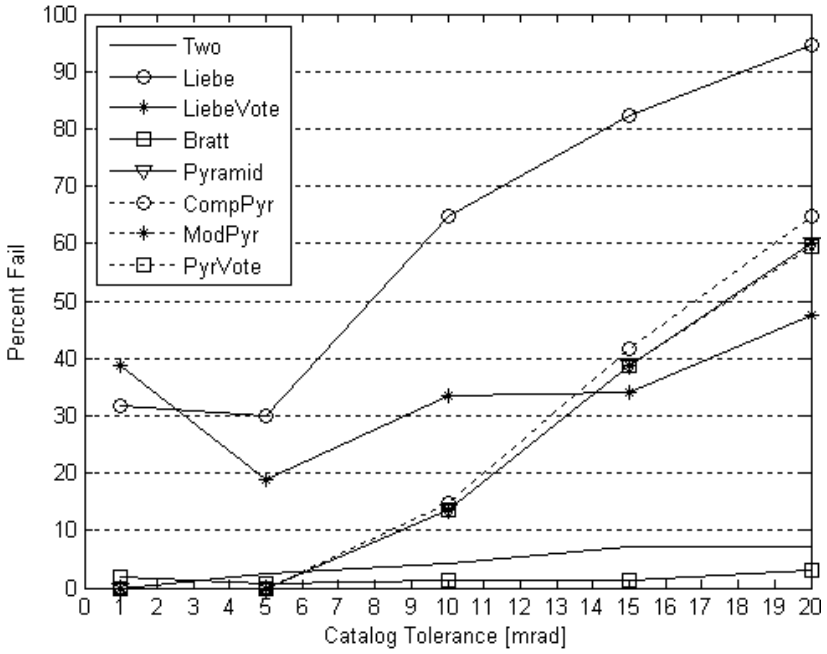

Figure B.67 Oct data at mag. 4 showing average false solutions for all algorithms

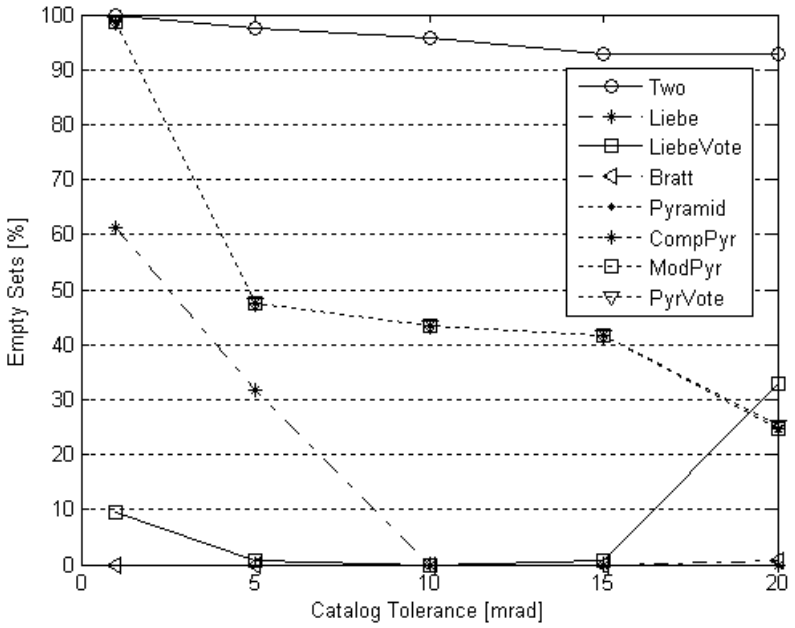

Figure B.68 Oct data at mag. 4 showing average empty set for all algorithms

4. **Magnitude 3 Threshold - NOV**

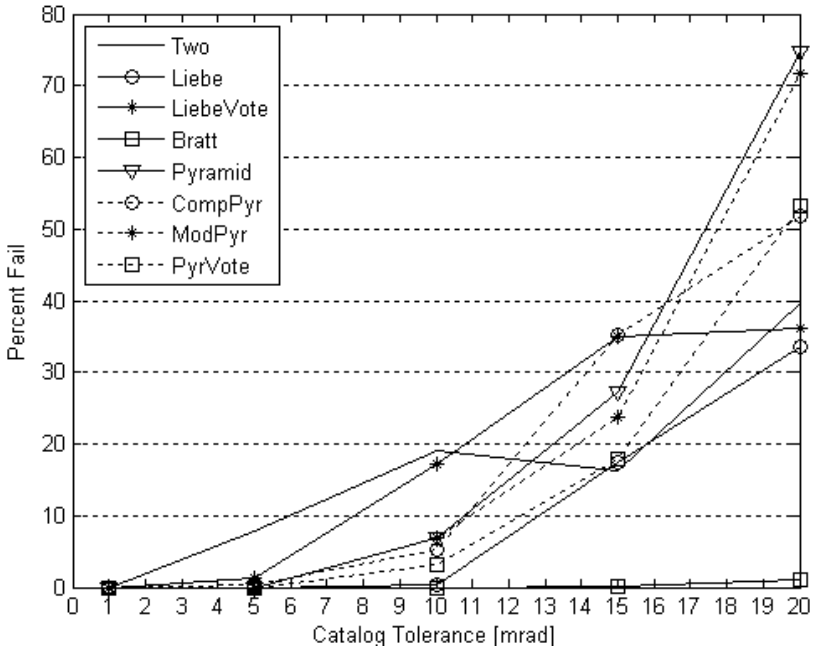

Figure B.69 Nov data at mag. 3 showing average false matches for all algorithms

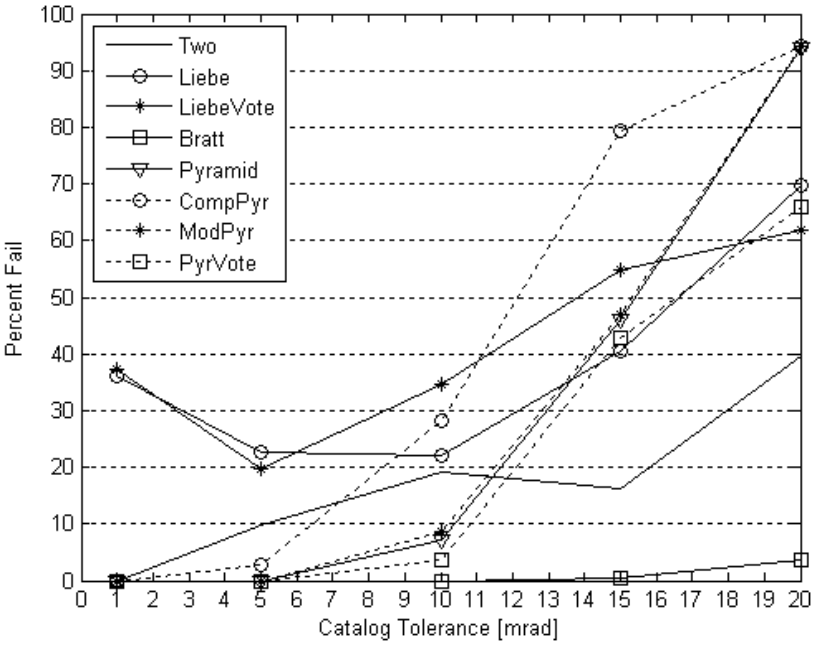

Figure B.70 Nov data at mag. 3 showing average false solutions for all algorithms

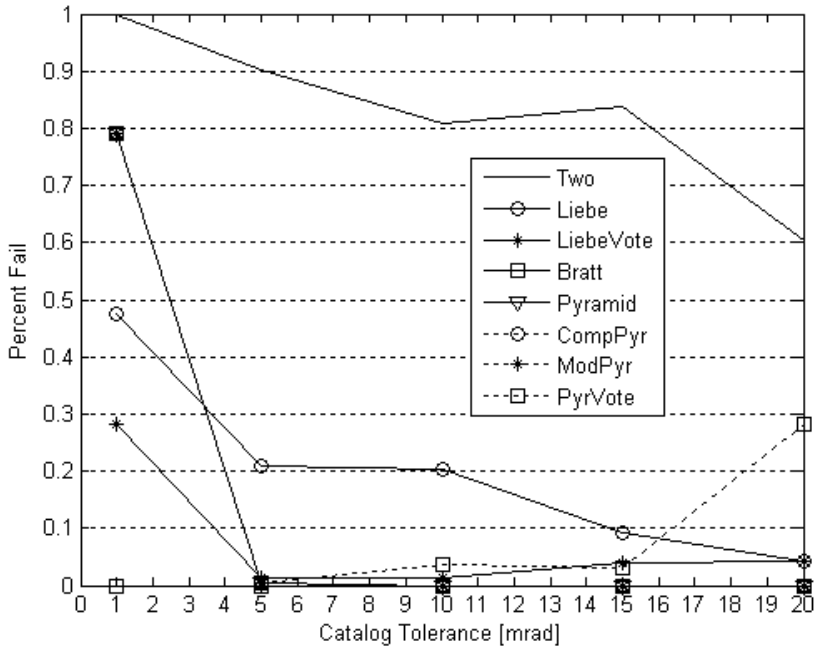

Figure B.71 Nov data at mag. 3 showing average empty set for all algorithms

5. **Magnitude 3.5 Threshold - NOV**

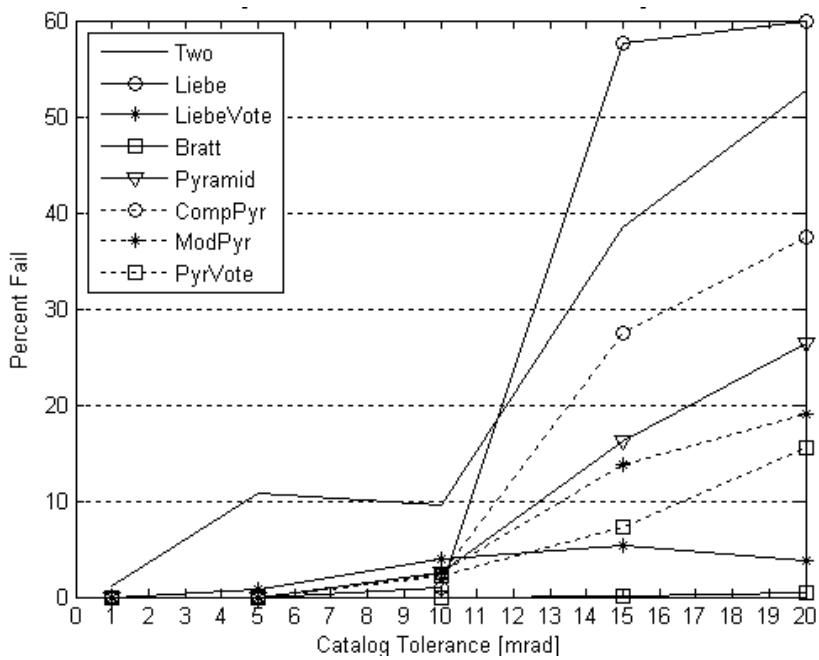

Figure B.72 Nov data at mag. 3.5 showing average false matches for all algorithms

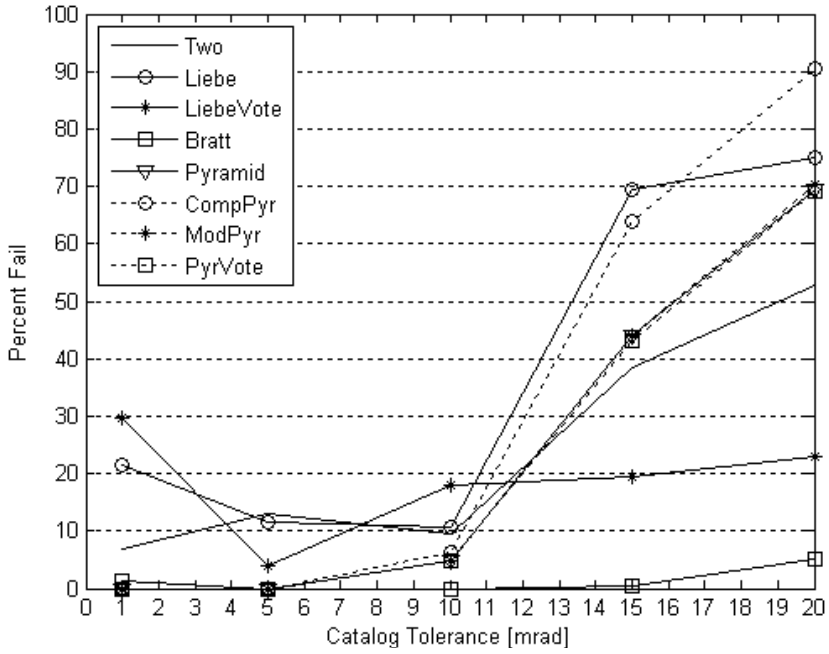

Figure B.73 Nov data at mag. 3.5 showing average false solutions for all algorithms

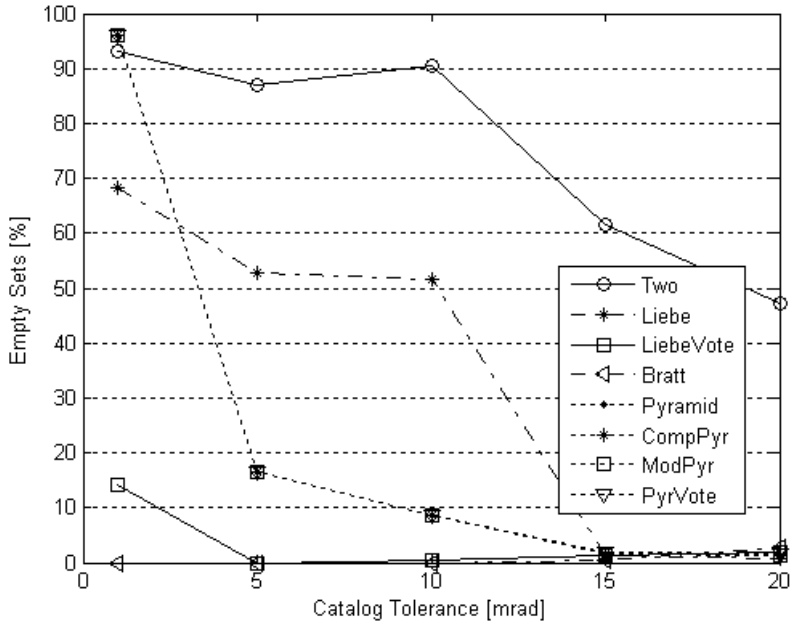

Figure B.74 Nov data at mag. 3.5 showing average empty set for all algorithms

6. **Magnitude 4 Threshold - NOV**

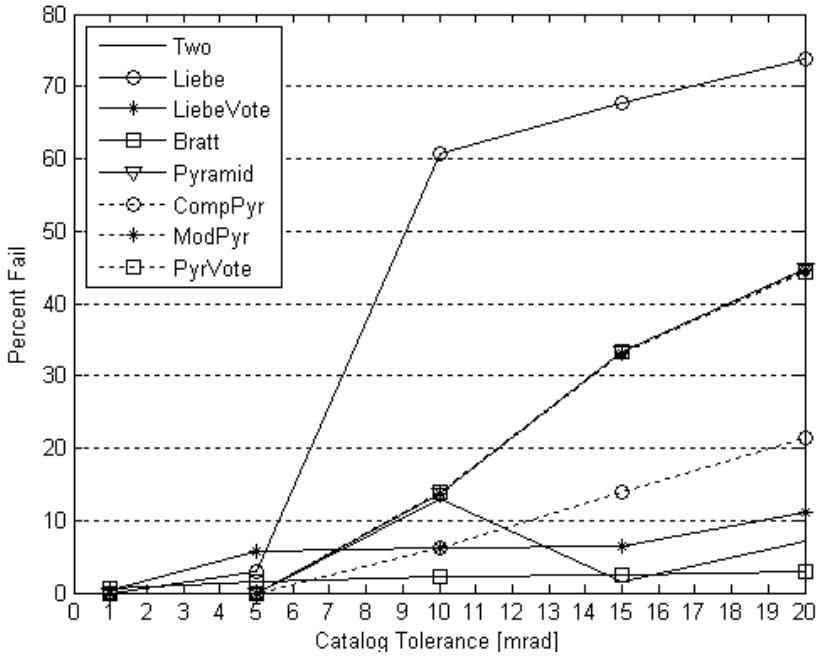

Figure B.75 Nov data at mag. 4 showing average false matches for all algorithms

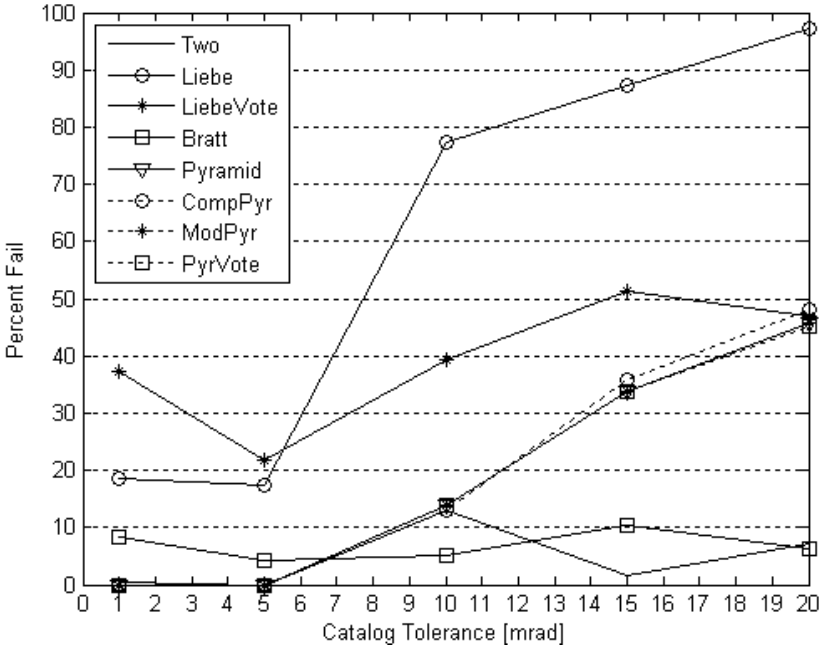

Figure B.76 Nov data at mag. 4 showing average false solutions for all algorithms

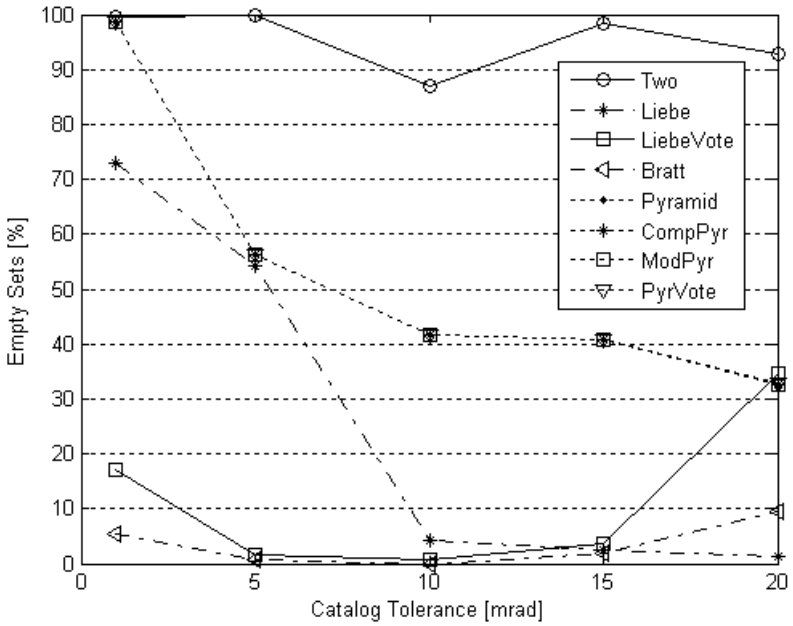

Figure B.77 Nov data at mag. 4 showing average empty set for all algorithms