The Thesis committee for Christopher Ryan McBryde
Certifies that this is the approved version of the following thesis:

A Star Tracker Design for CubeSats

APPROVED BY

SUPERVISING COMMITTEE:

E. Glenn Lightsey, Supervisor

Chris D'Souza

A Star Tracker Design for CubeSats

by

Christopher Ryan McBryde, B.S.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2012

To my parents, Ryan and Laura, for their constant encouragment and support

Acknowledgments

I would like to begin by thanking my advisor, Dr. E. Glenn Lightsey. Thank you for your steady guidance and the freedom to explore my interests. This thesis could not have been completed without your support and I look forward to continuing our partnership through my doctoral research.

I would also like to thank my other reader, Dr. Christopher D'Souza. Your support of my research and advice regarding this paper were invaluable.

Thank you to my colleagues in the Satellite Design Laboratory, including Travis Imken, Andrew Fear, Andrew Leba, Alexandra Long, and many others. The SDL is a wonderful and enriching environment, and it owes that to intelligent motivated people like you. Special thanks to Henri Kjellberg, for his mentorship regarding my masters journey, and Katharine Brumbaugh for taking that journey with me as well as exemplary leadership of the SDL.

I also thank Noah Smith for your insight on the subject of star tracking and never-ending enthusiasm for discussion.

Thank you to Dr. John Christian for the brilliant research he completed for his dissertation, much of which served as foundation for my own research.

To my girlfriend, Alex Rodriguez, thank you for your patience when this thesis drove me up a wall and delicious meals after a long day in the lab. It meant more than you know.

Finally, thank you to my family. To my parents, Laura and Ryan McBryde, I cannot begin to tell you what your support throughout my endeavours has meant to me. I could not ask for more loving and supportive parents. Thank you also to my sister Katy and my brother Patrick. You have always been there for me and I look forward to seeing what you can both accomplish in the future.

Thank you to all the people mentioned above and all others whom I neglected to mention. I am lucky to be surrounded by intellegent, kind, and caring people in my life, without whom none of this would be possible.

Christopher Ryan McBryde

The University of Texas at Austin

February 2012

A Star Tracker Design for CubeSats

Christopher Ryan McBryde, M.S.E.
The University of Texas at Austin, 2012

Supervisor: E. Glenn Lightsey

This research outlines a low-cost, low-power, arc-minute accurate star tracker that is designed for use on a CubeSat. The device is being developed at the University of Texas at Austin for use on two different 3-unit CubeSat missions. The hardware consists of commercial off-the-shelf parts designed for use in industrial machine vision systems and employs a 1024x768 grey-scale charge coupled device (CCD) sensor. The software includes the three standard steps in star tracking: centroiding, star identification, and attitude determination. Centroiding algorithms were developed in-house. The star identification code was adapted from the voting method developed by Kolomenkin, et al. Attitude determination was performed using Markley's singular value decomposition method. The star tracker was then tested with internal simulated star-fields. The resulting accuracy was less than an arcminute. It was concluded that this system is a viable option for CubeSats looking to improve their attitude determination. On-orbit demonstration of the system is planned when the star tracker flies on the planned CubeSat missions in 2013 or later. Further testing with external simulated star fields and night sky tests are also planned.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Military, commercial, and educational organizations see the usefulness and economy of employing CubeSat-sized satellites to accomplish various missions. Space weather, communication, and surveillance are all tasks that, in the past, could only be given to much larger and more expensive spacecraft. Unfortunately, one major challenge that stands in the way of these organizations fully embracing CubeSats as a platform is precision attitude determination and control. As of yet, no CubeSat has demonstrated the necessary attitude knowledge and pointing accuracy to take high-resolution images or employ high-bandwidth data transfer via directional antennae, for example. Attitude determination accuracy is necessarily limited by sensor measurement accuracy and currently only lower accuracy attitude measurement instruments such as sun sensors and magnetometers have flown on CubeSats with success. Star trackers will provide the level of attitude determination needed to support more challenging pointing requirements. This technology has not been fully explored in the realm of CubeSats.

1.1 The CubeSat platform

CubeSat is a standard developed by California Polytechnic State University (Cal Poly) that facilitated and continues to foster low-cost satellite missions [7]. CubeSats are sized using "units." Typically, they are 1-U or 3-U, though 6-U satellites are becoming more prevalent. A 1-U CubeSat has dimensions of approximately 10 cm on each side with

a maximum mass of 1.333 kg, whereas 3-U satellites are 10 cm by 10 cm by 34 cm and can weigh up to 4 kg. The advantage to the CubeSat standard is that the satellite can be enclosed within a standardized launcher, such as the Poly Picosatellite Orbital Deployer, or P-POD, developed by Cal Poly [7]. This encapsulation reduces the risk of flying a CubeSat as a secondary or tertiary payload. The effect of the CubeSat standard is evident through analysis of the satellite launches over the past 20 years. According to Swartwout [8], the average number of small satellites weighing less than 100 kg launched per year was 14.4 from 1990-2001. Of those, about 28% were in the less than 10 kilogram range. But in the years since 2000, there were an average of 20 small satellite launches per year. The percentage of less than 10 kilogram satellites has risen to 57%, and CubeSats, which did not exist in the 1990s, now comprise about half of the small satellite launches in the past three years.

1.2 Relevance to CubeSats

What the results from Swarthout [8] means is that the opportunity exists for many future CubeSat launches. Because of the miniaturization of electronics and battery technology, largely as a result of the smartphone market, CubeSats now have the sensing capabilities and processing power that in the past could only be found on much larger satellites. Where CubeSats lag behind, though, is in accurate attitude determination. As of yet, no CubeSat has flown with a functioning star tracker.

On June 30, 2003, CanX-1, a 1-U CubeSat developed by the University of Toronto, was launched from Plesetsk, Russia [9]. CanX-1 contained a star-tracker and horizon sensor for attitude determination. However, radio contact with the satellite was never established [10]. The successor to that satellite, CanX-2, was launched from Sriharikota, India on April 28, 2008. That satellite is equipped with CMOS imagers for verification of the ADC system

in post-processing, but they were never used as a star tracker on-orbit [11].

As far as satellites currently in development, the BRITE Nano-Satellite Constellation mission plans to use a miniature star tracker developed by Sinclair Interplanetary on its bus. The launch date for this mission is undetermined [12]. Also in development is ExoPlanetSat from the Massachusetts Institute of Technology [13]. It will use the same charged-coupled device (CCD) for star tracking as it will for detecting exoplanets, and a launch date has not been determined.

Currently, the CubeSat attitude determination platform is limited to sun sensors, magnetometers, and inertial measurements. A suite of sun sensors can provide fairly accurate measurements, but can only operate in sunlight. For a low Earth orbit (LEO) satellite as much as 30% of the orbit may occur in darkness. Magnetometers are small and can provide precise measurements with correct calibration. Their drawback lies with limited magnetic field knowledge and electromagnetic interference due to the cramped confines of a CubeSat. Finally, microelectromechanical systems (MEMS) gyroscopes are small enough to fit on a smartphone and certainly on a CubeSat. However, they suffer from rapid drift and certainly could not hold an accurate attitude estimate during the 15 minute eclipse period of a LEO satellite. For CubeSats to be seen as a viable scientific and technological platform, they must be able to provide accurate attitude determination, and the most direct way to accomplish that goal is via star trackers.

1.3 Contributions and organization of the thesis

The research in this document aims to fill a gap in the instrumentation which is currently available for use on the CubeSat platform by providing the design and implementation of a star tracker to be used on CubeSats. This star tracker design meets the stringent

size and power requirements for CubeSats while still providing a significant attitude determination improvement. Chapter 2 provides an overview of the various processes required for star tracking and the current techniques for accomplishing them. The algorithms that were chosen for this star tracker are explained in greater detail along with their mathematical bases. Chapter 3 explains the software implementation of the techniques. The various functions and their flows are explained as well as the overall organization of the star tracking program. Results from analysis of the star tracker software are presented in Chapter 4 using tests with computer generated images. Chapter 5 details the hardware selected for the star tracker as well as an initial demonstration of the hardware capabilities with the star tracker software. Finally, Chapter 6 provides a conclusion and an outline for future work on the design.

Chapter 2

Algorithms

The process of star tracking consists of three main steps: centroiding, star identification, and attitude determination. Centroiding takes the image from the camera and determines the coordinates of light sources in the image plane, which can then be converted to unit vectors in the tracker coordinate frame. Star identification is the crux of the star tracker. The unit vectors in the tracker frame are analyzed and compared to a star catalog to determine which stars are in the image frame and consequently provide unit vectors in the inertial reference frame. Finally, the list of unit vectors in both the tracker and inertial frame are run through a vector-based algorithm to determine the attitude of the star tracker in the inertial frame. The attitude can be output in various formats, among most common are quaternions, Euler angles and direction cosine matrices (DCMs).

2.1 Centroiding

The first step for any star tracker is to determine location of the stars in the image plane. If focused star images are recorded, the light from each star would fall on only one or two of pixels and would likely saturate these pixels, resulting in pixel-level accuracy. Such an image is shown in Fig. 2.1.

Most star trackers therefore analyze an intentionally defocused, i.e, blurry, image, such as in Fig. 2.2, in order to spread the photons over more pixels and allow a centroiding algorithm to yield subpixel-level accuracy [14].

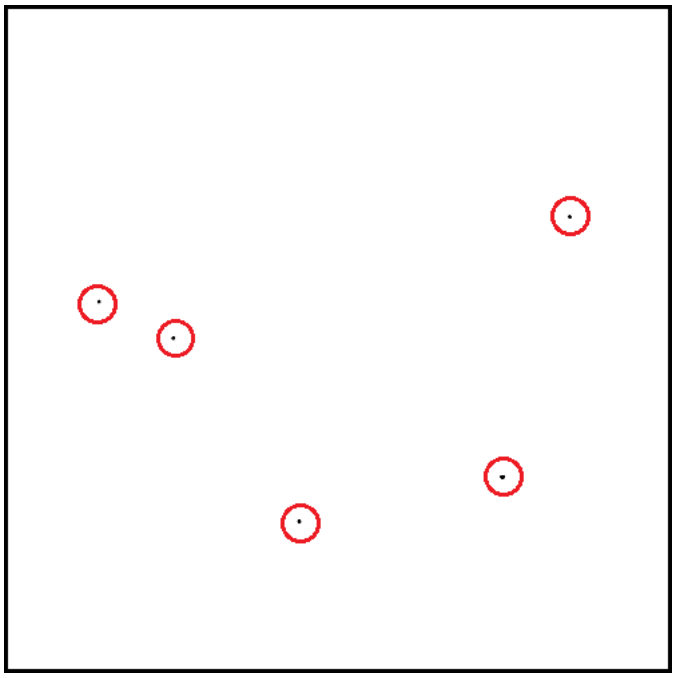

Figure 2.1: Star image without defocusing (negative).

After the defocused image is recorded, the centroid of the star is found much like the centroid of an array of point masses, with a couple differences. First, light intensity is used instead of mass. And second, the light intensity is usually normalized by the pixels around the star in order to filter out glare or background noise. The resulting output from the centroiding algorithm is a series of two-dimensional coordinates in the image plane with the origin at the image center. This coordinate system allows the coordinates of the stars to be easily converted to unit vectors in a later step.

The following is the centroiding algorithm used for this star tracker. It was adapted from the method presented by Liebe [14]. The algorithm requires the specification of the light intensity threshold I_{thresh} and the region-of-interest (ROI) size a_{ROI} in pixels. These values can be adjusted to tune the performance of the centroiding algorithm. For example,

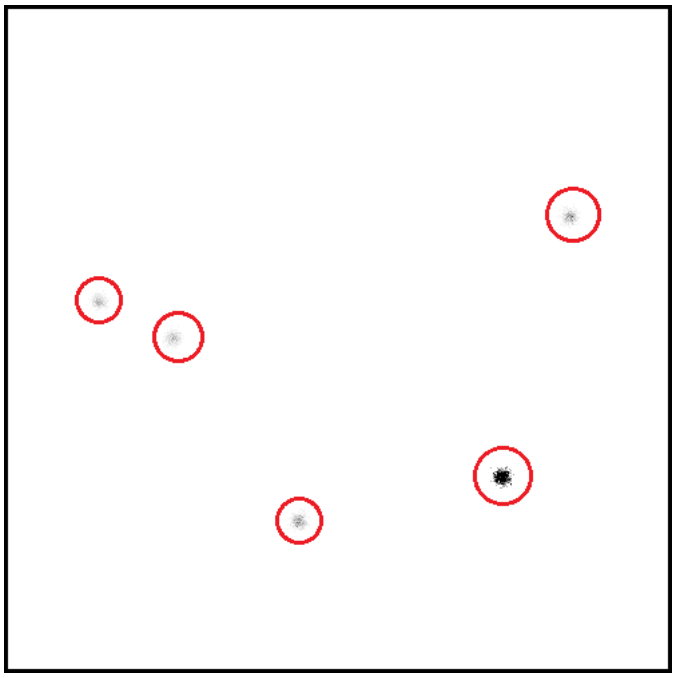

Figure 2.2: Star image with focusing (negative).

a higher I_{thresh} value is more robust to noise but might miss some actual stars in the image. Similarly, a large a_{ROI} value means a more accurate centroiding value but might read one star where there are actually two in close proximity. Note that a_{ROI} must be odd for the the algorithm to function properly.

The centroid algorithm is listed in the following steps:

1. For a pixel at image coordinate (x, y) with intensity value $I(x, y) > I_{thresh}$, the ROI is defined as the square of pixels with side length a_{ROI} and bottom-left corner at

(x_{start},y_{start}), given by Eqs. 2.1 and 2.2.

$$x_{start} = x - \frac{a_{ROI}-1}{2} \tag{2.1}$$

$$y_{start} = y - \frac{a_{ROI}-1}{2} \tag{2.2}$$

$$x_{end} = x_{start} + a_{ROI} \tag{2.3}$$

$$y_{end} = y_{start} + a_{ROI} \tag{2.4}$$

2. If $x_{start} < 0$ or $y_{start} < 0$, discard the pixel and return to step 1 with the next pixel.

3. Find the average intensity value of the border pixels I_{border}, given by Eq. 2.5 and shown for an ROI with $a_{ROI} = 7$ in Fig. 2.3

$$I_{bottom} = \sum_{i=x_start}^{x_{end}-1} I(i, y_{start}) \tag{2.5a}$$

$$I_{top} = \sum_{i=x_start+1}^{x_{end}} I(i, y_{end}) \tag{2.5b}$$

$$I_{left} = \sum_{j=y_start}^{y_{end}-1} I(x_{start}, j) \tag{2.5c}$$

$$I_{right} = \sum_{j=y_start+1}^{y_{end}} I(x_{end}, j) \tag{2.5d}$$

$$I_{border} = \frac{I_{top} + I_{bottom} + I_{left} + I_{right}}{4(a_{ROI}-1)} \tag{2.5e}$$

4. Subtract I_{border} from $I(x,y)$ for all non-border pixels, yielding a normalized light intensity matrix $\tilde{I}$

$$\tilde{I}(x,y) = I(x,y) - I_{border} \tag{2.6}$$

5. Calculate the centroid location (x_{CM}, y_{CM}) using Eqs. 2.7, 2.8, and 2.9. The bright-

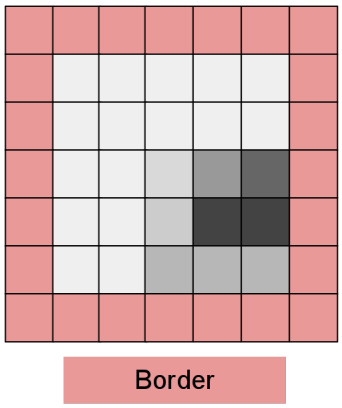

Figure 2.3: Border pixels for $a_{ROI} = 7$. Shading indicates the negative of a star in the image.

ness B in Eq. 2.7 is analogous to the total mass in an array of point masses.

$$B = \sum_{i=x_{start}+1}^{x_{end}-1} \sum_{j=y_{start}+1}^{y_{end}-1} \tilde{I}(i,j) \tag{2.7}$$

$$x_{CM} = \sum_{i=x_{start}+1}^{x_{end}-1} \sum_{j=y_{start}+1}^{y_{end}-1} \frac{i \times \tilde{I}(i,j)}{B} \tag{2.8}$$

$$y_{CM} = \sum_{i=x_{start}+1}^{x_{end}-1} \sum_{j=y_{start}+1}^{y_{end}-1} \frac{j \times \tilde{I}(i,j)}{B} \tag{2.9}$$

6. Once the centroid location (x_{CM}, y_{CM}) has been calculated for every pixel above the threshold, cycle through the centroid locations and average together any values that are clustered together. These values are assumed to represent the same star, though there is the possibility that there could be two stars in close proximity. This is a reasonable assumption, though, if the magnitude limit of the camera is high enough. The clustering process can be accomplished by checking each new centroid location against a list of already processed centroid locations. If the new location is within, for example, 5 pixels of a pre-existing location, average the two together. The output of this step is a list of averaged centroid coordinates, each of which should represent a

separate light source.

7. Convert the list of averaged centroid locations into unit vectors using the camera pixel size μ and camera focal length f using Eq. 2.10. The relevant geometry is seen in Fig. 2.4

$$\mathbf{u} = \frac{\begin{bmatrix} \mu x_{CM} & \mu y_{CM} & f \end{bmatrix}^T}{\left|\left| \begin{bmatrix} \mu x_{CM} & \mu y_{CM} & f \end{bmatrix} \right|\right|} \tag{2.10}$$

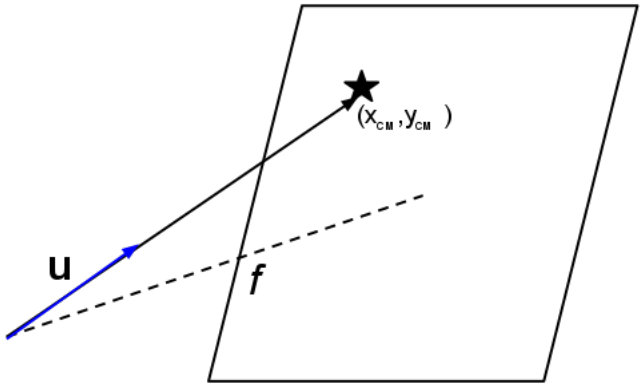

Figure 2.4: Unit vector to star.

2.2 Survey of star identification algorithms

The problem of star identification is well researched and numerous studies have been done into various methods [5, 15, 4, 16, 17]. All of the methods treated here utilize the unit vectors found from the centroiding step in Eq. 2.10, and some employ other information such as the apparent brightness of the stars. In addition, the star identification algorithms can be broken into two types: lost-in-space (LIS) and tracking. The former attempts to identify the stars in the image based strictly on the information in the image, while the

latter also uses *a priori* attitude data, such as the previous locations of identified stars or an attitude estimate from another sensor or dynamic filter. Most of the algorithms in this section can be used both as LIS and tracking algorithms.

All of these methods share a basic sequence in common:

1. Generate a list of possible geometries from a given star catalog.
2. Match the observed stars to the geometries in the catalog.
3. Assess the confidence of the identified stars and discard any false results.

2.2.1 Pyramid star identification technique

As opposed to the traditional triangle star identification techniques, Mortari [4] developed a related method based on pyramids. Mortari's method is given as follows:

1. Create a k-vector of possible star pairs. Instead of a standard list of geometries which must be searched linearly, Mortari employs a k-vector searchless algorithm. The k-vector is a sorted list of angles between stars, allowing the potential star pairs to be identified using a calculated factor instead of a linear or other search method [17].
2. Begin the star identification process by scanning the observed star for a unique triangle. Instead of a sequence which simply runs though every combination starting with the first three stars (i.e., 1-2-3, 1-2-4, 1-2-5, etc.), a prioritized sequence is used which cycles through all of the stars more rapidly (1-2-3, 2-3-4, 3-4-5, etc.) Also, Mortari defines uniqueness using a formula which outputs a probability that the triangle is mismatched. If that probability is too high, the triangle is thrown out.

3. Scan the remaining stars for a star which forms additional triangles with the three triangle stars. If no such star can be found, return to step 2 and test the next triangle in the sequence.

4. Once a high-confidence combination of 4 stars has been found, the pyramid which gives the technique its name is shown in Fig. 2.5. The remaining observed stars can be identified or thrown out as noise, again using the confidence formula mentioned before.

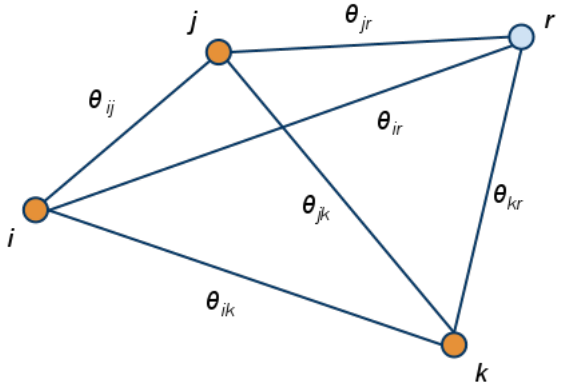

Figure 2.5: Star pyramid with candidate stars i, j, k and reference star r. [4]

2.2.2 Voting method

Similar to the pyramid star identification technique, the voting method uses additional star information to improve the accuracy of the identification. However, this method is not limited to just one additional star. The voting method [16] uses information from every star in the image to democratically assess the identity of each star and the method is described below.

1. The catalog generated for the voting method is a simple set of possible star pairs

and their angular distances. If desired, the k-vector approach [17] described for the pyramid method could be employed.

2. For each star pair in the image, calculate the angular distance. Run through the catalog and add the identities of any star pair whose distance lies within a certain tolerance of the observed star to a list for each star. Both identities are added to both lists, since both identities are possible for both candidate stars.

3. Once all of the star pairs have been analyzed, select the identity for each star as the catalog star which received the most votes.

4. Verify the accuracy of the identified star pairs. For each identified star pair, find the angular distance using the catalog. If that distance is within a given tolerance, those stars each get votes.

The output of the algorithm is the unit vectors of the stars that received votes greater than a certain value, usually the maximum number of votes any star received less 1. If that value is 0 or less, it is likely a failed identification.

2.2.3 Planar triangle star identification technique

Cole [15] presents a different approach to star identification. While this method still uses the premise of star triangles, Cole analyzes the triangles themselves, pattern matching using characteristics of the entire triangle rather than each side as is done in the two previously mentioned techniques. The algorithm is outlined as follows:

1. Construct a list of possible star triangles from catalog stars p, q, and r, as well as their areas and polar moments of inertia. The polar moment, given in Eq. 2.13 as J, predicts the resistance to torsion of a planar figure about its center of mass in the

z-direction. All three stars must satisfy the field-of-view and magnitude requirements, and the area and polar moment for each triangle can be found using Eqs. 2.11, 2.12, Heron's formula, and Eq. 2.13. Repeat this step for every potential combination of three catalog stars.

$$s = \frac{1}{2}(a + b + c) \tag{2.11a}$$

$$a = ||\mathbf{u}_p - \mathbf{u}_q|| \tag{2.11b}$$

$$b = ||\mathbf{u}_q - \mathbf{u}_r|| \tag{2.11c}$$

$$c = ||\mathbf{u}_p - \mathbf{u}_r|| \tag{2.11d}$$

$$A = \sqrt{s(s-a)(s-b)(s-c)} \tag{2.12}$$

$$J = A\frac{(a^2 + b^2 + c^2)}{36} \tag{2.13}$$

2. Begin the star identification process by selecting three stars and finding area and polar moment of the planar triangle that they form using Eqs. 2.12 and 2.13 again. In addition, calculate the variances of the area and polar moment. The process to find the variances can be found in [15]; it not repeated here. Using a k-vector or other search method, find all of the triangles from the list formed in step 1 whose areas and polar moments fall within a standard deviation of the observed triangle. If only one triangle meets these criteria, proceed to step 4.

3. If more than one catalog triangle meets the area and polar moment criteria, select another star from the image to identify and see how many stars overlap the two lists. If only two, the triangle is identified. If not, the solution is thrown out and the algorithm returns to step 3 with the next combination of three stars.

4. Once the three stars have been identified, the remaining stars in the image can identified if desired by using the same pivoting process described in step 4. Otherwise, proceed to attitude determination.

2.2.4 Grid algorithm

Padgett and Kreutz-Delgado [5] present a method called the grid algorithm. Instead of reading a series of stars and identifying a series of relative positions, as is done in the first two methods, the grid algorithm matches a catalog of patterns to the observed stars. To accomplish this, the field of view of the camera is divided into a grid and a matrix is formed with zeroes and ones, depending on whether a star exists in each grid element.

1. Construct a list of patterns for each star in the catalog. For each star in the catalog, rotate the other catalog stars so that the chosen star lies along the positive z-axis in the camera frame. If another catalog star is outside of a buffer radius but within the FOV of the camera, place that star on a fictitious image and change the value for that cell from zero to one.

2. Once step 1 is accomplished, there should be a matrix of zeroes and ones for each star in the catalog.

3. To perform the actual star identification, sort the identified stars in the image by brightness. Starting with the brightest star, translate the locations of the other stars so that the brightest star lies in the center of the image.

4. Apply the same grid size from step 1 and form a matrix of zeroes and ones based on whether or not a star exists in each cell.

5. Compare that matrix to the patterns in the catalog. and record the pattern with the most non-zero matrix values that the image and catalog patterns agree upon and how many values agreed.

6. Continue this process for each star in the image and select as the reference the star which had the pattern with the most matches.

7. Finally, verify the distances between the tentatively identified stars and the reference star. If enough distances agree, return the identified stars.

The grid method can be somewhat challenging to understand. Fig. 2.6, reproduced from Padgett [5], provides a better visual representation of the steps.

2.2.5 Comparison of methods

Each of the four methods described above have distinct advantages and disadvantages. Before comparing their relative merit, it is important to define what characteristics are important to a CubeSat. First, the algorithm must be robust, that is, tolerant of noise and errors in the image. CubeSats are small and must be lightweight, which means radiation shielding is usually left off satellites which will fly in low Earth orbit (LEO). Thus, there is a greater possibility of errors on images due to stray radiation or cosmic rays. In addition, satellites in LEO have a greater chance of observing other satellites and space debris and mistaking them for stars. Also, lower resolution cameras are preferable on CubeSats. They are smaller and consume less power, and they reduce the burden on the command and data handling (CDH) system. Another concern for CDH is database storage and access. The fewer calls to the catalog and the smaller the catalog is, the more efficiently both the star tracker and CDH system can perform. Despite these considerations, the CubeSat

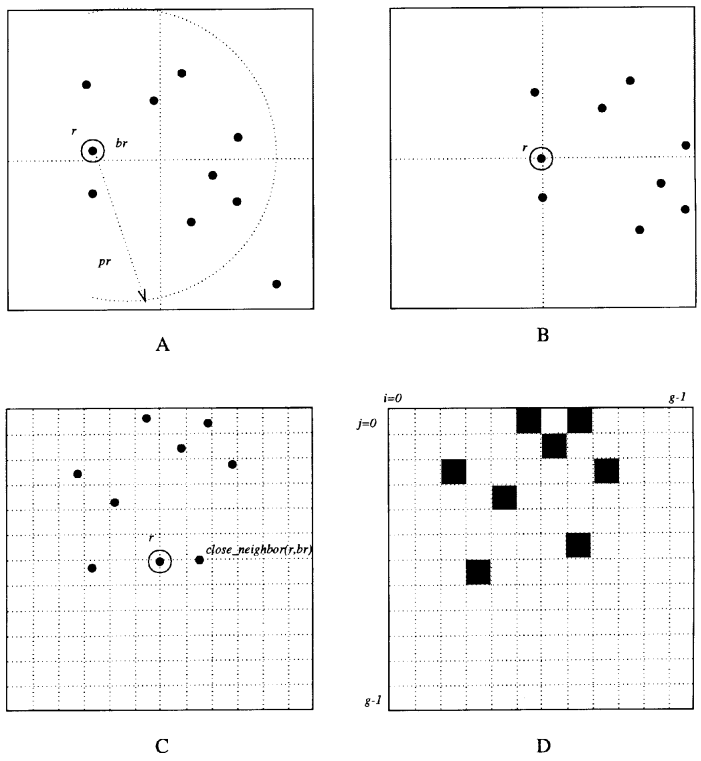

Figure 2.6: Description of grid algorithm steps. A: Brightest star identified. B: Image translated with respect to brightest star. C: Grid aligned with nearest neighbor. D: Bit pattern [5].

star tracker must provide a significant improvement on the accuracy of other instruments. Volume aboard a CubeSat is at a premium, and a star tracker must justify its presence.

The pyramid star identification method can be considered quite robust. Since it must match a total of four stars, any spikes or false stars would cause a potential match to be rejected. However, a rejected match means the algorithm must start with a new combination of stars, which takes processing time. The voting method is extremely robust to false stars. In tests described in Kolomenkin et al. [16], the voting method retained the correct number of correctly identified stars until the number of false stars was as much as three times the number of actual stars, and it accomplishes this without restarting the

algorithm. The planar triangle technique is less robust. Requiring a match of both area and polar moment helps remove ambiguity, but a poorly placed false star could cause the method to return a false positive. Finally, the grid method is quite robust. A false star simply adds an additional 1 in the matrix, but it would not significantly impact the matching accuracy unless there were many false stars.

The lower resolution requirement does not significantly affect any of the first three methods. Using defocusing and centroiding, subpixel accuracy should be attained for any one of these methods. The same cannot be said for the grid method. Lower resolution means that the grid must tighten, and that increases the likelihood of stars on a boundary or actual stars which might be off by one or two cells.

As far as catalog requirements, both the pyramid algorithm and voting method are similar. The catalogs are the same size, since both use star pairs, and both should use about the same number of calls to the catalog, though the demand might be slightly higher for the voting method if many stars are observed in the image. The planar triangle method requires two large catalogs for area and polar moment, but should not have significantly more calls to that catalog. The grid method can also have a large catalog, since there must by a bit matrix for each star in the catalog.

Weighing the three criteria stated above, the pyramid and voting algorithms both stand out as the best fits. Ultimately, the voting method was chosen because of its high robustness, which should serve as an asset given the uncertain environment facing a CubeSat star tracker.

2.3 Voting method

The selected method for this star-tracker was the voting method outlined by Kolomenkin et al. [16], which was presented in section 2.2.2. The algorithm is now presented in greater detail for implementation in software. An block diagram is shown in Fig. 2.7.

2.3.1 Catalog generation

The catalog generation in the voting method is very similar to the same process in other techniques. Using a given minimum light intensity that will be considered a star, and the field-of-view (FOV) of the lens, a list of possible star pairs is generated along with their angular distances. This step is done before installation and the catalog is stored in memory aboard the spacecraft. This catalog can be generated in the following way:

1. Convert star positions to unit vectors. Most star catalogs record the positions of stars in inertial right ascension-declination coordinates. While useful for astronomers, for star identification the unit vectors must be found to those stars in the inertial frame. Eq. 2.14 gives the unit vector in terms of right ascension α and declination δ.

$$\mathbf{u} = \begin{bmatrix} \cos\alpha\cos\delta \\ \sin\alpha\cos\delta \\ \sin\delta \end{bmatrix} \tag{2.14}$$

2. For each pair of stars with identification numbers a and b, verify that the pair meets the conditions on magnitude and angular distance given in Eqs. 2.15, 2.16 and 2.17. Note that the brighter an object, the lower its magnitude.

$$m_a \leq m_{max} \tag{2.15}$$

$$m_b \leq m_{max} \tag{2.16}$$

$$\mathbf{u}_a^T \mathbf{u}_b \geq \cos\theta_{FOV} \tag{2.17}$$

3. If the above conditions are met, record the star identification numbers as well as the value of the scalar $\mathbf{u}_a^T \mathbf{u}_b$ in a file for later access.

Note that the value of $\mathbf{u}_a^T \mathbf{u}_b$ is equal to the cosine of the angle between the two unit vectors. Checking that this value is greater than the cosine of the field-of-view angle is equivalent to stating that the angular distance is less than the field-of-view angle, ensuring that the two stars can both be seen by the camera at the same time.

2.3.2 Candidate matching

This step begins the process which takes place with each star tracking operation. At this point, a list of unit vectors of observed light sources is available in the camera-fixed frame. These will henceforth be called "candidate stars," since there is no way of knowing whether they are real stars or false stars, like planets, other satellites, or other noise sources in the image.

1. For the pair of candidate stars i and j, calculate the cosine of the angular distance between them d_{ij} using Eq. 2.18.

$$d_{ij} = \mathbf{u}_i^T \mathbf{u}_j \tag{2.18}$$

2. Find every pair of stars p and q in the catalog whose angular distance d_{pq} satisfies Eq. 2.19 for a given tolerance ϵ. Note that the k-vector approach [17] described earlier would suffice here.

$$d_{ij} - \epsilon \leq d_{pq} \leq d_{ij} + \epsilon \tag{2.19}$$

3. For each possible star pair found in step 2, add the identification number for both catalog stars p and q to arrays for candidate stars i and j. Note that both identification

numbers are added to both lists since either catalog identity is a possibility for both stars.

4. Once all possible pairs of candidate stars have been processed, assign each candidate star the catalog star that received the most votes in its array.

At this point, each candidate light source should have a likely catalog star assigned to it. However, there is still no way of knowing if any of these are false stars that could cause an erroneous attitude solution.

2.3.3 Verification and final result

The verification step removes false candidate stars. Another round of voting is performed, after which a final matched list of observed and candidate stars will be produced.

1. For the pair of candidate stars i and j, find the cosine of the angle between their matched catalog stars r_{ij} using Eq. 2.20, where $\mathbf{v}_i$ is the unit vector of the catalog star matched to candidate star i.

$$r_{ij} = \mathbf{v}_i^T \mathbf{v}_j \tag{2.20}$$

2. If Eq. 2.21 is satisfied, add one vote each to the lists for candidate stars i and j.

$$d_{ij} - \epsilon \leq r_{ij} \leq d_{ij} + \epsilon \tag{2.21}$$

3. After every candidate star pair has been processed, find the threshold T for real stars by applying Eq. 2.22.

$$T = \max(\text{votes}(i)) - 1 \tag{2.22}$$

4. Any candidate star with more votes than threshold T is passed out of the star identification algorithm as a real star.

The power of the voting method lies in this step. Real stars have numbers of votes in this step clustered together and above the threshold T, while any false stars will not receive more than one or two votes, removing them in this step. Assuming at least three stars have been positively identified, the two lists of matched candidate and catalog stars are now passed to the attitude determination algorithm.

2.4 Attitude determination

Once the candidate stars have been matched to stars from the catalog, two lists of unit vectors exist. One is in the camera frame, the other in the inertial frame. To find the rotation between them, and ultimately, the attitude of the satellite, an attitude determination method must be applied. There are several well-known methods which return quaternions, including Davenport's q-method [18] and QUEST [19]. However, for the purposes of this research, a direction cosine matrix was desired, so the singular value decomposition method developed by Markley [20] is used. This method is reproduced below.

1. Calculate the matrix $\mathbf{B}$ using Eq. 2.23 with the n observed stars from the star identification algorithm, where $\mathbf{b}_i$ and $\mathbf{r}_i$ for real star i are the unit vectors of the candidate star and catalog star, respectively.

$$\mathbf{B} = \sum_{i=1}^{n} \mathbf{b}_i \mathbf{r}_i^T \tag{2.23}$$

2. Find the singular value decomposition of matrix $\mathbf{B}$; that is, the orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ and the diagonal matrix of singular values $\mathbf{S}$ which satisfy Eq. 2.24

$$\mathbf{B} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{2.24}$$

3. Define the proper orthogonal matrices $\mathbf{U}_+$ and $\mathbf{V}_+$ using Eqs. 2.25 and 2.26

$$\mathbf{U}_+ = \mathbf{U}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det \mathbf{U} \end{bmatrix} \tag{2.25}$$

$$\mathbf{V}_+ = \mathbf{V}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det \mathbf{V} \end{bmatrix} \tag{2.26}$$

4. The direction cosine matrix $\mathbf{A}$ can now be found using Eq. 2.27

$$\mathbf{A} = \mathbf{U}_+\mathbf{V}_+^T \tag{2.27}$$

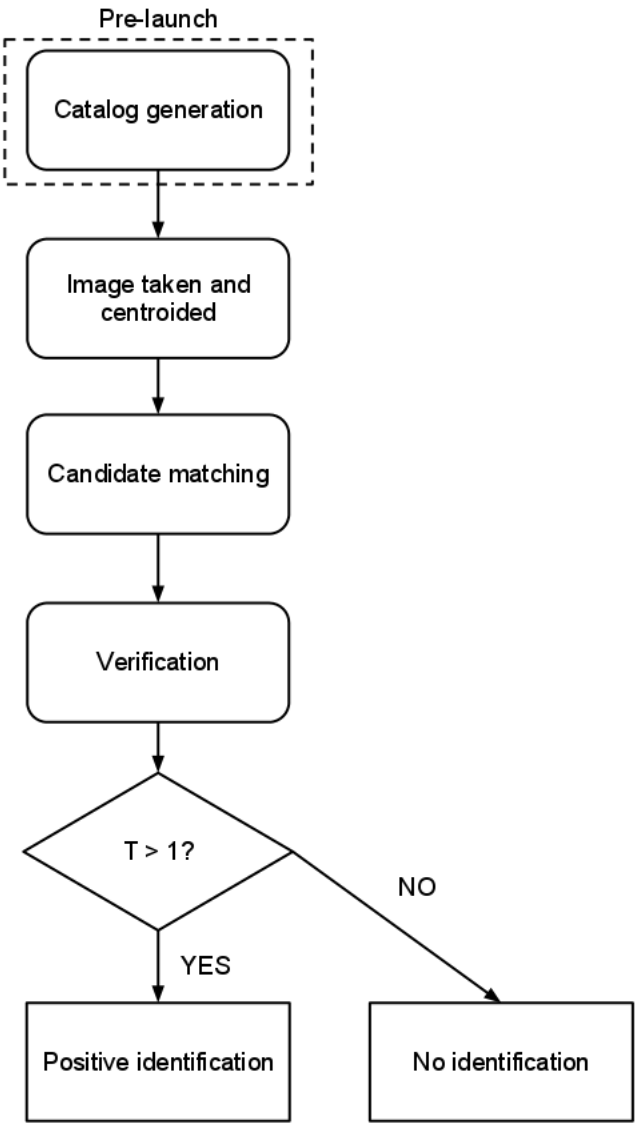

Figure 2.7: Voting method block diagram

Chapter 3

Software Architecture

Selecting and adapting the appropriate algorithms are the crux of any star tracker, but they must be actually implemented in software and hardware. This chapter deals with the former topic, while Chapter 5 treats the latter topic.

3.1 Software flow

A block diagram of the software functionality is shown in Fig. 3.1. Before launch, the function *catinit* is run to generate the list of possible star pairs and their interior angles. This file is saved aboard the spacecraft. When the star tracker function is called by the on-board computer, *catinit* is run again to load the catalog into memory. The function *loadimg* takes a picture using the on-board camera and outputs the resulting light intensity matrix. That information is passed onto *centroid*, which finds the light sources in the image and outputs their coordinates in the image plane. The function *uvec* uses the camera geometry to convert these image plane coordinates into unit vectors in the star tracker coordinate frame. Those unit vectors are given to *starid*, which identifies the stars and outputs their unit vectors in both the star tracker and inertial coordinate frames. Finally, those lists are given to *attdet*, which finds the DCM from the inertial to the star tracker coordinate frame and returns that information to the on-board computer.

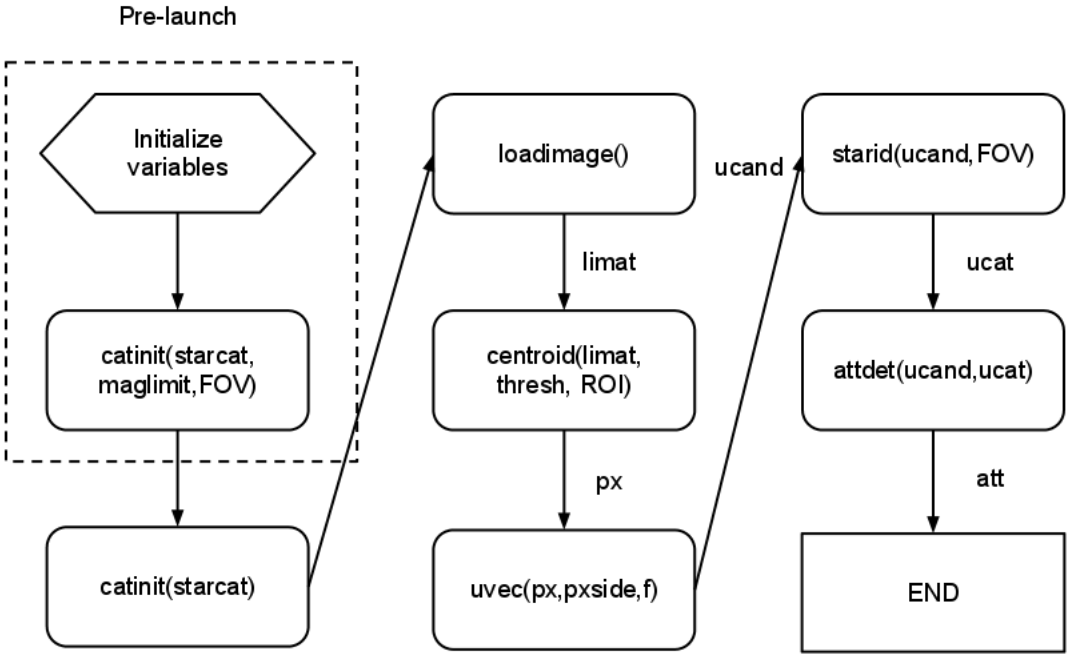

Figure 3.1: Star tracking flow diagram.

3.2 Component functions

The star tracker program consists of a number of functions whose purpose roughly mirrors the three major star tracker steps outlined in Chapter 2: centroiding, star identification, and attitude determination. In some cases, these steps have been further broken up into sub-steps, depending on their place within the software flow or the constraints of the programming language. For this work, the programming was done in MATLAB, though eventually these functions will be reproduced in either C or C++ for real-time execution in the embedded system.

3.2.1 Function *catinit*

The function *catinit* has the purpose of generating the possible star pairs and their interior angles based on magnitude and field-of-view information and returning that data to the main star tracker program. It also saves this data to a tab-delimited text file. If this text file has already been generated, *catinit* just reads the information into the main star tracker program. A flow diagram of *catinit* is shown in Fig. 3.2.

Inputs

The function *catinit* takes either one or three inputs. It always takes the name of a text file as an input. Optionally, the field-of-view and magnitude limit of the camera also can be given as inputs.

Outputs

The function *catinit* returns two outputs: the table of the possible star pairs and their angular distances as well as the list of unit vectors from the star catalog. If a field-of-view value and magnitude limit are also given as inputs, *catinit* will save the generated table as a text file with the given file name; if not, *catinit* will load the previously generated file given by the file name.

3.2.2 Function *loadimg*

The function *loadimg* reads an image, either from a file or attached camera and returns the accompanying matrix of light intensity values. A flow diagram of *loadimg* is shown in Fig. 3.3.

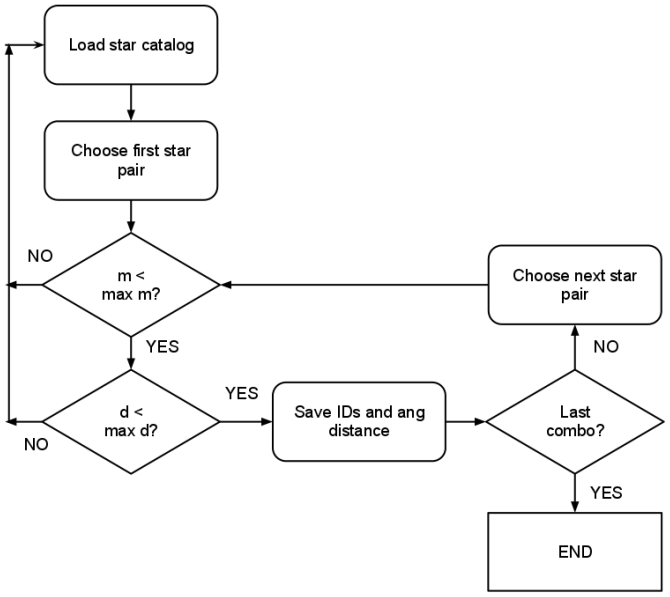

Figure 3.2: Function *catinit* flow diagram.

Inputs

The function *loadimg* takes either zero or one input. With no input, it calls the attached camera and takes an image. With one input, the filename of an image, it loads that image.

Outputs

The function *loadimg* returns one output. That is the matrix of integer light intensity values from 0 to 255 associated with the given image.

3.2.3 Function *centroid*

The function *centroid* takes a light intensity matrix of a star field and returns a list of coordinates representing the centers of the stars in the image. A flow diagram of *centroid*

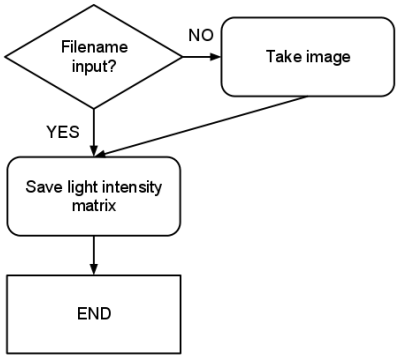

Figure 3.3: Function *loadimg* flow diagram.

is shown in Fig. 3.4.

Inputs

The function *centroid* takes four inputs. These are the light intensity matrix itself, a threshold value between 0 and 255, and the length of the region-of-interest in pixels in the x and y directions.

Outputs

The function *centroid* returns one output. That is the list of image coordinates for the star center in pixels.

3.2.4 Function *uvec*

The function *uvec* takes a list of image coordinates and information about the camera geometry and returns unit vectors to those coordinates in the camera-centered coordinate frame based on a pinhole model of the camera.

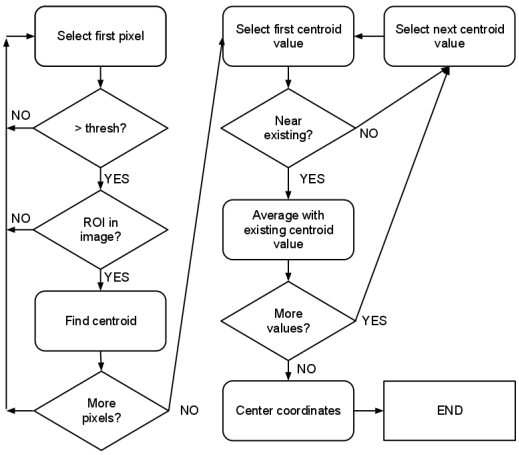

Figure 3.4: Function *centroid* flow diagram.

Inputs

The function *uvec* takes three inputs. The function *uvec* needs the list of image coordinates and the focal length of the camera as well as the size of the pixels on the camera sensor, which are constant for a given hardware setup.

Outputs

The function *uvec* returns one output, the list of unit vectors in the camera-centered coordinate frame.

3.2.5 Function *starid*

The function *starid* contains the programming that performs the geometric voting method described in Section 2.3. The function will take the list of candidate star unit vectors, camera geometry information, and the list of angular distances and return a paired list of unit vectors in the camera and inertial frames. A flow diagram of *catinit* is shown in Fig. 3.5.

Inputs

The function *starid* takes five inputs. The list of unit vectors generated by *uvec* and the list of angular distances read by *starid* are the most obvious data. In addition, the field-of-view is also input to perform a sanity check on the data. Also required is the pixel size to help determine the tolerance and the list of unit vectors of the catalog stars to perform the verification step.

Outputs

The function *starid* returns three outputs. The matched lists of candidate and catalog star unit vectors are returned. Additionally, the list of star identification numbers for the matched catalog stars is returned, which can be used as diagnostic information.

3.2.6 Function *adet*

The function *adet* performs the singular value decomposition method developed by Markley [20] on the lists of unit vectors generated by *starid* and returns a direction cosine matrix of the attitude of the camera in the inertial frame.

Inputs

The function *adet* takes two inputs. They are the two lists of unit vectors in the camera-centered and inertial frames, respectively.

Outputs

The function *adet* returns one output: the direction cosine matrix from the inertial to the camera-centered frame.

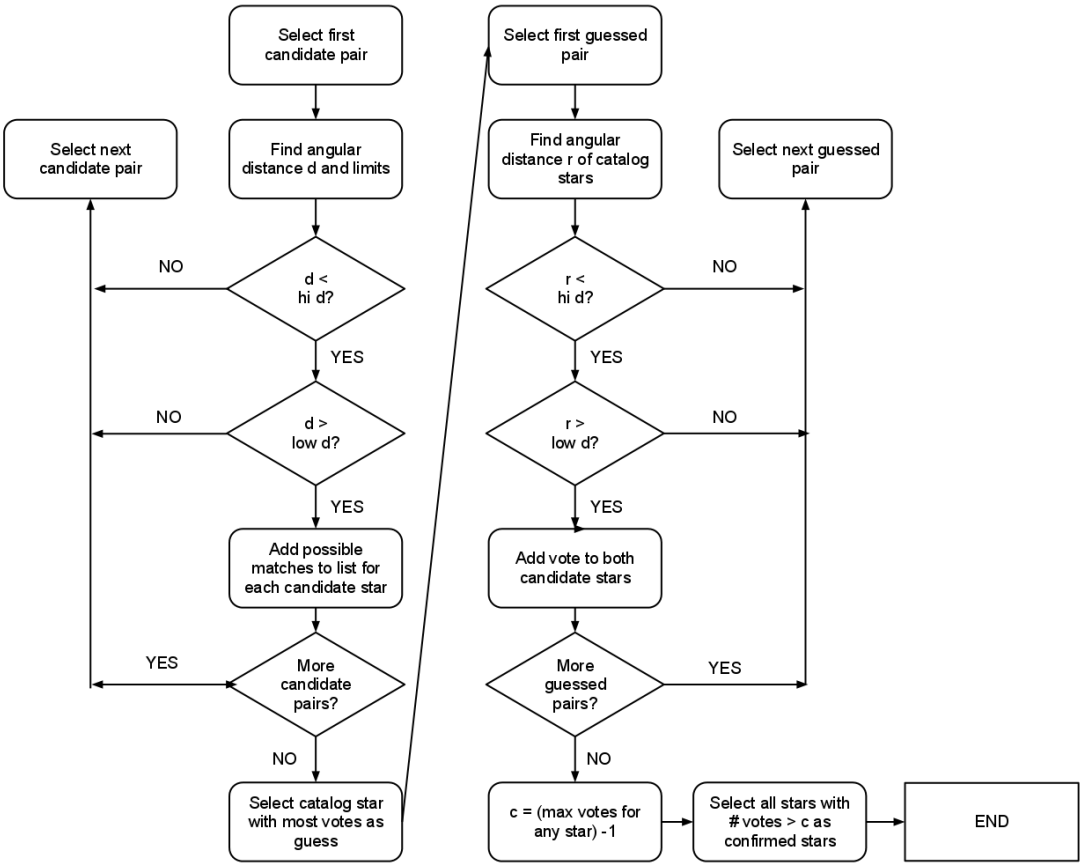

Figure 3.5: Function *starid* flow diagram.

Chapter 4

Software Results

The star-tracking algorithm was tested using a computer generated star field. The test involved a series of star images generated using a virtual camera that were generated with a set of random attitudes. These images were then run through the star tracking program and the calculated attitude compared with the true attitude to find the errors.

4.1 Setup

The setup for the analysis consisted of three parts: picking the random attitude, generating the images, and running the star tracker software.

4.1.1 Random attitude

Randomness was introduced by picking three random angles between -180 degrees and 180 degrees. Since MATLAB's random number generator provides a matrix of values between 0 and 1, Eq. 4.1 was used to convert the 3×1 output α_1, α_2 and α_3 into three angles θ_1, θ_2 and θ_3 the desired range.

$$\begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \end{bmatrix} = \left(\begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \end{bmatrix} - \begin{bmatrix} .5 & .5 & .5 \end{bmatrix} \right) \times 180^\circ \tag{4.1}$$

From there, a direction cosine matrix was formed by performing a 3-2-1 rotation about angles θ_1, θ_2 and θ_3, resulting in a rotation matrix from the inertial to the camera frame

described in Eq. 4.2.

$$\mathbf{R}_I^C = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_3 & \sin\theta_3 \\ 0 & -\sin\theta_3 & \cos\theta_3 \end{bmatrix} \tag{4.2}$$

This was done a total of 200 times, resulting in a series of 200 rotation matrices.

4.1.2 Image generation

Once the attitudes were selected, images at each attitude had to be generated. A virtual camera with parameters listed in Table 4.1 and a virtual charged-coupled device (CCD) with parameters listed in Table were selected to create the virtual image, though any reasonable virtual camera and sensor would be acceptable.

Aperture diameter, d (cm)	2.5
Field-of-view (°)	20
Focal length, f (mm)	66.8
Transmittance, τ	1

Table 4.1: Virtual camera parameters.

Resolution (pixels)	1024×1024
Pixel size (μm)	23×23
Quantum efficiency	0.2
Dynamic range (dB)	69
Spectral range, $\lambda(\mu$m)	0.6
Saturation limit, l_{sat} (photons)	135000

Table 4.2: Virtual CCD parameters.

The image generation process then proceeds as follows:

1. For each attitude, the unit vector along the boresight direction $\mathbf{u}_{bore}$ was found in the inertial coordinate frame by Eq. 4.3.

$$\mathbf{u}_{bore} = (\mathbf{R}_I^C)^T \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \tag{4.3}$$

2. Each star in the catalog is then checked to see if it lies in the field of view of the camera using the condition in Eq. 4.4.

$$\mathbf{u}_{bore}^{T}\mathbf{u}_{star} > \cos\frac{\theta_{FOV}}{2} \tag{4.4}$$

3. If the star lies in the field of view, its unit vector is converted to the camera frame by Eq. 4.5.

$$\mathbf{u}_{star,C} = \mathbf{R}_{I}^{C}\mathbf{u}_{star} \tag{4.5}$$

4. The coordinates of the star on the image plane (μ, ν) can then be found using Eqs. 4.6 and 4.7. $(u_{star,C})_x$ is the x-value of the star's unit vector in the camera frame and similarly for $(u_{star,C})_y$ and $(u_{star,C})_z$.

$$\mu = -f\frac{(u_{star,C})_x}{(u_{star,C})_z} \tag{4.6}$$

$$\nu = -f\frac{(u_{star,C})_y}{(u_{star,C})_z} \tag{4.7}$$

5. The image generation function requires how many points the stars photons will be divided over n_p and the blur factor b. For this simulation, $n_p = 500$ and $b = 3$. The exposure time t_{exp} also required. In order to simulate the effects of gain control that are available on a real camera, t_{exp} was set to 25 seconds to compensate for the lack of gain control on the simulated image.

6. The number of photons S the star generates can now be found using the camera and CCD parameters as well as the magnitude of the star m in Eq. 4.8

$$S = (t_{exp})\left(\frac{\pi}{4}d^2\right)(\tau)(\lambda)\left(10^{-0.4m}\right) \tag{4.8}$$

7. The star is now blurred by assigning each of the n_p points a random location (μ', ν') near the star's image location (μ, ν).

$$\begin{bmatrix} \mu' & \nu' \end{bmatrix}^T = \begin{bmatrix} \mu b\cdot\text{rand}(-.5,.5) & \nu b\cdot\text{rand}(-.5,.5) \end{bmatrix}^T \tag{4.9}$$

8. For each pixel in the light intensity matrix $\mathbf{L}$, if a point (μ', ν') lies within that image, add $\frac{S}{n_p}$ photons to that point.

9. Finally, normalize the light intensity matrix to $\tilde{\mathbf{L}}$ by the saturation limit l_{sat}.

$$\begin{cases} \tilde{\mathbf{L}}(i,j) = \frac{\mathbf{L}(i,j)}{l_{sat}}, & \frac{\mathbf{L}(i,j)}{l_{sat}} \leq 1 \\ \tilde{\mathbf{L}}(i,j) = 1 & \frac{\mathbf{L}(i,j)}{l_{sat}} > 1 \end{cases} \tag{4.10}$$

MATLAB can now use $\tilde{\mathbf{L}}$ to create an image in the desired format, which was a bitmap file for this analysis.

4.2 Analysis

These images were then processed by the star tracker software, which output an attitude. The real and calculated attitudes were compared to find the error in the star tracker measurement. The error about each axis was then calculated as shown in Eqs. 4.11, 4.12, 4.13, and 4.14.

$$\mathbf{\Theta} = \mathbf{R}_{calc}^{-1}\mathbf{R}_{actual} = \begin{bmatrix} \approx 1 & -\phi_z & -\phi_y \\ \phi_z & \approx 1 & -\phi_x \\ \phi_y & \phi_x & \approx 1 \end{bmatrix} \tag{4.11}$$

$$\epsilon_x = 90^\circ - \cos^{-1}\phi_x \tag{4.12}$$

$$\epsilon_y = 90^\circ - \cos^{-1}\phi_y \tag{4.13}$$

$$\epsilon_z = 90^\circ - \cos^{-1}\phi_z \tag{4.14}$$

The attitude error was found about each axis for each of the 200 generated images. The algorithm was considered to have failed if the error was greater than 100 arcseconds for any of the three axes. No physical errors were included in the virtual camera. Therefore, the error must come from the algorithm itself, and 100 arcseconds is a reasonable threshold for this experimental setup. Given these conditions, the star tracking algorithm found the attitude within acceptable error 191 of 200 times, or an accuracy of 95.5%

Looking first at the failures, only twice did the algorithm fail for an attitude for which it had a confident match. The match confidence is determined by the value T in Eq. 2.22. If $T \leq 0$, then the no star received any votes in the verification step, indicating bad matches. On orbit, this would be output as an error condition and the attitude would not be used. Only two reported attitude solutions were incorrect despite having $T > 0$. These cases present a problem, since they would be passed on as a correct result. After visual analysis of both failure cases, the simulated images have at least one star pair in very close proximity, which may have fooled the algorithm into thinking there was one star and causing the failure. Further refinement of the attitude suite algorithm could be use to counteract this case if necessary, but since it represents only 1% of the total test cases, the algorithm seems acceptable.

As for the 191 successful cases, the average performance can be seen in Table 4.3

Stars generated	Stars observed	T	ϵ_x (arcsec)	ϵ_y (arcsec)	ϵ_z (arcsec)
11.4607	10.1152	8.6230	10.6237	7.7998	6.4789

Table 4.3: Average successful case performance.

Theoretically, performance should improve the more stars that are observed. Fig. 4.1 shows the amount of cases for which each number of stars was observed, while Fig. 4.2 shows the error for each axis averaged over the number of stars observed.

The cases where fewer stars are observed do have the highest errors for the x- and y-axes, and nearly the highest for the z-axis. However, there appears to be no strong correlation downward as the number of stars observed is increased. This observation is verified by a root-sum-square analysis of the error across the three axes, shown in Fig. 4.3. The phenomenon could be related to the clustering error seen in the failure cases.

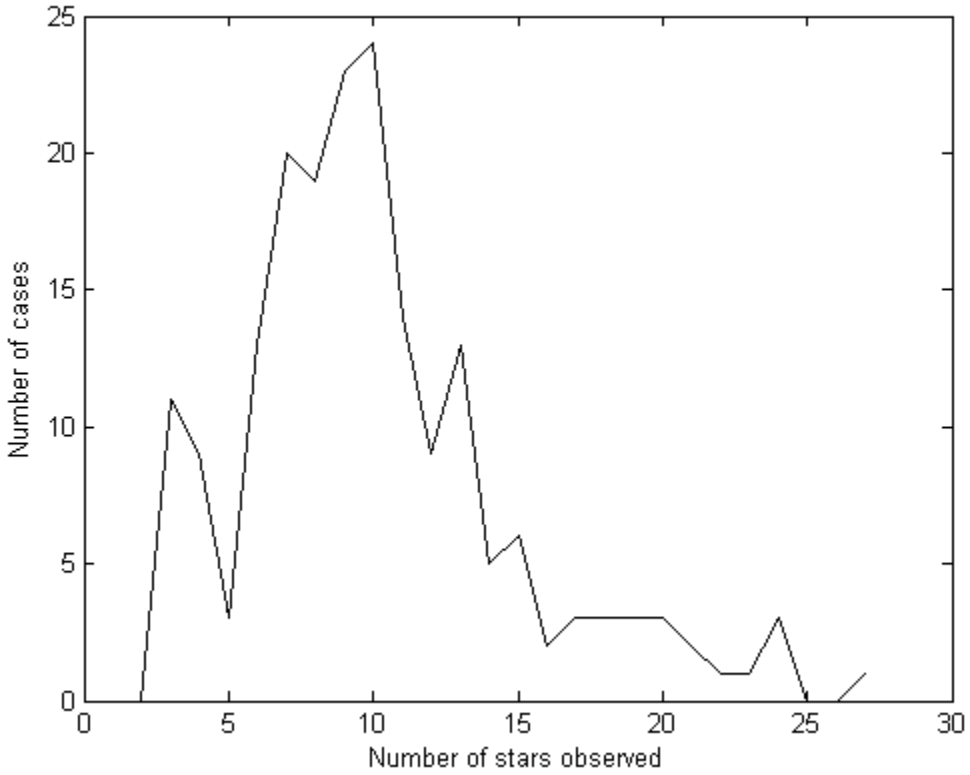

Figure 4.1: Number of cases for each number of stars observed.

4.3 Summary

The graphs from section 4.2 highlight some interesting results. First, even though all physical error has been removed from the test, the errors in the three axes are not zero. Looking through the process for potential error sources, the only likely contributor is the centroiding algorithm. Despite achieving subpixel accuracy, the centroiding function still has some slight error from the true location of the center of the star. Errors as small as 10 arcseconds should not concern most CubeSat developers looking to add star tracking to their satellites unless their mission requires very high accuracy.

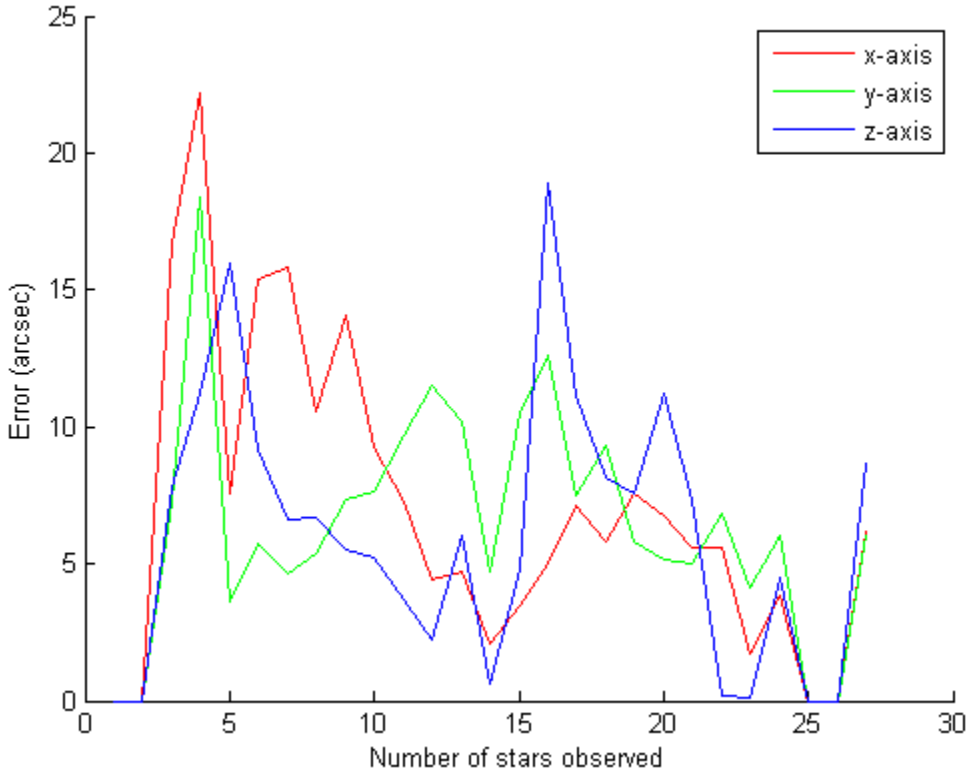

Figure 4.2: Error in each axis vs number of stars observed.

Another noteworthy figure is the average number of stars viewed, which was about 10. Even for a 20 degree FOV, this number is somewhat high and is the result of the brightness mask on the stars being magnitude 5.5. This choice was made to ensure that enough stars existed in each image for analysis. The purpose of the computer generated images was not to assess the real world performance of the star tracker, given all of the ideal assumptions that were made. Rather, it was to verify the functionality of the algorithm so further, more realistic tests can be performed.

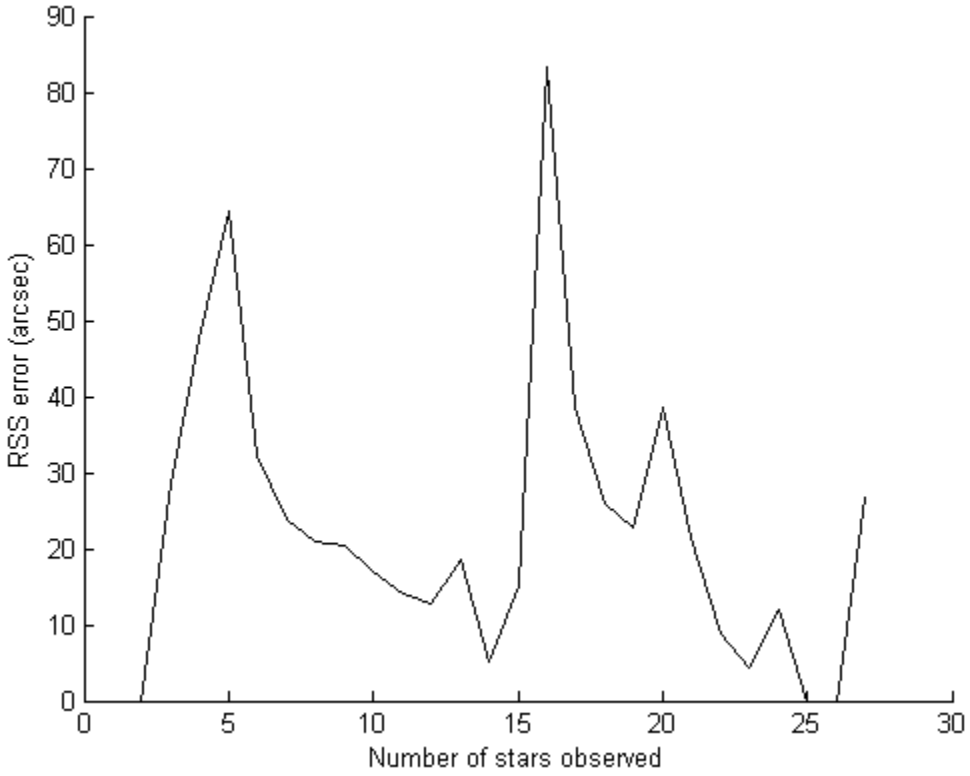

Figure 4.3: RSS error vs number of stars observed.

Chapter 5

Hardware Architecture

After verifying the performance of the software, the hardware implementation is the next step. This chapter outlines the hardware choices and their rationales, as well as giving the current state of hardware testing for this reserach.

5.1 Camera

The camera for the star tracker must be satisfactory in three main areas. The device must be small enough to fit the CubeSat form factor. It must also have enough resolution to provide an accurate star tracking solution. Finally, it must able to provide a stable and useful interface to the command and data handling system of the satellite. Three models were considered for the star tracker camera: the Aptina MT9P031 Demo Kit, the Matrix Vision mvBlueFOX-M105G, and the Matrix Vision mvBlueFOX-M121G. All three cameras employ a USB data and power connection.

5.1.1 Aptina MT9P031 Demo Kit

This camera model is a demo kit from the maker of the MT9P031 complementary metal-oxide-semiconductor (CMOS) sensor. This sensor was considered because it was previously used on a Sinclair Aerospace star tracker [21]. The Aptina demo kit employs a rolling shutter, which means that the sensor is exposed a slice at a time. This technique reduces the throughput needed for the sensor and contrasts with a full-frame shutter which

exposes the entire sensor all at once.

Figure 5.1: Aptina MT9P031 Demo Kit [6].

Resolution (px)	2592×1944
Pixel size (μm)	2.2×2.2
Maximum frame rate (fps)	14
Shutter type	Rolling
Dimensions (mm)	82.7×48.3×35.0

Table 5.1: Aptina MT9P031 Demo Kit specifications.

5.1.2 Matrix Vision mvBlueFOX-M105G

The mvBlueFOX-M121G is an industrial machine-vision camera from Matrix Vision. The M105G utilizes an Aptina MT9P031 CMOS sensor and connects to the satellite for both data and power using the USB 2.0 protocol.

5.1.3 Matrix Vision mvBlueFOX-M121G

The mvBlueFOX-M121G is an industrial machine-vision camera from Matrix Vision. The M121G utilizes a Sony ICX204AL charged-couple device (CCD) sensor and con-

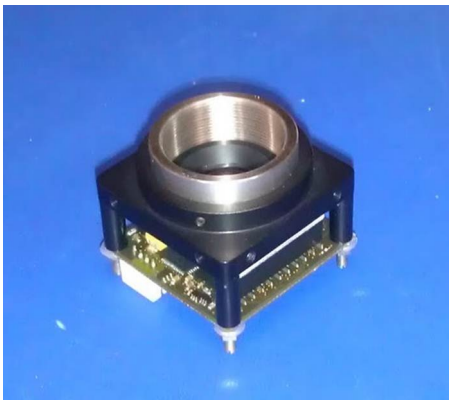

Figure 5.2: Matrix Vision mvBlueFOX-105G camera.

Resolution (px)	2592×1944
Pixel size (μm)	2.2×2.2
Maximum frame rate (fps)	5.8
Shutter type	Rolling
Dimensions (mm)	38.8×38.8×34

Table 5.2: mvBlueFOX-105G specifications. [1]

nects to the satellite for both data and power using the USB 2.0 protocol.

Resolution (px)	1024×768
Pixel size (μm)	4.65×4.65
Maximum frame rate (fps)	39
Shutter type	Full frame
Dimensions (mm)	38.8×38.8×34

Table 5.3: mvBlueFOX-121G specifications. [1]

5.1.4 Selection

All three cameras are acceptable for the CubeSat form factor, though the two Matrix Vision cameras are smaller and therefore better suited for a CubeSat. The Aptina and M105G share the same sensor, which has about 5 times the resolution and one quarter the pixel size. The drawback to the larger resolution is that both the Aptina and M105G utilize rolling shutters, which would distort an image if the satellite is slewing while the

Figure 5.3: Matrix Vision mvBlueFOX-121G camera.

image is taken. Therefore the M121G would be the better option if its resolution is fine enough. Eq. 5.1 gives an estimate of theoretical pixel accuracy based on pixel size μ and focal length f.

$$\theta_{theoretical} = \tan^{-1}\left(\frac{\mu}{f}\right) \tag{5.1}$$

Based on this equation, the theoretical pixel accuracy is 80 arcseconds for the M121G with a 12 mm focal length lens. Centroiding will improve this value by approximately 10 times, yielding a theoretical accuracy of 8 arcseconds. It is worth noting that both pixel size and focal length affect the theoretical accuracy. By increasing the focal length, the accuracy gets better. The tradeoff is a narrow FOV and therefore fewer observed stars. Table 5.4 shows this relationship.

$\theta_{theoretical}$ (arcseconds)	Focal length (mm)	FOV (degrees)
19.98	4.8	46.62
11.99	8	32.42
7.99	12	22.94
4.17	23	12.45
2.74	35	8.23

Table 5.4: Accuracy vs focal length and FOV for a 4.65 μm sensor.

Based on the error observed in analysis, the focal length can be adjusted to achieve the desired accuracy. In the case of this system, that accuracy is sub-arcminute.

In terms of interface, both Matrix Vision cameras have native Linux interfaces, while the Aptina Demo Kit is restricted to Windows only. Given the characteristics of the cameras, the M121G was deemed to be the best choice.

5.2 Lens

The lens for the star tracker has to meet two major requirements. The field-of-view must be large enough to view as many stars as needed for an accurate star tracking measurement. Also, the lens must be able to withstand the launch conditions while maintaining its ability to take images. The two lenses considered were the Xenoplan Compact C-Mount Lens from Schneider Optics and the NT59-870 from Edmund optics.

5.2.1 Schneider Optics Xenoplan Compact Lens

The Xenoplan Compact C-Mount Lens from Schneider Optics is a ruggedized machine vision lens ruggedized with iris and aperture locks.

Aperture	f/1.4 to f/11
Focal Length	23 mm
Length	40.4 mm
Diameter	34 mm

Table 5.5: Xenoplan Compact Lens specifications. [2]

5.2.2 Edmund Optics NT59-870

The NT59-870 is a machine vision lens designed for high-performance applications in low lighting. It features a ruggedized housing and iris and aperture locks.

Figure 5.4: Schneider Optics Xenoplan Compact Lens.

Aperture	f/1.8 to f/16
Focal Length	12 mm
Length	27.9 mm
Diameter	34 mm

Table 5.6: Edmund Optics Compact Lens specifications. [3]

5.2.3 Selection

Since both lenses are available in various focal lengths, that factor was not as important as the durability of the lens. Since the Xenoplan has previously been used in a NASA technology demonstration, it was selected as the lens for the star tracker.

5.3 Hardware testing

In order to verify the usability of the hardware as well as further validate the star tracking algorithms, a simulated star field test was set up. The test consists of one computer running a star field simulator and another computer attached to the camera which takes a

Figure 5.5: Edmund Optics NT59-870.

picture and runs the star tracking algorithm. This test setup is running as a demonstration and will be refined in order to produce results which can be rigorously analyzed.

5.3.1 Setup

The setup for the simulated star field test is fairly simple. One computer running the star field simulator Stellarium [22] is connect to a computer monitor at the end of the table. At the other end, the camera is connected to a second computer running the star tracker software as well as the camera drivers. The distance the camera is from the screen , d, is calibrated using an image with two known stars, for example Alpha Orionis and Gamma Orionis. These two stars have an angle between them γ and a distance apart on the screen ρ. These values are related using Eq. 5.2.

$$d = \frac{\rho}{\tan \gamma} \tag{5.2}$$

Once the screen and camera have been set up, Stellarium is given an attitude and the star tracking algorithm is run. In contrast to the process in Chapter 4, in this test the camera takes an actual, not a virtual, image and performs star tracking on that image. Once the star tracker returns a result, the error can again be found using Eqs. 4.11, 4.12, 4.13, and 4.14. Figs. 5.6 and 5.7 show the hardware setup, including the camera and starfield simulation.

Figure 5.6: Simulated star field setup.

5.3.2 Demonstration

The setup was used as a proof-of-concept in a demonstration mode. For this mode, the star tracking simulator was given an initial attitude estimate in the form of the identity of one of the stars in the image. The star tracker returned a DCM which was applied to the unit vectors of the catalog stars to yield a set of predicted unit vectors. This calculation is shown in Eq. 5.3.

$$\mathbf{u}_{pred} = \mathbf{R}_{calc}\mathbf{u}_{cat} \tag{5.3}$$

Figure 5.7: Simulated star field setup (camera view).

In this way the predicted unit vectors $\mathbf{u}_{pred}$ could be compared to the observed unit vectors $\mathbf{u}_{obs}$ to do a preliminary error analysis. For the demonstration, this analysis consisted first of finding a angular error θ_{err} given by Eq. 5.4.

$$\theta_{err} = \cos^{-1} \mathbf{u}_{pred}^T \mathbf{u}_{obs} \tag{5.4}$$

Also, the x- and y-components of the angles of both sets of unit vectors to the boresight direction were found using Eqs. 5.5 and 5.6.

$$\alpha_x = \tan^{-1} \frac{\mathbf{u}_x}{\mathbf{u}_z} \approx \frac{\mathbf{u}_x}{\mathbf{u}_z} \tag{5.5}$$

$$\alpha_y = \tan^{-1} \frac{\mathbf{u}_y}{\mathbf{u}_z} \approx \frac{\mathbf{u}_y}{\mathbf{u}_z} \tag{5.6}$$

These were plotted on the same graph to visually compare the attitude solution. Fig. 5.8 shows the image that was taken with blue x's over the centroid locations. Fig. 5.9 shows the predicted and observed unit vectors. For that particular demonstration, the two sets of unit vectors lined up very closely.

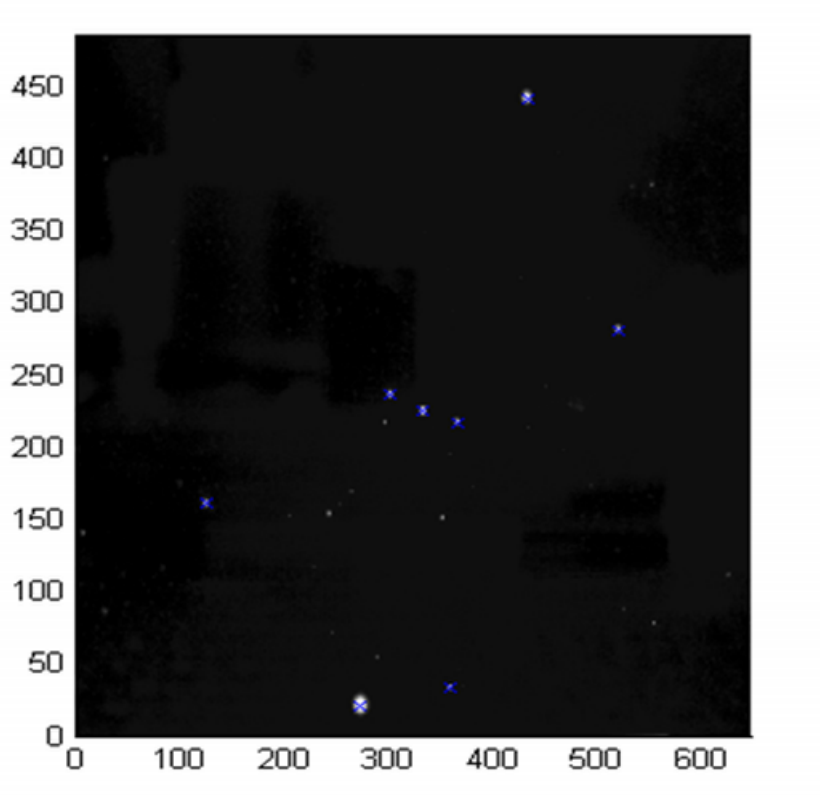

Figure 5.8: Star field image with centroid locations.

5.3.3 Further analysis

While the demonstration mode of the simulated star field setup shows promise, a few adjustments must be made in order to perform a rigorous analysis. First, the capabilities of the Stellarium program must be further investigated. Currently, the only way to change the attitude of the star field image is either with a mouse or by manually typing in a desired celestial object. While this functionality is fine for a demonstration, the ability to move to a series of desired attitudes would be required to perform the same kind of analysis as in Chapter 4. In addition, the calibration of the setup and star tracking program must be refined so that an initial attitude estimate need not be given.

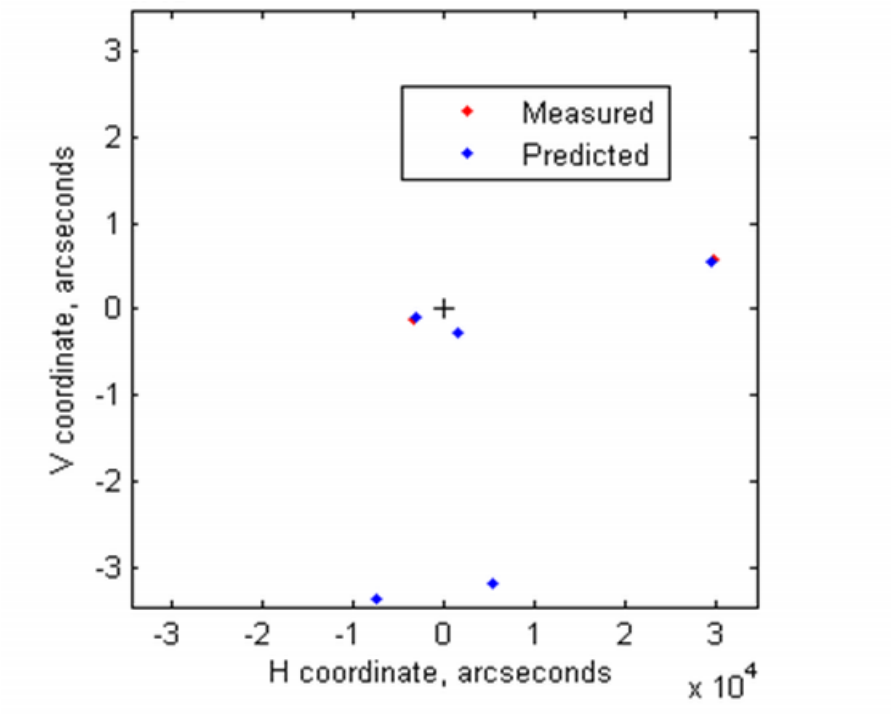

Figure 5.9: Predicted and measured unit vectors.

Chapter 6

Conclusions

The star tracker system defined in this research has the potential to be viable for CubeSat use. The computer generated image test consistently computed a correct attitude to within tens of arcseconds or returned an error. Only twice out of 200 times did the star tracker return a false positive, and even then the attitude was off by tens of degrees. A check with another attitude determination method should allow the flight computer to discard the result. Of course, the actual star tracker will not provide such accurate results due to imperfections in the lens, false stars, and other error sources.

6.1 Planned Demonstration of Student Satellites

The star tracker system being designed in this research is in a unique position to be put to a practical test in space. The design is planned to be included on two separately funded student satellite missions, known as Bevo-2 and ARMADILLO. Both satellites will employ the camera from Chapter 4 as a star tracker, and Bevo-2 will additionally employ the camera to take pictures of its partner satellite, AggieSat4. The satellites have an attitude determination and control suite using sun sensors, a magnetometer and MEMS gyros, so they can verify the performance of the star tracker to within the accuracy of those instruments. Also, the star images can be downloaded and post-processed.

Prior to launch, the camera system will undergo a series of hardware tests to verify its performance in space. The camera and lens system will be first put though a vacuum

chamber test. This procedure measures the amount of outgassing the star tracker will undergo once in orbit. Outgassing, which is the release of chemical vapor induced by a vacuum, is a particular concern for a star tracker since this residue can settle on the lens and inhibit the star tracker performance.

The camera system will also undergo thermal testing. A satellite and its hardware experience a broad range of temperatures in space as the satellite passes from sunlight to shadow. The camera and lens must be able to handle this thermal shift without degrading performance.

Finally, the lens will undergo vibration testing. This hazard is a concern due to launch, when the star tracker will be subjected to high acceleration and vibration. The lens is particularly susceptible to high vibration because of its precision optics. These various tests will ensure that the star tracker will perform as expected when it is launched.

6.2 Future Work

Future work for this research can take three tracks: improving the accuracy of the star tracker, expanding the versatility of the camera system, and improving the fidelity of the simulated star field analysis. To improve the performance, an on-orbit calibration algorithm could be implemented, possibly one similar to the one developed by Mortari [23]. In addition, a higher resolution camera might provide more information. In general, the hardware implement ion will undergo continuous iteration and reflect any new technology or capabilities.

The camera system also has the potential to be more versatile. A planetary navigation algorithm, like the one proposed by Christian [24], can be added to the suite of existing software. Also, future missions will perform autonomous rendezvous and docking using the

same sensor, so a proximity operations algorithm will be necessary for those missions.

In addition, the simulated star field analysis must be further refined so that those results can contribute to the design of the star tracker. This test accounts for more of the actual errors that the star tracker might experience on-orbit. The results of this test will provide a better idea of the calibration that must be done to the star tracker software, including parameters such as the threshold I_{thresh} and ROI size a_{ROI} used in Section 2.1.

After the simulated star field analysis, a night sky test will be the final step in testing the star tracking algorithm. The setup for this test is fairly simple. The star tracker is brought to an area with a very clear view of the sky and takes images of the sky. The identified stars and attitude is then compared with the actual observed stars. Once the star tracking system passes all of these tests, it will be ready to be used on a satellite.

Bibliography

[1] "Industrial Image Processing | mvBlueFOX-M - USB 2.0 board-level camera," 10 2011, accessed: 11/23/2011. [Online]. Available: http://www.matrix-vision.com/USB2.0-board-level-camera-mvbluefox-M.html

[2] "XENOPLAN 1.4/23mm COMPACT," accessed: 11/23/2011. [Online]. Available: https://www.schneideroptics.com/Ecommerce/CatalogItemDetail.aspx?CID=1377IID=5966

[3] "Compact Fixed Focal Length Lenses - Edmund Optics," 2011, accessed: 11/23/2011. [Online]. Available: http://www.edmundoptics.com/products/displayproduct.cfm?productid=3070

[4] D. Mortari, M. Samaan, and C. Bruccoleri, "The pyramid star identification technique," *Navigation*, vol. 51, pp. 171–183, 2004.

[5] C. Padgett and K. Kreutz-Delgado, "A grid algorithm for autonomous star identification," *IEEE Trans. Aerospace Electron. Syst.*, vol. 33, pp. 202–213, 1997.

[6] "Demo Kits - MT9P031I12STMD - Aptina Imaging," accessed 11/23/21011. [Online]. Available: http://www.aptina.com/products/demo_kits/mt9p031i12stmd/

[7] *CubeSat Design Specification*, The CubeSat Program Std., Rev. 12.

[8] M. Swartwout, "A brief history of rideshares (and attack of the CubeSats)," in *Aerospace Conference, 2011 IEEE*, 2011.

[9] "UTIAS/SFL - CanX-1 Mission Objectives," 2001, accessed: 11/23/2011. [Online]. Available: http://www.utias-sfl.net/nanosatellites/CanX1/

[10] M.Thompson, "Michael's list of CubeSat satellite missions," June 2009, accessed: 11/23/2011. [Online]. Available: http://mtech.dk/thomsen/space/cubesat.php

[11] K. Sarda, C. Grant, S. Eagleson, D. K. A. Shah, and R. Zee, "Canadian advanced nanospace experiment 2 orbit operations: One year of pushing the nanosatellite performance envelope," in *Proceedings of the 23rd Annual AIAA/USU Conference on Small Satellite*, 2009.

[12] A. Schwarzenberg-Czerny, W. W. Weiss, A. Moffat, R. Zee, S. Rucinski, S. Mochnacki, J. M. M. Breger, R. Kuschnig, O. Koudelka, P. Orleanski, A. P. A. Pigulski, and C. Grant, "The BRITE nano-satellite constellation mission," accessed: 11/23/2011. [Online]. Available: http://www.utias-sfl.net/docs/LivePapersAsOfJan2011/BRITE-COSPAR2010-PaperSR-WW-REZ-SM-AS-TM.pdf

[13] M. Smith, S. Seager, C. Pong, D. Miller, G. Farmer, and R. Jensen-Clem, "ExoplanetSat: detecting transiting exoplanets using a low-cost CubeSat platform," in *Space Telescopes and Instrumentation 2010: Optical, Infrared, and Millimeter Wave*, 2010.

[14] C. Liebe, "Accuracy performance of star trackers - a tutorial," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 2, pp. 587–599, April 2002.

[15] C. Cole and J. Crassidis, "Fast star-pattern recognition using planar triangles," *J. Guid. Control. Dynam.*, pp. 1283–1286, 1994.

[16] M. Kolomenkin, S. Pollak, I. Shimshoni, and M. Lindenbaum, "Geometric voting algorithm for star trackers," *IEEE Transactions on Aerospace and Electronic Systems*,

vol. 44, no. 2, 2008.

[17] D. Mortari, "Search-less algorithm for star pattern recognition," *J. Astronaut. Sci.*, vol. 45, pp. 179–194, 1997.

[18] G. Lerner, *Three-Axis Attitude Determination*, J. Wertz, Ed. D. Reidel Publishing Co.: D. Reidel Publishing Co., 1978.

[19] M. Shuster and S. Oh, "Attitude determination from vector observations," *Journal of Guidance and Control*, vol. 4, no. 1, pp. 70–77, Jan–Feb 1981.

[20] F. Markley, "Attitude determination using vector observations and the singular value decomposition," *J. Astronaut. Sci.*, vol. 38, no. 3, pp. 245–258, 1988.

[21] J. Enright, D. Sinclair, C. Grant, G. McVittie, and T. Dzamba, "Towards star tracker only attitude estimation," in *24th Annual AIAA/USU Conference on Small Satellites*, 2010.

[22] "Stellarium," accessed: 11/23/2011. [Online]. Available: http://www.stellarium.org/

[23] M. Samaan, D. Mortari, and J. Junkins, "Nondimensional star idenification for uncalibrated cameras," *J. Astronaut. Sci.*, vol. 54, no. 1, 2011.

[24] J. Christian, "Optical navigation for a spacecraft in a planetary system," Ph.D. dissertation, The University of Texas at Austin, 2011.

Vita

Christopher Ryan McBryde was born in Winter Park, FL. After graduating with an International Baccalaureate diploma from Winter Park High School in 2006, he enrolled at the University of Florida in Gainesville, FL. During this time there, he completed two summer internships with Florida Turbine Technologies, Inc. in Jupiter, FL. Christopher graduated summa cum laude from the University of Florida in 2010. He enrolled in graduate school for orbital mechanics at the University of Texas at Austin the following fall under the guidance of Dr. E. Glenn Lightsey.

Permanent address: mcbryde@utexas.edu

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.