Design of an Image Generation Truth Model for Testing Attitude Determination Algorithms for a Star Tracker

A project present to
The Faculty of the Department of Aerospace Engineering
San Jose State University

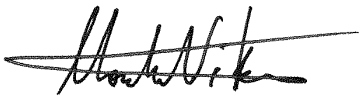
in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

By

Adrianna Y. Fukuzato

May 2015

approved by



Dr. Nikos Mourtos
Faculty Advisor

San José State
UNIVERSITY

The Designated Project Committee Approves the Project Titled

DESIGN OF AN IMAGE GENERATION TRUTH MODEL FOR TESTING
ATTITUDE DETERMINATION ALGORITHMS
FOR A STAR TRACKER

by

Adriana Y. Fukuzato

APPROVED FOR THE DEPARTMENT OF AEROSPACE ENGINEERING

SAN JOSÉ STATE UNIVERSITY

May 2015

Dr. Nikos J. Mourtos, Project Advisor
Department of Aerospace Engineering

14 May 15
Date

Ronny Votel, Project Mentor
Skybox Imaging + Google

May 15, 2015
Date

ABSTRACT

DESIGN OF AN IMAGE GENERATION TRUTH MODEL FOR TESTING ATTITUDE DETERMINATION METHODS FOR A STAR TRACKER

by

Adriana Y. Fukuzato

The objective of this project was to create software that would replicate the functionality of a star tracker in attitude acquisition mode. The software produced was written in Matlab and solves the "lost-in-space" problem where the quaternion output defines a rotation from the ICRF frame to the camera frame. This project consists of two parts: image generation and algorithm testing. Image generation yields unique star field images with star centroids accurate up to 1/10 of a pixel. Algorithm testing takes those images as inputs and runs a star identification and star matching algorithm to form a triangular feature [17] and find a star match. Attitude determination is completed using Davenport's Q Method, which outputs the same quaternion with an average loss function value of 1.61e-4 after a Monte Carlo simulation of 1000 runs. The star identification and star matching algorithms runs on average in 1.9 seconds, with a 99.1% probability of finding a star match. The Hipparcos catalog was used as the main star catalog, of which stars as dim as magnitude 6 were assumed visible to the detector. This project defines camera specifications similar to a star tracker being used on a small satellite in Low Earth Orbit.

Table of Contents

List of Figure

Acronyms

AD	attitude determination
CCD	charge-coupled device
DCM	direction cosine matrix
DG	digital gain
FOV	field of view
GNC	guidance, navigation, and control
HDID	Henry Draper Identification number
ICRF	International Celestial Reference System
LEO	low earth orbit
NEA	noise equivalence angle
QE	quantum efficiency
QSE	quantum step equivalence
ST	star tracker
TF	triangular feature
TLE	two line element

Nomenclature

A	area of the detector
D	diameter of lens aperture
f	focal length of detector
Int	integration time
Nsize	square side length of an individual star
pp	pixel pitch
$\mathbf{q}$	quaternion
$\mathbf{r}_{\mathbf{rel}}$	vector from (0,0) to centroid on individual star block
σ	Gaussian distribution standard deviation
U	detector size in the x-direction
Uc	center of the detector in the x-direction
V	detector size in the y-direction
Vb	vector measured in the body/star tracker reference frame
Vc	center of the detector in the y-direction
Vr	vector measured in the reference frame

1. Introduction

Traditional Attitude Determination systems consist of an absolute attitude reference system and an inertial attitude reference system. The absolute attitude sensor determines the pointing direction of the spacecraft, which is then used to calibrate the inertial attitude sensor that measures the changes in attitude between the absolute calibrations [1]. Star trackers fall under the category of absolute attitude sensor. Compared to magnetometers, sun sensors, horizon sensors and others, star trackers are by far the most accurate. They provide arc second accuracy in three axes and serve as the most important type of attitude determination on a spacecraft [1].

In effect, it is necessary that star trackers are reliable in providing attitude. For small satellites in low earth orbit, star trackers are needed to track the stars and obtain lock even with the satellite's changing pointing direction. A star tracker functions in two modes: initial attitude acquisition and tracking mode [2]. The objective of this project is to replicate the initial attitude acquisition mode of a star tracker. Matlab will be the used as the main tool to design the software. First, test images of the night sky will be generated using a random quaternion to represent a portion of the sky that the star tracker would be pointing towards. Then an algorithm will run star identification and attitude determination on those images to output the corresponding attitude.

1.1 Motivation

In January 2014, I started an internship at Skybox Imaging working as a satellite controller of their recently launched SkySat-A. At the time, Skybox was a start-up company with a vision of launching a constellation of imaging satellites. They then became acquired by Google in the summer of 2014 and are continuing to work towards delivering data analytics and will be

launching their third satellite in December 2015. During the first week of training, we received a lecture from the lead Guidance, Navigation, and Control engineer. He discussed star trackers and their role for SkySat-A. Star tracker functionality interested me, especially with their application to the celestial sky. I decided to make a Masters project out of this and sought help from the GNC lead who was very passionate about the subject himself.

Initially, the project began with the goal of taking pictures of the night sky and using those in determining attitude information. However, this proved difficult with the camera that I had available and decided to get clearer direction from a potential mentor at Skybox who suggested that generating my own images would be more effective. This process essentially yielded a truth model based on the fundamental physics behind star light and image detectors. With the help of the lead GNC engineer at Skybox, this project was made possible.

1.2 Literature Review

Star tracker sensors have undergone a huge development in the last ten years. An overview of major advancements and papers on star trackers will be presented in this section. Second generation star trackers will be compared to first generation star trackers. Then developments in the algorithms used in the processing computers will be reviewed. Although the star tracker design presented in this paper does not account for tracking mode, the papers summarized include both initial attitude acquisition and tracking mode. Finally, recent papers involving expectations for future models will be discussed.

1.2.1 Second Generation Star Trackers

First generation star trackers, the first of which used charge coupled devices (CCDs) were pioneered at the Jet Propulsion Laboratory in Pasadena, CA [3]. They were considered a first

generation model in that they were not able to directly output attitude information in the inertial reference frame, rather relied on an external computer to process the spacecraft attitude data. The JPL ASTROS star tracker, shown in Fig. 1, was able to detect two to six star images per frame. The data coming from the detected stars would then be transferred to the satellite's main computer or sent to the ground to be processed later [3]. In this case, additional information like the sun vector may have been necessary in order to completely determine the attitude of the spacecraft.

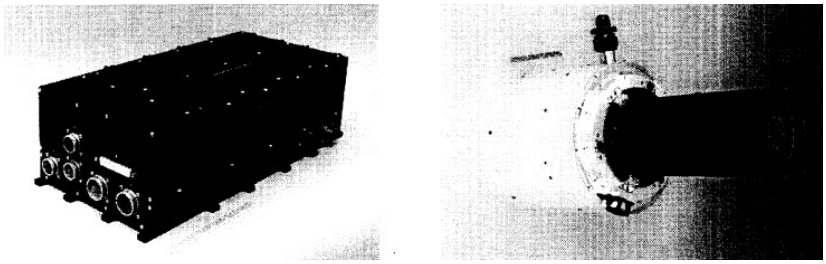


Figure 1-1: JPL Astros Star Tracker [3]

In comparison, second generation star trackers are able to output three axis attitude autonomously without requiring external processing. Second generation sensors have been developed in the last decade and have major improvements compared to previous versions. Star pattern recognition can be done autonomously by the microcomputers which are attached to the cameras. These microcomputers have internal star catalogs and can solve the "lost-in-space" problem. In addition, they utilize an average of 25 to 65 stars in the field of view (FOV) for each data frame [3]. The attitude obtained is also based on a signal which is much larger and which improves accuracy over the entire sky. Because these star tracker models can function autonomously they can be integrated into the spacecraft more economically as well. Figure 2 shows the Doug Sinclair ST-16 star trackers currently used on-board Skybox Imaging's SkySat-1 and SkySat-2 [4].

Figure 1-2: Doug Sinclair ST-16 star trackers [4].

A star tracker functions in two modes: initial attitude acquisition and tracking mode. The first mode, initial attitude acquisition, requires pattern recognition of the stars in FOV. A second generation star tracker determines which stars are present in the FOV using an algorithm; the output is computed very quickly, all within a few seconds or less [3]. The image provided in the FOV represents less than 1% of the night sky and it is important for star trackers to use efficient star patter recognition algorithms.

According to Eisenman, the most important parameter that allow second generation STs to function autonomously is the FOV. They range from a few degrees to over 30 degrees diagonally [3]. The FOV can be measured horizontally, vertically, or from one corner to the corner diagonally. The smaller the field of view, the better the individual angular resolution. This results in an increased accuracy in pitch and yaw. The roll accuracy will remain constant. For a smaller field of view, the camera needs to have a lens aperture with a larger diameter in order for the ST to detect the same average number of stars. This will cause an increase in the mass of the ST since the focal length is increased as well. The number of stars in the internal catalog must also be bigger with a larger aperture since dimmer stars can be detected in the FOV [3]. Camera specifications are parameters that should be considered first when deciding what type of ST to

choose for a spacecraft mission. This will determine how much space should be allocated economically in the spacecraft, how complex the pattern recognition algorithm will be, and thus the cost of the ST. This paper describes the fundamental importance of the camera's specifications in choosing what denotes optimal star tracker hardware.

The mass of the ST is important especially for the increasing market of CubeSats and microsatellites. The mass varies from a few hundred grams to more than 20 kg. The processing electronics and the optics are what determine the mass of a ST [3]. Typically, larger star trackers are more accurate and more expensive.

Because second generation STs are highly sensitive, they detect many stars in the FOV. Sky coverage denotes the percentage of the sky over which the ST can lock onto stars and track stars [3]. If the number of stars in the FOV are too low, then the algorithm will reject the image. The acquisition catalog is also much smaller than the full tracking mode catalog. A typical second generation ST will have sky coverage close to 100% [3]. The star catalog size that a ST has depends on the sensitivity of the system. A large star catalog is required for cameras with larger aperture and longer exposure time [3]. Figure 3 shows catalog size as a function of the FOV. For a larger diagonal FOV with lower average stars in its FOV, a smaller catalog size is needed. For a diagonal FOV with an average of 75 stars, a much larger star catalog is required. In contrast, a ST with a diagonal FOV with a lower average number of stars requires a smaller catalog. It is better to have a smaller star catalog because larger catalogs occupy more nonvolatile memory space and complicate the initial attitude acquisition algorithm. The processing power is also thus increased.

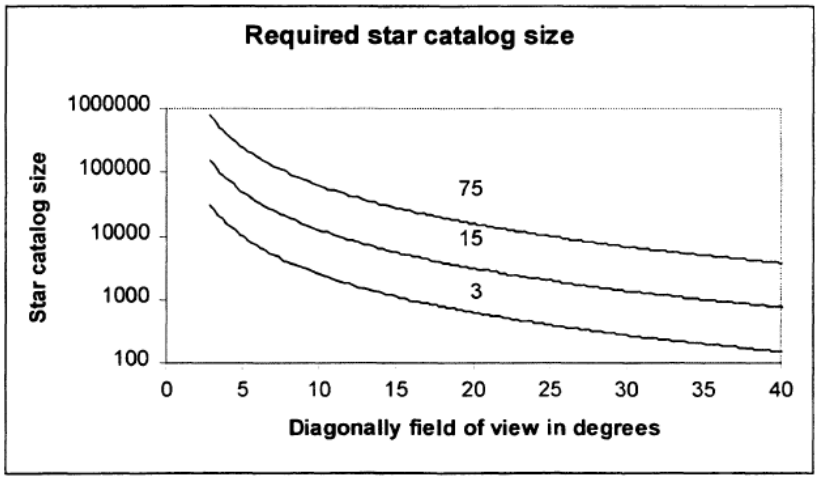


Figure 1-3: Star catalog size vs diagonal FOV [3].

The accuracy of a ST depends on the star catalog used. As of 1997, the HIPPARCOS star catalog became available which contains the most recent, accurate star positions. The purpose of the HIPPARCOS satellite, sent by the European Space Agency, was to record the 120,000 brightest stars with an accuracy of 1 milli-arcseconds. Using this catalog, STs are able to provide accuracy in the range of 0.1 – 20 arcseconds [2]. These accuracies in roll, pitch, and yaw are specified about the optical boresight. Sub-pixel accuracy in particular can be achieved by calculating the centroid of star regions. Therefore it is better to have a smeared image instead of a sharp image. If the light coming from a star is spread over a few pixels, compared to it being focused onto one pixel, then it becomes possible to determine the position of the star by calculating the centroid of that region. The location can be determined to a fraction of a single pixel [2].

Angular resolution is also based on having multiple stars in each frame. The stars present on the focal plane can be translated to a right handed coordinate inertial reference frame, one of which is the Earth Centered Inertial J2000 reference frame. This conversion depends on the camera and CCD specifications such as the focal length, pixel width and height. Lens distortion

can also be corrected through calibration [2]. As mentioned before, the roll or twist angle is about 4 to 12 times less accurate compared to the yaw and pitch angles. The noise equivalent angle is the variation of the attitude estimate when there is a constant input. This measures accuracy and can be found through mounted simulations.

In terms of future advancements, Eisenman expects STs to undergo developments in accuracy and miniaturization. STs are used on all sort of missions and for an increasing market of microsatellites, STs will be developed for use on both large and small missions. In particular, active pixel sensor technology has become an alternative to CCDs. They have an advantage over CCDs in that they include enhanced radiation resistance and control over individual pixel integration times [3]. It has also been thought of to combine a ST with a GPS receiver for low – Earth applications, where the two components use the same microcomputer. This may be able to replace the standard GPS system on satellites [3].

It has even been proposed that star tracker devices are accurate enough to be used as the only form of attitude determination on small satellites orbiting Earth [5]. A STOAE system has never been implemented before on-board a satellite mission, although many tests have been run in particular with an S3S star tracker. Typical Low-Earth-Orbit (LEO) satellites use magnetometers and sun sensors and other types of sensors in conjunction with each other to produce three angles of attitude, whereas star trackers don't need to work with other devices to output the data. There is a great significance in implementing STOAE on nanosatellites in terms of mass and cost reduction; removing extra sensors may decrease the mass and thus cost of the satellite. One disadvantage is the fact that building a new algorithm to account for it being the only attitude information bank will be difficult [5] and most likely increase their overall cost. Star trackers range in the price range of $100K to 1 million dollars.

1.2.2 Star Tracker Algorithms

The initial attitude acquisition mode of a star tracker is dependent on the "lost-in-space" star pattern recognition algorithm which can determine what stars are present in the field of view and convert them into useful attitude determination without having a priori knowledge. Spratling and Mortari provide a very good review of star identification algorithms since the beginning of their development. Star identification typically follows the workflow shown in Fig. 4 [6]. Previously, star trackers needed a priori knowledge of the attitude in order for their algorithms to work. In 1981, Junkins et al. published a star pattern recognition algorithm which could parse a star catalog but the attitude estimate, besides requiring a priori knowledge, would only updated once or twice a minute in real time [4]. It wasn't until 1997 that Mortari a faster database search technique called the Search-Less Algorithm (SLA) [6]. This algorithm was tested in an Indian satellite. Mortari later on developed the "Pyramid" algorithm which is on exclusive contract to StarVision Technologies and has been tested on Draper's "Inertial Stellar Compass" star tracker and on MIT's satellites HETE and HETE-2 [6]. Currently star pattern recognition can be accomplished through different processes such as least-squares, TRIAD, or QUEST [6]. Simulated star fields can be used to test the effectivity of these algorithms [6]. A three-axis spacecraft simulator provides the reference frame for these algorithms to work with. The difference between TRIAD and QUEST is that QUEST produces better accuracy, while TRIAD produces good accuracy and a lower computation time [6].

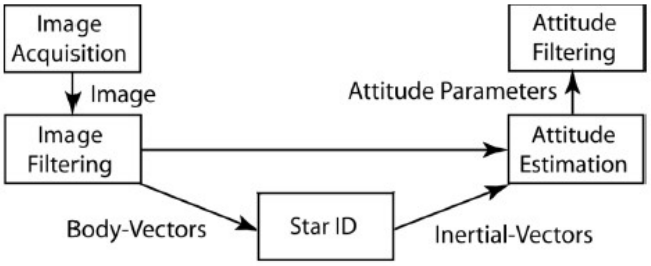


Figure 1-4: Typical Attitude Determination flow chart for Star Trackers [6].

Before stars vectors can be used in attitude determination algorithms, they must first be detected by the camera sensor. Star detection is dependent on a given threshold. The stars that are detected in an image will be used in finding a matching triad only if they pass a pre-defined brightness threshold. This value is chosen empirically and is proportional to the exposure time of the star tracker [7]. In the case that there is a low threshold, then noise objects that may not be actual stars will be detected. For a high thresholds, pixels nearby bright stars can be detected due to the spreading of the light on the pixels. Kandiyil eliminates this by setting a minimum distance between any two stars as well as through centroiding [7].

The star recognition method is usually based on having a triad of stars that form a triangle. The detected stars will have vectors in the spacecraft frame, and those stars will be compared to the vectors of stars in the inertial frame. One method of matching stars is through area and polar moment calculations. The areas and moments of the stars, using the sides of the triangle and the star vectors in the inertial frame, are calculated and compared to a Look Up Table. The Look Up Table, as referred to in Kandiyil's Master's thesis, includes all the areas and moments of the stars in the catalog. If there are any matches, then those stars are considered to be detected [7]. Another pattern method is the planar moment method. This method requires calculating the angles between the sides of the triangle. They are then compared with the Look

Up Table that has all the angles between the stars in the star catalog, calculated. The vector angle method is another method carried out by finding the angle between each set of two star vectors and comparing those to the Look Up Table as well [7].

Once a triad of stars are detected, their centroids must be calculated in order to rule out any noise objects and to provide sub-pixel accuracy. A typical centroiding algorithm will give a precision of 1/10 of a pixel [2]. A point spread function is used to measure how much light will spread across the CCD array. With this and the magnitude of the star, the size of the array can be determined for the algorithm. The size of the array is basically the number of pixels surrounding the brightest pixel that will be used in the centroid equation. Kandiyil points out that it is a good idea to rule out bright regions that have multiple adjacent pixels with equal intensity values since it would be hard to apply the centroiding algorithm to that region and could result in an inaccurate center [7].

Because a star tracker can't cover the entirety of a constellation in its FOV, specific parameters need to be set in order to identify which stars make up a triad of stars. Kandiyil specifies that from one detected star, the angular distance to neighboring stars, spherical angles between close stars, and magnitude are all useful in identifying a star triad. He shows that from the pivot start, or first detected star, to the second and third star, there is a higher frequency of stars within the 6-8 degree angular distance range [7]. These stars are then compared with the stars selected in the star catalog. In Kandiyil's thesis, two separate catalogs are constructed from the main HIPPARCOS catalog: the Base catalog and the Oriented catalog. This way, less processing power is required in the star triad formation.

In summary, the image analysis process defined by Kandiyil takes the following form. From a given image, all the stars that pass a predefined threshold are detected and put through an

algorithm to find their centroids. The brightest of those stars is then taken as the pivot star, and the second and third star are chosen to form a triad. Using the planar moment method, a match between the selected triad and those in the base catalog is chosen [7]. The unit vectors in the spacecraft frame are then calculated. The corresponding unit vectors in the Oriented catalog are then found and used as the unit vectors in the ECI frame. These inertial frame vectors are then put through a TRIAD algorithm which output a quaternion and thus the attitude information.

The HIPPARCOS catalog contains around 115,000 stars that have magnitudes up to values of 11. A higher magnitude denotes a fainter star. Figure 5 shows how the number of stars in the catalog increases with magnitude [7]. Typical low cost star trackers only detect stars with magnitude as faint as 6 [7]. It can be seen from Fig. 5 that more sophisticated STs are required to process and detect stars with higher magnitude. The Base catalog in Kandiyil's thesis truncates stars that have magnitude greater than 6, leaving only 5029 stars to process.

2 Attitude Determination Sensors

Compared to other attitude determination devices, star trackers can provide greater than arcsecond accuracy, as well as all three angles in attitude. They are considered to be the most accurate devices used for attitude determination. Some sensor devices typically used in Attitude Determination Systems (ADS) include magnetometers, sun sensors and gyroscopes, for comparison.

Magnetometers can detect the strength of the Earth's magnetic field and align the spacecraft accordingly. They provide the direction and magnitude of the Earth's magnetic field. They are not accurate inertial attitude sensors for the reason that the Earth's magnetic field is not entirely known and the models used to predict its direction can cause errors [8]. In addition,

because the Earth's magnetic field decreases with $1/r^3$, the spacecraft's magnetic biases dominate the magnetic field measurement and thus magnetometers are only most effective on spacecraft in orbits less than 1000 km [9].

Sun sensors can track the sun in various ways to determine where the satellite is, but only provide two degrees of freedom in attitude and arcminute accuracy. Gyroscopes are spinning flywheels that apply torques which align the axes of the satellite. Star trackers, on the other hand, have powerful microcomputers that can determine the spacecraft's motion and extrapolate the attitude from mathematical information internally, and provide three degrees of freedom in attitude as well as arcsecond accuracy [9].

3 Project Overview

The objective of this project was to replicate the fundamental functionality of a star tracker in initial attitude acquisition mode. This required star images with known stars in the field of view in order to accurately test attitude determination algorithms. For this reason, image generation was incorporated into the project as the first part.

Overall, this project can be split up into two parts: image generation and algorithm testing. Image generation will focus on outputting star field images before the attitude determination process. This creates a truth model for testing whether the algorithm method is accurate. The steps for generating a star image are shown in Fig. 3-1. This process begins with a random quaternion. The quaternion represents a rotation from the spacecraft (S/C) frame to the Earth-Centered Inertial (ECI) frame. Therefore, the vector component can be considered the bore sight of the star tracker camera. Using this information, the vectors of all the stars in field of view can be obtained for both the ECI frame and star tracker frame. Noise was not incorporated

into image generation. A quantum efficiency is defined with respect to the camera specifications which prevent the detector from being perfect and is thus more realistic. However, because noise was not incorporated, the star identification algorithm was written with less restrictions as will be defined in Section 8. Overall, the output is an accurate representation of a portion of the night sky that a star tracker would see in orbit.

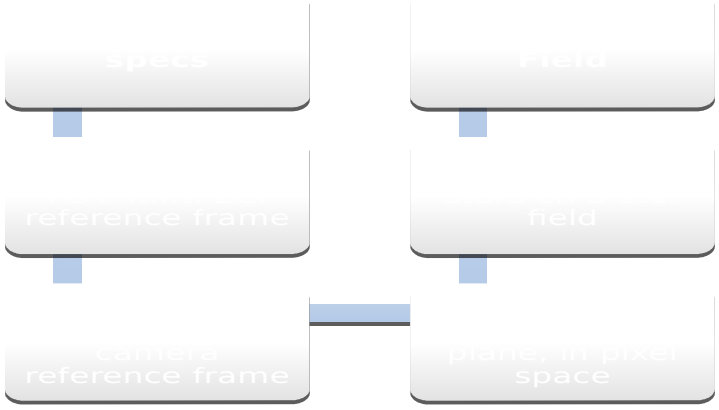


Figure 3-5: Flow Diagram for Image Generation

After star images were generated, the attitude determination process was started. The images were put through algorithms to output the correct attitude information. In theory, the quaternion that is output after attitude determination should match the quaternion that is used to generate the star image in part one. The steps for algorithm testing are shown in Fig. 3-2. Since a majority of this project involves transformations to other reference frames, the specific reference frames will be defined in the next section.

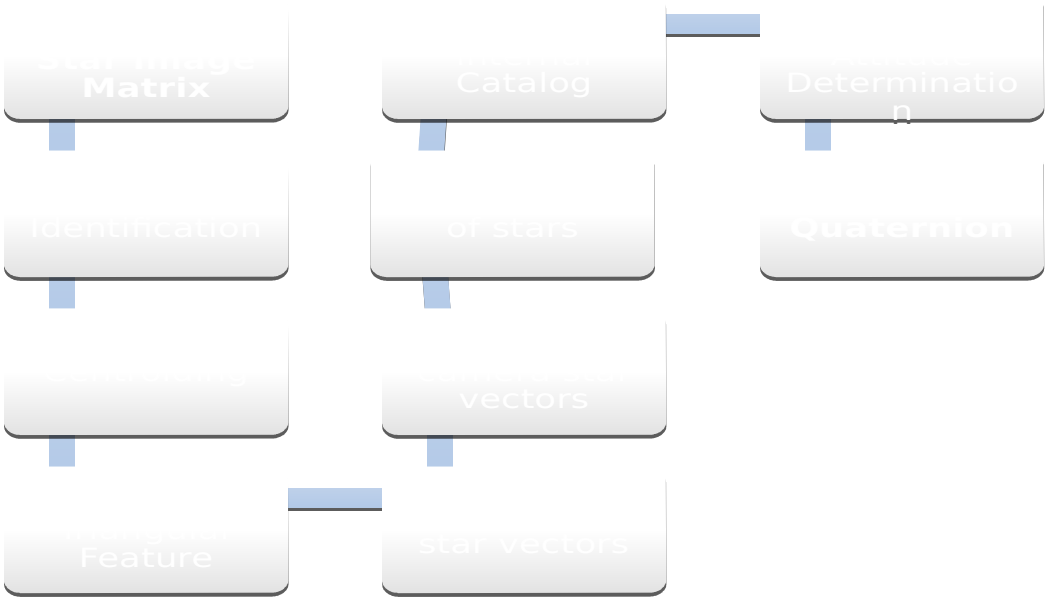


Figure 3-6: Algorithm Testing Flow Chart.

The attitude determination method that will be used is Davenport's Q-method. Once the image is read into Matlab, stars that pass a certain threshold will be chosen and centroided. The three brightest starts are selected to form a triad, which will be used in comparing to stars in the Hiparcos star catalog. After a match is made between the image's stars and the catalog's stars, the coordinates of the set of stars can be put through the Q-method in order to output the corresponding quaternion that relates the two reference frames.

4 Reference Frames

There are many reference frames used for attitude analysis. They are usually defined by the location of their origin and orientation of their axes. An inertial reference frame in particular is one in which Newton's laws of motion are valid. Any frame that is moving at a constant velocity and without rotation with respect to an inertial frame is also considered inertial [8].

4.1 Earth Centered Inertial Frame

The Earth Centered Inertial frame, or ECI, is one of many inertial reference frames used in spacecraft dynamics. Its center is fixed with the Earth's center; the x axis points along the vernal equinox, the z axis points through the North Pole, and the y axis completes the orthogonal system. Although the frame is rotating itself with respect to the Earth's rotation around the sun, this can be neglected when dealing with distances to celestial objects. This may not be the case, though, when dealing with problems where nearby planets are taken into account. The stars contained in the Hipparcos catalog are measured with respect to the International Celestial Reference Frame (ICRF) as are other star catalogs [8]. The origin of the ICRF is at the center of mass of the solar system.

The J2000 frame defines the ECI frame as derived on January 1, 2000 at 12:00 [9] with the vernal equinox at a particular position with respect to the sun. The reason why this derivation is important is because the Earth's precession has an effect on the location of the vectors measured in the ECI frame. In other words, an epoch is used to denote the starting time of the frame [9]. Epoch is important when defining observations of celestial objects.

The vectors of celestial objects measured from the ECI are usually given in coordinates of Right Ascension, RA, and Declination, De. Right Ascension is measured from the vernal equinox in a unit of time. At the vernal equinox the RA has a value of 0 hours, 0 minutes, and 0 seconds. The declination is measured in terms of degrees and ranges from 0 to positive 90 degrees or to negative 90 degrees. In order to make RA and Dec useful coordinates in this project they need to be transformed from RA, Dec to a spherical coordinate system. This is what will be done in creating the truncated catalog in Section 9. It is useful to note that the entire rotation matrix is not necessary in converting these coordinates to an x, y, and z location. The derivation of the

conversion has been omitted but the equations that convert a star's RA, Dec values to a spherical coordinate system are presented in Eq. (1).

$$x_{star} = \cos(Dec) * \cos(RA) \tag{1-1}$$

$$y_{star} = \cos(Dec) * \sin(RA) \tag{1-2}$$

$$z_{star} = \sin(Dec) \tag{1-3}$$

4.2 Spacecraft Frame

The spacecraft's reference frame is usually defined with the origin at some point in the spacecraft body. However, because components of the spacecraft can shift after launch and possibly even due to thermal effects, the axes are usually aligned with a navigational base that will stay rigid enough [8]. This navigational base includes the most critical payload instruments and attitude sensors. The spacecraft's reference frame is oriented with respect to some external reference frame, with that being the ECI in this project. The spacecraft reference frame in this project will be assumed as the star tracker's body; the z axis will point outwards toward the bore sight, and the x and y axes form the focal plane. A reference for the spacecraft's orbit can also be called the Local-Vertical Local-Horizontal reference frame [8]. This is usually defined for Earth-pointing spacecraft. The z axis points toward the center of the Earth along the bore sight of the spacecraft, the y axis points along the negative orbit normal and the x axis completes the orthogonal system. Figure 4-1 depicts this type of reference frame.

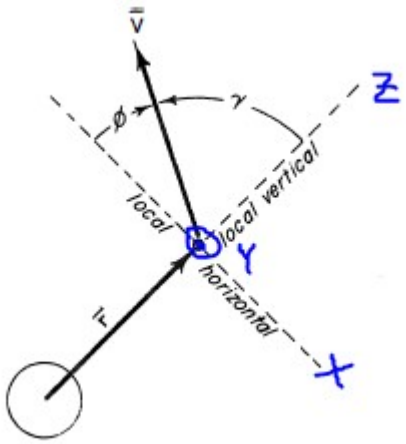


Figure 4-7: Local Vertical Local Horizontal spacecraft reference frame [10]

5 Attitude Representations

The importance of reference frames is tied to attitude determination. The attitude of a body is defined as the coordinate transformation that transforms the ECI reference coordinates into the body coordinates of the spacecraft. In other words, the attitude of a spacecraft represents an orientation of the body in space. This representation is obtained through attitude determination which involves solving for a particular matrix using vectors defined with respect to two difference reference frames. The following are different ways of representing that attitude transformation [11].

5.1 Direction Cosine Matrix

All coordinate transformations that give the attitude of a body are based on the direction cosine matrix (DCM). The DCM can also be called the attitude matrix. This matrix will transform vectors in the ECI frame, or reference frame, into the spacecraft frame. The DCM can be represented by Eq 2.

$$[A]=\begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix} \tag{2}$$

The unit vectors **u, v,** and **w** are the components of the unit vectors along the three axes of the ECI reference frame. If there exists a vector in the ECI frame, $\mathbf{a} = [a_1\ a_2\ a_3]^T$, then multiplying it by the attitude matrix, A, will express that vector in the spacecraft frame. See Eq. 3 for a representation of this.

$$[A]a=\begin{bmatrix} u\cdot a \\ v\cdot a \\ w\cdot a \end{bmatrix}=\begin{bmatrix} a_u \\ a_v \\ a_w \end{bmatrix}=a_B \tag{3}$$

5.1.1 Properties of the Direction Cosine Matrix

Basic properties of the A matrix are defined as follows and represented by Eq. 4 – Eq. 6 .

1. Each element in A is the cosine of the angle between the body unit vector and the ECI reference axis

2. Each vector in the spacecraft frame **u, v, w** are of unit length and therefore:

$$\sum_{i=1}^{3} u_i^2=1, \sum_{i=1}^{3} v_i^2=1, \sum_{i=1}^{3} w_i^2=1 \tag{4}$$

3. The unit vectors **u, v, w** are orthogonal to each other and therefore:

$$\sum_{i=1}^{3} u_i v_i=0, \sum_{i=1}^{3} u_i w_i=0, \sum_{i=1}^{3} v_i w_i=0 \tag{5}$$

4. Since the unit vectors are of unit length and are orthogonal to each other we get the follow relationships, a_B is the vector in the ECI frame mapped into the spacecraft frame.

$$[A][A]^T = 1, [A]^T = [A]^{-1} \tag{6-1}$$

$$det[A] = 1 \tag{6-2}$$

$$a = [A]^T a_B \tag{6-3}$$

The A matrix is considered a proper, real orthogonal matrix. It can be shown that this matrix transformation preserves the lengths of vectors and also the angles between them and so represents a rotation. If two proper real orthogonal matrices are multiplied, such as [A] = [A1] [A2], then two successive rotations result. First there is a rotation by [A1] and then by [A2].

5.2 Euler Angles

Euler angle rotations are defined as rotations about three orthogonal frame axes. If we have three orthogonal axes defined in the spacecraft frame, i, j, k, and axes in the ECI reference frame defined as I, J, K then there are several combinations of rotations that can be done. There are two distinct types of rotations.

1. Successive rotations about each of the three axes i, j, k. There are six possible order of rotation: i-j-k, i-k-j, j-i-k, j-k-I, k-i-j, k-j-i

2. First and third rotations about the same axis and the second rotation about one of the two remaining axes. There are 6 possibilities: i-j-i, i-k-i, j-i-j, j-k-j, k-i-k, k-j-k

The second type of rotation may be useful in avoiding singularities. The type of rotation depends on the situation. It is common to define the roll angle, ϕ, as a rotation about the x-body axis, the pitch angle, θ, as a rotation about the y-body axis, and the yaw angle, ψ, as a rotation about the z-body axis.

5.3 Quaternions

Quaternions are another form of attitude representation. Through linear algebra it can be shown that a proper real orthogonal matrix, such as the DCM, has at least one eigenvector with eigenvalue of unity. The eigenvector is therefore unchanged by the matrix A, meaning that it has the same components along the body axes as those along the reference frame axes: [A] $\mathbf{e}_1 = 1\mathbf{e}_1$.

The existence of this eigenvector demonstrates Euler's famous theorem which says that "*The most general displacement of a rigid body with one point fixed is a rotation about some axis,"* where the rotation is about the eigenvector, $\mathbf{e}_1$. Any attitude transformation by consecutive rotations about three orthogonal unit vectors can be achieved by a single rotation about the eigenvector with unity eigenvalue. This means that if A is a proper, real, orthogonal matrix which all attitude matrices are, then there must exist an eigenvalue of unity. The eigenvector corresponding to this eigenvalue is going to be the vector about which the rotation is done and also corresponds to the quaternion.

A quaternion is a vector defined by Eq. 7, and is composed of a scalar quantity, q_4, and a vector component, $\boldsymbol{q}$.

$$q = q_4 + i\,q_1 + j\,q_2 + k\,q_3 \equiv q_4 + q \tag{7}$$

The conjugate of **q**, is $q*¿q_4-i\,q_1-j\,q_2-k\,q_3$. Elements of a quaternion are sometimes referred to as the Euler symmetric parameters. They can be expressed in terms of the principal eigenvector, **e**, of the DCM as shown in Eq. 8.

$$q_1=\cos\left(\frac{\alpha}{2}\right) \tag{8-1}$$

$$q_2=e_1\sin\left(\frac{\alpha}{2}\right) \tag{8-2}$$

$$q_3=e_2\sin\left(\frac{\alpha}{2}\right) \tag{8-3}$$

$$q_4=e_3\sin\left(\frac{\alpha}{2}\right) \tag{8-4}$$

One important property is that $q_1^2+q_2^2+q_3^2+q_4^2=1$, where $|q|=1$. The DCM can therefore also be expressed in terms of the quaternion as shown by Eq. 8. Where the matrix Q in A(q) is defined as that given by Eq. 9.

$$[A(q)]=\left(q_4^2-q^2\right)1+2\,q\,q^T-2\,q_4[Q] \tag{9}$$

$$[Q]=\begin{bmatrix}0 & -q_3 & q_2\\ q_3 & 0 & -q_1\\ -q_2 & q_1 & 0\end{bmatrix} \tag{10}$$

A quaternion is similar to the DCM in that it represents attitude. It can be of advantage to use a quaternion over a DCM since a quaternion gives a complete attitude representation without having to deal with singularities. An example of a singularity is gimbal lock; the case where two of a spacecraft's axes are aligned which results in two degrees of freedom. A quaternion can represent attitude without having to deal with this and avoids any ambiguity when transforming

from one attitude to another. With regards to Euler angles, there may be several ways, even an infinite amount of ways, to represent attitude using Euler angles. Pitching up by 90 degrees and then rolling by 90 degrees gives the same attitude that does yawing by 90 degrees and then pitching up by 90 degrees. A quaternion is simple and complete in that it gives a clear and unique attitude representation.

6 Attitude Determination Methods

There are many ways to solve the attitude determination problem. Among these are Triaxial Attitude Determination (TRIAD) method, Davenport's q Method, and Quaternion Estimator (QUEST). Given two sets of vector measurements in different reference frames, the attitude matrix or representation can be solved for directly with some margin of inaccuracy, or also approximately using a least squares method. Davenport's q Method and QUEST both aim to solve Wahba's problem which was developed in 1965 by Grace Wahba [8]. Her problem was posed to solve for the orthogonal matrix, A, with determinant equal to +1 that would minimize the loss function and essentially provide an optimal least squares solution. The loss function is represented by Eq. 11.

$$L(A)=\frac{1}{2}\sum_{i=1}^{N} a_i \left\| b_i - A r_i \right\|^2 \tag{11}$$

In comparison, TRIAD takes a set of two measurements in each reference frame and obtains an orthonormal right-handed triad of vectors from each. However, this assumes that one of two unit vectors in the body frame is more accurately determined than the other. In other words, the estimate for attitude will satisfy $A\mathbf{v_{r1}} = \mathbf{v_{b1}}$ exactly but only $A\mathbf{v_{r2}} = \mathbf{v_{b2}}$ approximately. Using the two orthonormal right-handed triad of vectors, $\{\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}\}$ in the

reference frame and {$\mathbf{W_1}$, $\mathbf{W_2}$, $\mathbf{W_3}$} in the spacecraft frame, the attitude matrix is estimated to be represented by Eq. 11 via TRIAD [8].

$$A=\left[w_1\, w_2\, w_3\right]\left[v_1\, v_2\, v_3\right]^T=\sum_{i=1}^{3} w_i v_i^T \tag{12}$$

Paul Davenport's Q Method was the first presented useful solution to Wahba's problem. It involves solving the loss function, L(A), and allows the expression of the loss function in terms of the quaternion as shown in Eq. 12. K(B) is the symmetric traceless matrix as represented by Eq. 14. Paul Davenport proved that the eigenvector corresponding to the largest eigenvalue of this matrix is equal to the quaternion. The matrix B is composed of the product between the weights, ai, and the measurement vectors in the body frame, **b**i, and reference frame, **r**i. A weight value of ai = 1 was chosen as suggested by Markley [8].

$$L\left(A\left(q\right)\right)=\lambda_0-\sum_{i=1}^{N} a_i b_i^T A\left(q\right) r_i \tag{13}$$

$$K\left(B\right)=\begin{bmatrix} B+B^T-tr\left(B\right)I_3 & z \\ z^T & tr\left(B\right) \end{bmatrix} \tag{14}$$

$$z=\begin{bmatrix} B_{23}-B_{32} \\ B_{31}-B_{13} \\ B_{12}-B_{21} \end{bmatrix} \tag{15}$$

$$B=\sum_{i=1}^{N} a_i b_i r_i^T \tag{16}$$

QUEST improves on the solution to the q-method and is the most widely used method since it can provide more frequent attitude computations [8]. However, the efficiency of QUEST is in not having to iteratively solve 4x4 matrices as required by Davenport's q-method. However, numerical analysts consider QUEST to be not as robust since it has to solve the characteristic equation for the eigenvalues [8]. Davenport's eigenvalue condition is expressed as Eq. 17 and the optimized loss function can be written as Eq. 18.

$$K(B)=\sum_{i=1}^{4}\lambda_i q_i q_i^T \tag{17}$$

$$L(A(q))=\lambda_0-\lambda_{max} \tag{18}$$

7 Project Design Methodology

An overview of the project goals were outlined in Project Overview, and background was given with regards to the applied attitude determination theory in Attitude Representations and Attitude Determination Methods. This section will cover the design specifications, including assumptions made and a holistic introduction to the Matlab function files that were written.

As defined previously, reference frames are an important aspect of attitude determination. There are two main reference frames that need to be defined for this project: the body frame and the inertial frame. The body frame is assumed to be aligned with the star tracker detector as it orbits the Earth. The reason why the body frame is assumed to be aligned with the axes of the star tracker is to simplify the problem. Otherwise, once star vectors in the camera's frame are obtained, an additional rotation from the star tracker's camera frame to the main spacecraft body frame would have been necessary. For completion, it will be assumed that the star tracker is orbiting the Earth at an altitude of 600 km in Low-Earth-Orbit (LEO). As the star tracker goes

along in its orbit, its field of view (FOV) encounters different regions of the night sky. The region of the sky that the camera sees is defined by a quaternion and will be replicated in image generation.

The inertial frame in this project will be the ECI frame, which was defined in Section 4.1. However, for the purposes of this project the star vectors in the reference frame will be assumed to be measured from the ECI frame. Table 7-1 summarizes the reference frames used in this project.

Table 7-1: Project design reference frames.

S/C Frame	The origin is at the center of the Star Tracker focal plane
	• +x is in bore sight of camera • y and z form the focal plane
Inertial Frame	Earth-Centered-Inertial J2000
	• +x along the vernal equinox • +z through the Earth's North Pole • +y completing the orthogonal system

The first part of this project will be called the Image Generation Truth Model and is purely dependent on the physics of the problem. A realistic image of the night sky will be generated by putting together numerous 25x25 pixel stars of varying magnitude, but not to exceed a magnitude of 6. This step will require three main Matlab functions. The first script will calculate the number of photons coming from any individual star and convert them to a digital

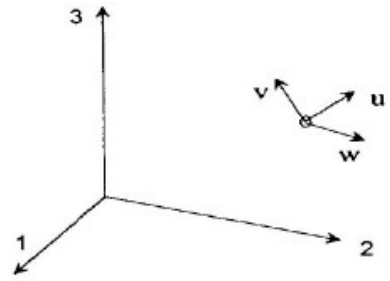
count number that a camera detector would normally interpret. The second will generate the star using a Gaussian distribution and output a matrix representation with those count values. Camera specifications will be discussed later. The third script will find star coordinates in pixel space, of where the individual stars belong, and place them in a large matrix that is then plotted as a star field image. Image generation makes use of the Hipparcos Star Catalog which will be covered in the next section. See Table 7-2 for a summary of Matlab function descriptions.

Table 7-2: Function files used in the Image Simulation Truth Model.

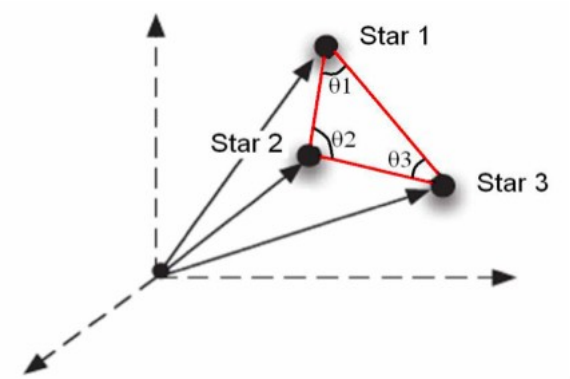
MATLAB Function with Inputs	Description
1. NumPhotons(Vmag,bv,specs)	• Vmag – visual magnitude of star • Bv – BV Color Index • Specs – camera's specifications The number of photons coming from an x-magnitude star is determined from this function. They are converted to electron counts in order to be interpreted by a camera detector.
2. $GenerateStar(Vmag, bv, specs, u, v)$	• Vmag – visual magnitude of star • Bv – BV Color Index • Specs – camera's specifications • u,v – pixel coordinates of stars Each star is generated by applying a Gaussian distribution, or brightness, to a 25x25 block given the inputs above. For each star, there are a certain number of photons or electron counts that should be displayed on the star image as defined by Function 1.

3. *GenerateStarField*(q)	• q – quaternion Takes the star images generated from function file 2 and places them in the FOV with correct coordinates.

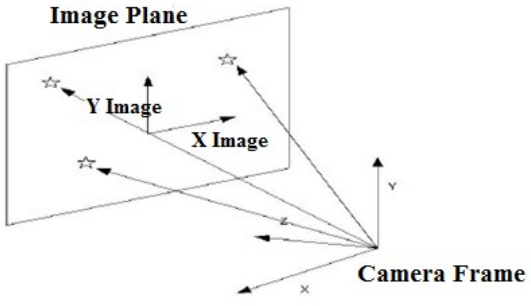
The second part of this project is Algorithm testing. This requires testing the generated star images in part one and running attitude determination on them. Attitude determination as defined previously, involves solving for the matrix that transforms vectors from one frame to another. The two reference frames are the star tracker frame and the ECI frame. The attitude is represented here by a quaternion. There are different methods that can solve for the quaternion given two sets of vectors. The one used in this project is Davenport's Q-Method. The difficulty with solving for the attitude is that the problem is over-determined when there are several vector measurements. For this reason, certain attitude determination methods have an advantage over others. Different methods include TRIAD, QUEST, and SVD. Fig. 7-1 shows the two main reference frames from which each set of star vectors is measured. Pixel space in comparison to the camera frame is where the photons of light get stored as electrons.



(a)



(b)



(c)

Figure 7-8: (a) 1-2-3 represents the ECI frame, u-v-w the spacecraft frame (b) stars measured from the camera reference frame (c) stars on the focal plane

8 Hipparcos Star Catalog

The star vector measurements used in image generation and then later in finding a star match in attitude determination are taken from the Hipparcos Star Catalog. In 1989 the European Space Agency launched the Hipparcos satellite and for 3 years it collected star light in efforts to map the celestial sky. As a result, the Hipparcos catalog [12] maps the 120,000 brightest stars in the night sky with high accuracy as well as dimmer stars with not as high accuracy. The catalog contains information about each star including star ID, BV-color index, right ascension and declination coordinates, and many other parameters. The most important values necessary for use in this project's code are the following:

- Hipparcos ID
- Henry Draper ID
- Right Ascension in Radians
- Declination in Radians
- BV Color Index
- Visual Magnitude

The Hipparcos ID and Hendry Draper ID (HDID) numbers are useful for differentiating each star. For a given identification number, the corresponding vector and magnitude among other characteristics can be looked up. The HDID numbers were not obtained from the same archive [13]. The right ascension and declination, in radians, define the location of the star in a spherical coordinate system. From these values the unit vector of the star can be obtained. The BV Color Index [14] is a parameter that gives the color and thus temperature of a star. This is useful for defining the radiation coming from a star. The visual magnitude is important for

defining a threshold of stars that the camera is limited to detect. The visual magnitude indicates how bright a star appears.

8.1 Star Properties

Stars in general can be categorized into spectral types. Each spectral type is defined by a temperature range, where the star's surface temperature categorizes it into either O, B, A, F, G, K, or M type. O denotes the hottest stars and M, the coolest stars. This classification scheme assumes that stars are black body radiators and re-emit all the incident electromagnetic radiation upon them. The radiation coming from stars is therefore only dependent on temperature and wavelength and is represented by Planck's radiation formula, Eq. 19.

$$I(\lambda, T) = \frac{2\pi h c^2}{\lambda^5 (e^{\frac{hc}{kT}} - 1)} \tag{19}$$

The temperature represents how hot a star is and how much radiation is being given off by the star. This allows for spectral type classification. Within each spectral type, stars are defined by their apparent magnitude or how bright they appear. In terms of apparent magnitude, a bright star will have a small value and a dim star will have a high magnitude. For reference, our sun has an apparent magnitude of Mv = -26.7. The Hipparcos catalog contains stars as dim as Mv = 11. The software defined in this project is assuming a star tracker that is able to detect stars as dim as visual magnitude of 6.

The amount of radiation or light that reaches a star tracker is dependent on the range of wavelengths that the detector is made to sensitize. In this project, the range is 400 – 800 nm and encompasses the visual spectrum. As mentioned previously, the stars will be assumed to be

black body radiators and emit a certain amount of flux dependent on its surface temperature. Using the first law of Thermal Radiation, Stefan-Boltzmann's Law [15], the energy density emitted from a star at some temperature can be calculated more simply as shown by Eq. 20. This can be derived using Planck's radiation formula by taking the limit as wavelength goes to infinity. Stefan-Boltzmann's Law represents a star's flux as being dependent only on surface temperature.

$$\sigma T^4 = 5.7\, x\, 10^{-8} \frac{W}{m^2 K^4} T^4 \tag{20}$$

It is also important to note that not all the blackbody radiation coming from a star will reach the camera's detector, especially since it is designed for only a certain range of wavelengths. In order to account for this, the Johnson V curve is used. The Johnson V Curve is a transmission curve that defines real radiation spectra as opposed to a black body spectrum [16].

8.2 Stored Catalog Variations

The Hipparcos star catalog is used as the main source for obtaining star properties. The Hipparcos catalog is available for free online and although it was originally put together in the early 1990's, updates are made every so often to account for the Earth's precession and other factors that may affect the location of stars as measured from the ICRF. A truncated catalog is also created to include only stars that are as dim as visual magnitude 6. This truncated catalog is saved as a structure in Matlab called mag6, and includes 4559 stars. This is a small quantity compared to the hundreds of thousands that are originally included in the catalog. This is the catalog that would be stored on the star tracker and is used in creating star images as well since the detector will ignore stars that have brightness dimmer than the threshold value of 6.

An additional catalog database is created for the purpose of star matching. A star match occurs when stars in the field of view are matched with stars in the star catalog. The star identification algorithm searches for bright regions and saves them as potential stars. After finding the angular distance to those stars as well as the spherical coordinates between the two closest neighbors, the algorithm creates a triangular feature (TF). This TF is then searched for in a separate database called the StarCatalogDatabase, which is saved as a structure in Matlab as well. Once a match is found between the chosen TF in the FOV and the TF in the StarCatalogDatabase, the corresponding HDID numbers of the stars are searched for in the mag6 truncated catalog. The star parameters are then found from there. Table 8-1 summarizes the different catalogs created for this project.

Table 8-3: Star Catalog Variations

Catalog Variation	Description
Original Hipparcos Catalog	• May not be stored in the star tracker • Contains the 120,000 brightest stars in the night sky and their properties with high accuracy
mag6	• Matlab structure containing only stars with visual magnitude 6 or lower • 4559 stars
StarCatalogDatabase	• Contains 4559 triangular features (TF) • A TF includes each star and its two closest neighbors

9 Part One: Image Generation Truth Model

As mentioned in Project Design Methodology, part one will focus on creating a truth model that can be used for algorithm testing in part two. The truth model is essentially an image

generation process that will output star fields that accurately represent portions of the night sky. Matlab functions were created, which collectively generate a star field image given a quaternion (Appendix C).

A quaternion is the input that defines what region of the sky the star tracker is pointing towards. Using the stars in the truncated Hipparcos catalog called mag6, the quaternion can be used to obtain the stars that are in the camera's field of view. Stars within the FOV are kept and are used to generate the star field. Using the physics of the stars as covered in Star Properties, and after taking into consideration camera specifications which are defined later, a star field is generated using coordinates of the stars within the FOV. An example of a star field produced with quaternion, q = [-0.9632 0.1788 0.1988 0.0279], is shown in Fig. 9.1. Note that the colors have been inverted in order to visually represent what the star field image looks like. Appendix E contains the raw image outputs without color inversion. Table 9-1 shows the time spent in image generation for a Monte Carlo run of 1000 trials. The average time to generate a star field image is 0.1375 seconds.

Table 9-4: Image Generation Results

Monte Carlo Trial Runs	Avg. Time Spent in Image Generation [sec]
1000	0.137518

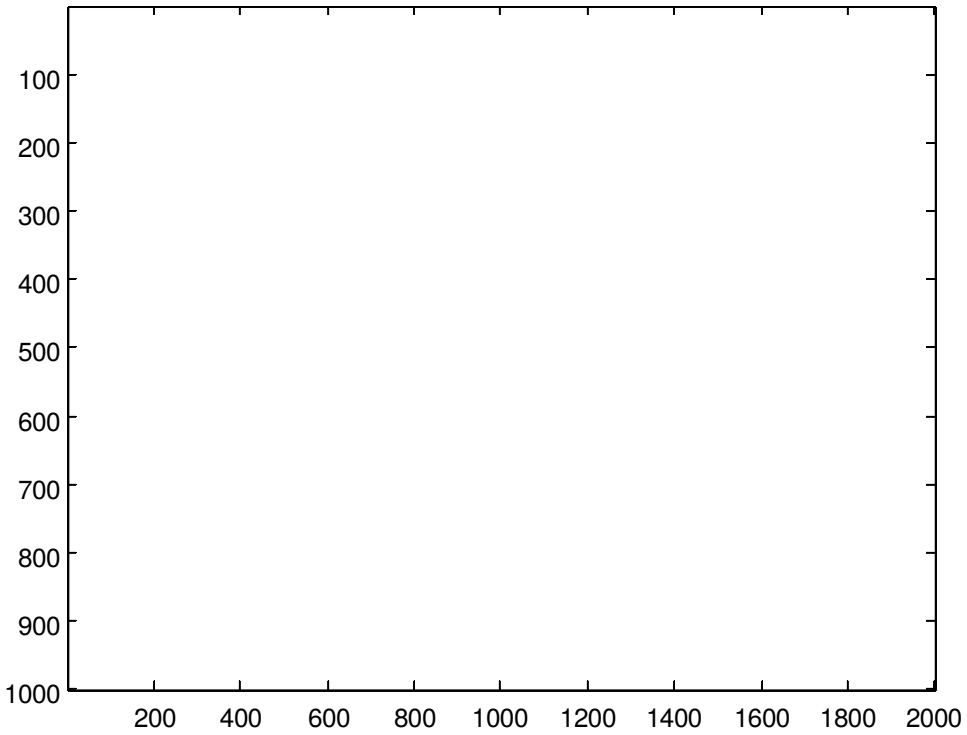


Figure 9-9: Star field generated with Matlab, Inverted Colors

9.1 Theory behind Image Generation

The following summarizes the steps and theory behind the function files listed in Table 6-1. As an overview, there are three main functions used in image generation: NumPhotons, GenerateStar, and GenerateStarField. The first function calculates the number of photons coming from a star and converts that number to electron counts. The second applies a Gaussian distribution to generate an individual star, and the third function puts together all the individual stars in the right location to represent a star field image.

The physics behind stars, in addition to the camera specifications, need to be taken into consideration in order to accurately represent how they would appear on a detector. Starlight hitting the focal plane of the camera is proportional to the around the amount of radiation that the

star gives off after taking into account the limitations of the camera in capturing 100% of this. Planck's radiation formula, as stated by Eq. 18, is integrated over the 400 – 800 nm wavelength range. The temperature is found by plugging in the B-V color index specified in the catalog into Eq. 20. Knowing the temperature and integrating Eq. 18 gives the flux coming from a star in Watts per meters cubed. Fig. 9-2 shows the power flux vs. wavelength plot of a star with magnitude 5 and temperate 7641 Kelvin.

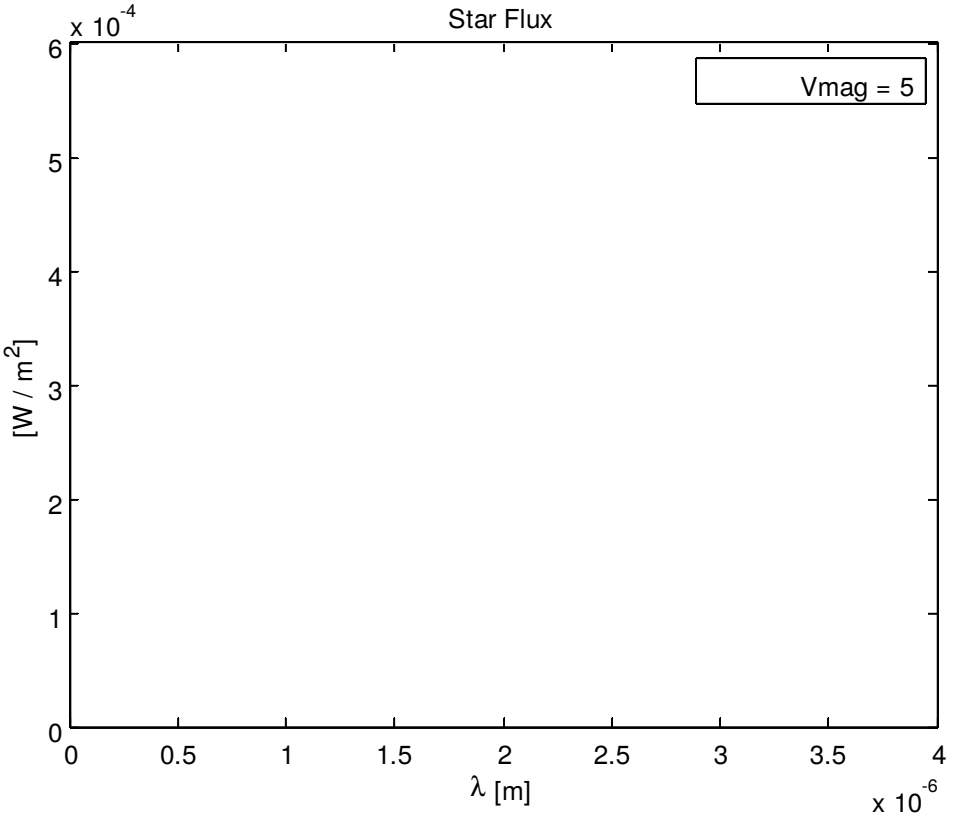


Figure 9-10: Power flux vs. Wavelength

The flux is then divided by photon energy, represented by Eq. 22, which outputs the number of photons coming from that star. In addition, a useful relation between flux and magnitude is also used in finding the flux coming from an x-magnitude star given some reference star. This is represented by Eq. 23, where m_1 and m_2 are the visual magnitudes and F_1

and F_2 are the fluxes. The reference star used in this project was the sun which has a known flux of 1300 W/m^3 and a visual magnitude of -26.7. Figure 9-3 shows the star's flux versus wavelength.

$$m_1 - m_2 = -2.5 \log\left(\frac{F_1}{F_2}\right) \tag{23}$$

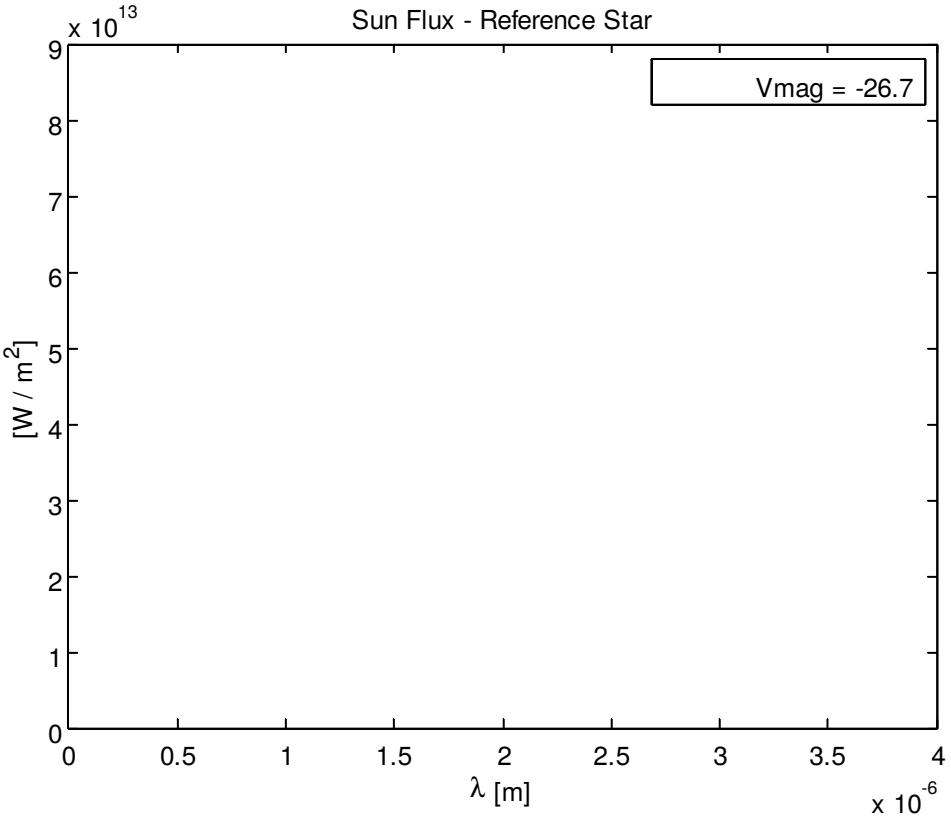


Figure 9-11: Sun's flux vs wavelength

However, the star tracker detector cannot capture all of the photons emitted from a star since some of the radiation gets lost as it travels through interstellar space. In addition, no detector is perfect and many factors affect its capability of capturing light including quantum efficiency and sources of noise. After taking into account quantum efficiency, although noise is not, the photons are converted into digital counts by using the relation presented in Eq. 23. First

the photons are converted into electrons by multiplying by the quantum efficiency, QE. This specifies how many photons actually get detected by the camera. Next the normalized value of electrons is divided by the quantum step equivalence (QSE) and multiplied by a digital gain (DG) which are both scaling factors.

$$T = 4600\left(\frac{1}{0.92(B-V)+1.7}+\frac{1}{0.92(B-V)+0.62}\right) \tag{21}$$

$$E = \frac{hc}{\lambda} \tag{22}$$

$$e- \text{¿ } photons * QE \tag{23-1}$$

$$\begin{gathered} e- \text{¿} \\ \text{¿} \\ \text{¿} \\ counts = \text{¿} \end{gathered} \tag{23-2}$$

Once the count values are known that should appear on the detector for a given star of magnitude and temperature, a Gaussian star is created. With a chosen PSF radius, or standard deviation, a Gaussian function is used to spread the light from a star onto a 25x25 pixel block. Equation 24 represents this function.

$$I(u,v) = \frac{1}{2\pi\sigma^2} e^{\frac{-1}{2\sigma^2}\left\|r(u,v)-r_{ref}(u,v)\right\|^2} \tag{24}$$

If the radius or standard deviation were 3, then the function would spread the star light with maximum peak at the center of the square and decrease from there. Using a pinhole model, it is important for the light coming from a star to disperse onto the focal plane over several pixels. Fig. 9-4 shows a pinhole model of a star tracker. The dashed line represents the bore sight of the

camera. In the center picture, the focal plane is visible where the starlight will fall. On the far right the lens is replaced with a pinhole. An example of an individual star generated is shown in Fig. 9-5.

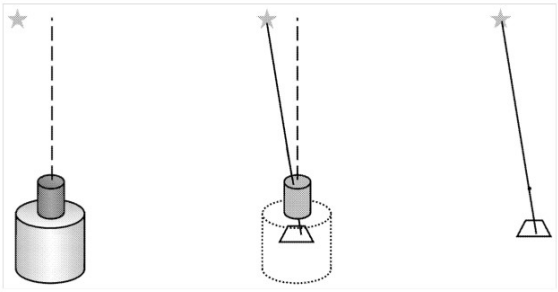


Figure 9-12: Pinhole model [2]

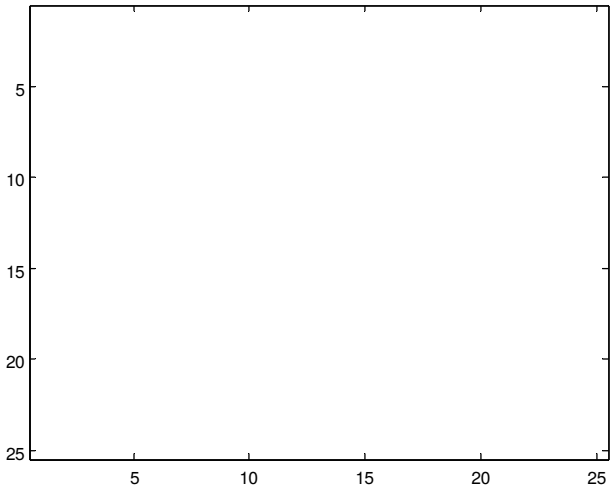


Figure 9-13: Gaussian Star Distribution created in Matlab

9.2 Camera Specifications

The camera's specifications affect the kind of images that are produced. A list of the specs used in image generation are listed Table 9-2.

Table 9-5: Camera Specs used in generating images.

Camera Spec	Value
QE: Quantum Efficiency	0.6
Int: Integration	0.1 [s]
D: Detector diameter	0.016 [m]
A: Detector Area	$\frac{\pi * D^2}{4} = ¿$ 2.0106e-4 [m^2]
Individual Star Size	25 x 25 pixels
pp: Pixel Pitch	2.5e-6 [m]
f: Focal Length	0.016 [m]
U: Detector Size in x-direction	2000 [pixels]
V: Detector Size in y-direction	1000 [pixels]
FOVx: Field of View in U-Direction	$2\tan^{-1}\frac{V * pp}{2f} = 0.1559$
FOVy: Field of View in V-Direction	$2\tan^{-1}\frac{U * pp}{2f} = 0.3100$
FOV: Half Angle Field of View	$\frac{\sqrt{FOV\,x^2 + FOV\,y^2}}{2} = 0.1735$
σ = Sigma	2
QSE: Quantum Step Equivalence	$\frac{N_{well}}{N_{DR}} = \frac{30000}{2^{12}} \cong 7.3$
# of Bits in Detector	12
Digital Gain	10

Depending on the focal length, there will either be high magnification or low magnification of stars. The $F_{\#}$ is sometimes used to specify the amount of magnification. The $F_{\#}$ is the ratio of the focal length to aperture diameter. The higher an $F_{\#}$, the higher the magnification.

9.3 Obtaining Star Vectors in the FOV

In order to generate the star image using the camera specifications, the star vectors that are in the field of view need to be obtained from the star catalog. The bore sight of the camera

corresponds to the vector component of the quaternion. The stars in the camera's FOV can be found using the simple cosine property represented by Eq. 25. If the dot product of the bore sight vector with the vector of the star in the ECI frame, obtained from the star catalog, is greater than or equal to the cosine of the FOV diagonal angle, then the star can be considered in the star tracker's field of view.

$$V_{bore} \cdot V_{ECI} \geq \cos(FOV) \tag{25}$$

$$V_{cam} = q\, V_{ECI} \tag{26}$$

The stars that are in the camera's FOV are unit vectors that were picked out from the truncated star catalog. The truncated catalog as specified in Section 8.2, contains stars with visual magnitude up to 6. This is the dimmest star that was included. From this truncated catalog, the star vectors that are in the camera's FOV are then transformed from the ECI frame to the camera frame using the quaternion and a simple multiplication as represented in Eq. 26. This was also discussed in more detail in Section 5. The vectors in the camera frame then need to be translated onto the detector's focal plane, where the photons of light get stored as electron counts a focal length distance, f, away.

10 Part Two: Algorithm Testing

Part two deals with running a star identification algorithm as well as an attitude determination algorithm on the images generated in part one. Figure 10-1 shows a block diagram of the process that the images go through with the corresponding Matlab "m" file right above each process. Star identification involves detecting regions of interest that could potentially be

stars. The attitude determination algorithm solves the lost-in-space problem by outputting the spacecraft's orientation in the form of a quaternion. Matlab script files were created to do this (Appendix 10). The processes presented in this section are those that a typical star tracker would go through while functioning. Part one was done in efforts to validate attitude determination methods.

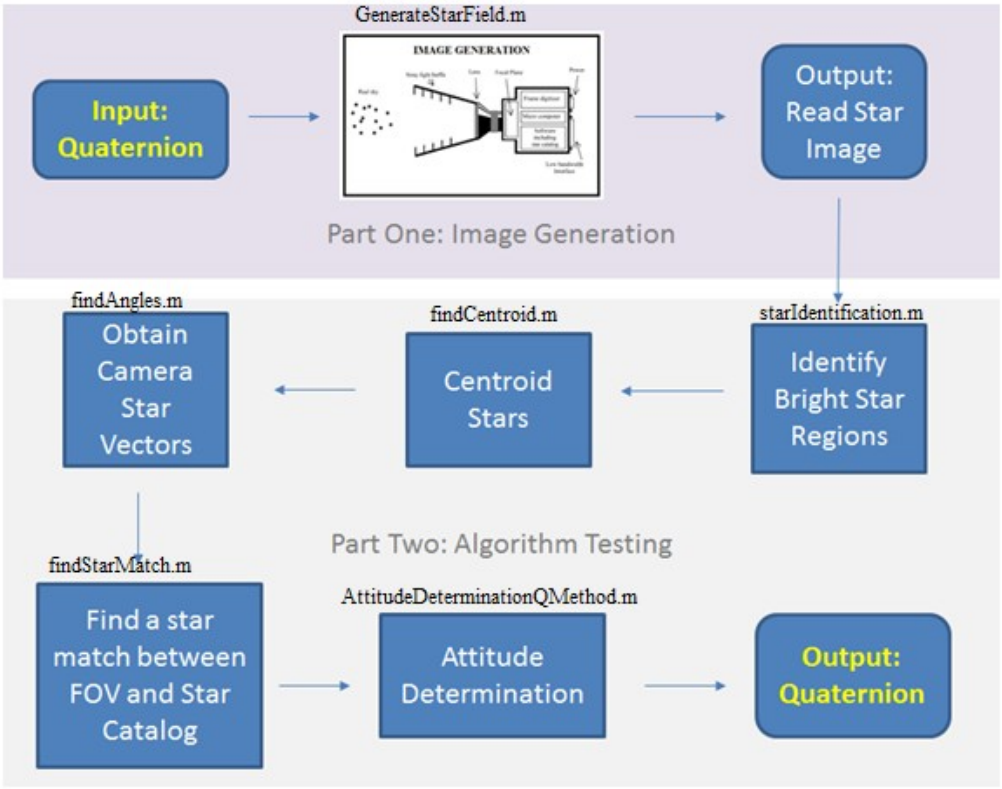


Figure 10-14: Star Tracker Code Block Diagram

10.1 Star Identification Algorithm

The first step in the algorithm testing process was to run star identification. The algorithm presented follows the pattern recognition method suggested by Liebe [17]. As satellites orbit the Earth its star trackers will point to different regions of the sky and identify stars their FOV. Identifying those stars requires an algorithm that searches for potential regions of interest (ROI).

Because part one essentially simulates images of the sky that a star tracker would see, those images were fed directly into the star identification algorithm (Appendix 10.1).

Identifying bright ROIs that could potentially be stars involves searching for pixels values that pass a certain threshold. This threshold is defined by the specifications of the camera. The star tracker design presented in this project assumes a maximum detection threshold of stars as dim as magnitude 6. Therefore, the threshold could be assigned as the maximum count value in a magnitude 6 star. In typical star trackers, noise is something that needs to be taken into account and thus there are other considerations that would affect the threshold. In this project, noise was not incorporated. The minimum threshold value was chosen as a count value of 5 since all pixels with a count value greater than 5 are guaranteed to be pixels in a star.

The star image is read into Matlab as a matrix of count values; each element corresponds to an intensity value in the range [0, 4095]. This range is dependent on the detector's bit count which is 12 for the ST presented here. A count value of 4095 corresponds to the highest intensity or brightness. This step size is determined by Eq. 27.

$$steps = 2^{bit} - 1 \tag{27}$$

For each ROI that the algorithm detects, the centroid is calculated in order to obtain subpixel accuracy. Equation 28 represents the centroid of a given region. The centroid locates the star in pixel space. In other words, it represents the coordinates of the star on the focal plane of the charge-coupled device (CCD), or the camera detector. The focal plane is where the photons of light get converted into a digital signal and saved as a count value. Once the coordinates are located in pixel space, the location of the star gets converted into a unit vector measured from the camera frame. This requires a simple translation as represented by Eq. 29. The values for Uc and

Vc correspond to the center of the overall star field, which represents a 1000 by 2000 pixel detector. Table 9-2 also lists other camera specs used in the equation. This translation is the reverse of what was done in image generation, where stars in the FOV were converted from camera space into pixel space. This step creates a list of potential star vectors as measured from the camera's reference frame.

$$C_x = \frac{\Sigma_{i=1}^{N} I(x,y) * x}{\Sigma I(x,y)}, C_y = \frac{\Sigma_{i=1}^{N} I(x,y) * y}{\Sigma I(x,y)} \tag{28}$$

$$\frac{V_{xcam}}{V_{zcam}} = (U_c - V_{xpix}) \frac{pp}{f} \tag{29-1}$$

$$\frac{V_{ycam}}{V_{zcam}} = (V_c - V_{ypix}) \frac{pp}{f} \tag{29-2}$$

$$V_{zcam} = \sqrt{1 - \left(\frac{V_{xcam}}{V_{zcam}}\right)^2 - \left(\frac{V_{ycam}}{V_{zcam}}\right)^2} \tag{29-3}$$

Star trackers are affected by different types of noise and aren't perfect detectors. Therefore the stars in the center of a star tracker's FOV appear brighter. This is due to the fact that stars closest to the rim are reflected the most and thus transmitted the least. A typical ST may ignore rim stars; Liebe's algorithm first has all stars discarded whose distance to the second closest neighbor is larger than that star's distance to the rim [17]. However, because no noise is accounted for in image generation and the stars appear the same regardless of where they are in the FOV, the rim stars are not discarded in this project.

In the star matching algorithm, the first star that is tried is the first star detected in the list of potential stars. This star is defined as the pivot star. A triangular feature (TF) is then created

which consists of the pivot star and its two closest neighboring stars. Three star parameters are saved in the TF: the angular distance to the first star, the angular distance to the second star, and the spherical angle between them. Figure 10-2 illustrates a TF, where only the angle θ_2 is saved.

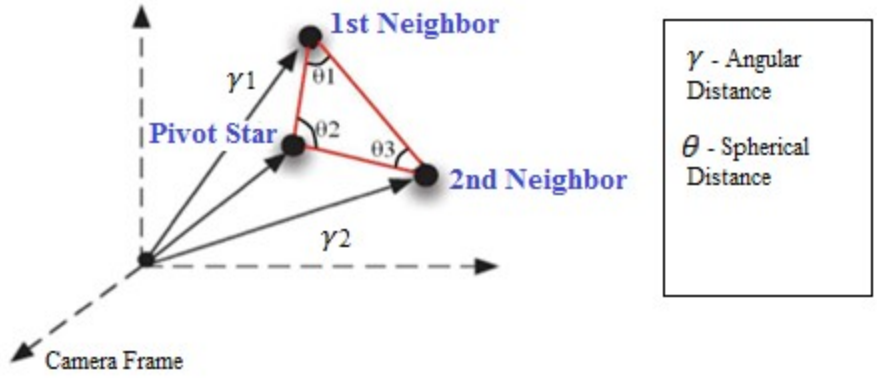


Figure 10-15: Three stars form a Triangular Feature (TF)

A separate catalog database is then searched, which is called StarCatalogDatabase and was outlined in Section 8.2. If the TF found in the FOV matches a TF saved in this separate database with angular distances within 1% difference, a star match is found and the HDID numbers corresponding to those stars are output. If the first TF tried does not output a star match, the next star in the list is chosen as the pivot star and a different TF is chosen. The HDID numbers are used in looking up the vectors of the stars in the mag6 truncated star catalog, also summarized in Section 8.2. Now two sets of vectors are known, one in the camera frame and the other in the inertial reference frame. These vectors are then fed into the attitude determination problem. Figures 10-3 and 10-4 show a star image after stars in the FOV have been centroided. The red crosses pinpoint the centroids of each star. Colors have been inverted in order to more

clearly represent the star image. Figure 10-4 shows the triangular feature. Images without an inverted colormap are included in Appendix E.

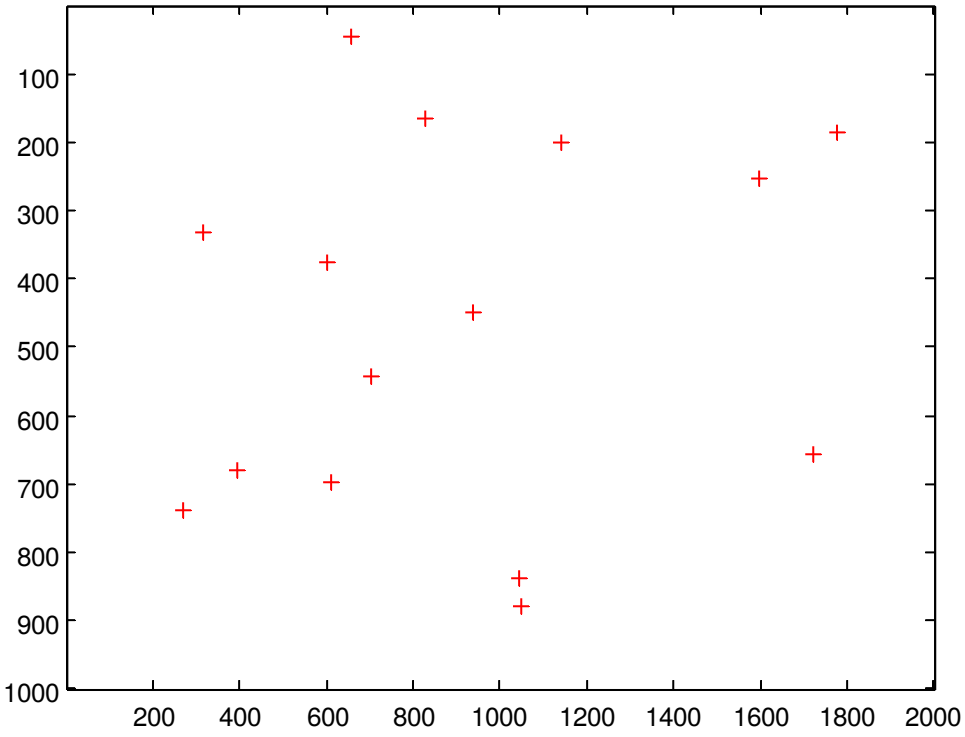


Figure 10-16: Image after Star Identification

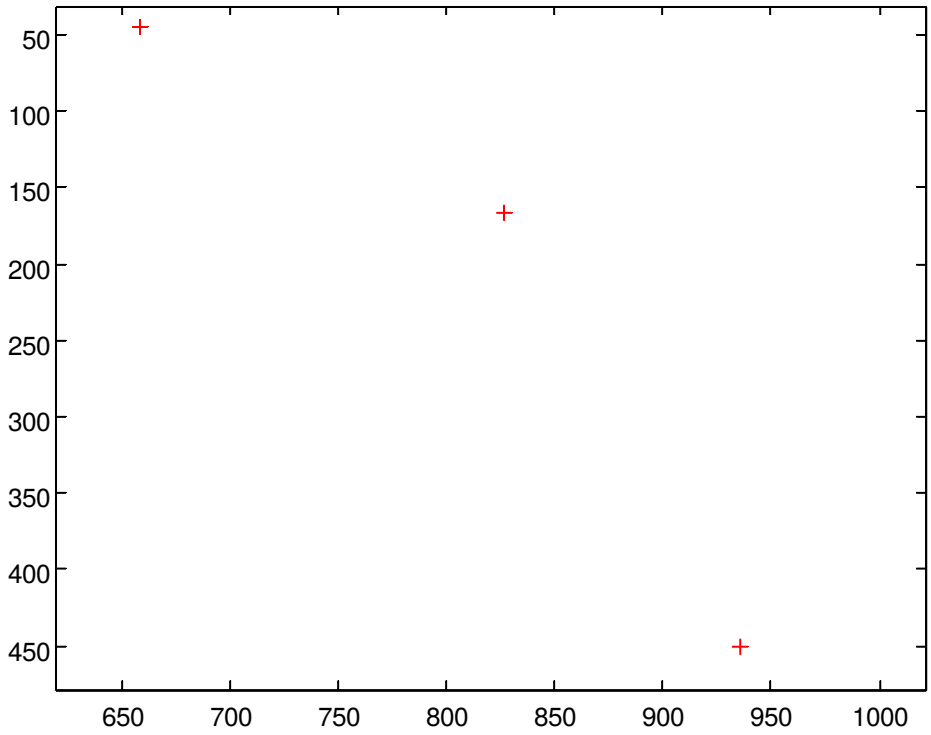


Figure 10-17: Triangular Feature after Star Matching

10.2 Attitude Determination

There are many attitude determination methods that can solve for the quaternion or attitude matrix as covered in Section 6. The simplest one is TRIAD which solves for an attitude matrix using two vector measurements. An improvement to the TRIAD solution was presented by Grace Wahba in 1965 [8]. Wahba's problem is essentially an optimization problem where the aim is to minimize the loss function as represented previously by Eq. 12 and restated here. Minimizing the loss function yields the optimal least squares solution of the attitude matrix.

$$L(A)=\frac{1}{2}\sum_{i=1}^{N} a_i \left\| b_i - A r_i \right\|^2 \tag{12}$$

Davenport's Q Method and Quaternion Estimator (QUEST) are two methods that can solve Wahba's problem and in effect minimize the loss function, L(A). The method of attitude determination used in this project is Davenport's Q Method [8]. This method involves solving for the symmetric, traceless matrix K restated here by Eq. 14. It has been proven that the attitude which represents the rotation of the star vectors from the reference frame, $\mathbf{r_i}$, to the star tracker frame, $\mathbf{b}_i$, is given by the largest eigenvector of the matrix K. A Matlab Script file was created that solves the eigenvalue problem and outputs the estimated quaternion using Q Method (Appendix D.9).

$$K(B)=\begin{bmatrix} B+B^T-tr(B)I_3 & z \\ z^T & tr(B) \end{bmatrix} \tag{14}$$

$$z=\begin{bmatrix} B_{23}-B_{32} \\ B_{31}-B_{13} \\ B_{12}-B_{21} \end{bmatrix} \tag{15}$$

$$B=\sum_{i=1}^{N} a_i b_i r_i^T \tag{16}$$

The same quaternion that was used to generate the image is the true quaternion which should match the quaternion output after going through attitude determination, the estimated quaternion. An example of star vectors in the body frame are given by Eq. 30 and the star vectors in the reference frame are given by Eq. 31. The estimated quaternion, output from attitude determination and represented by Eq. 32, will transform the "inertialVectors" into "starVectors." This is essentially rotating from the inertial reference frame – ECI , to the body frame – the star tracker. Sometimes a negative of the expected quaternion is output. A negative quaternion

represents a vector in the opposite direction, however the rotation is still the same. A rotation of 0 degrees is equivalent to a rotation of 360 degrees.

$$starVectors = \begin{bmatrix} 0.0826 & 0.0753 & 0.1128 \\ 0.0670 & 0.0343 & 0.0255 \\ 0.9943 & 0.9965 & 0.9932 \end{bmatrix} \tag{30}$$

$$inertialVectors = \begin{bmatrix} -0.2866 & -0.2982 & -0.2636 \\ 0.4176 & 0.3877 & 0.3790 \\ 0.8623 & 0.8722 & 0.8871 \end{bmatrix} \tag{31}$$

$$q = \begin{bmatrix} -0.9632 \\ 0.1788 \\ 0.1988 \\ 0.0279 \end{bmatrix} \tag{32}$$

One way of verifying whether the estimated attitude is accurate is by finding an attitude error of the output. This can be found by making use of the orthonormal nature of the rotation matrix and of the quaternion through the property expressed by Eq. 33. The principal Euler angle, θ, will ideally be as close to 0 degrees, or 360 degrees, as possible.

$$\acute{q} \times q^{-1} = \delta q = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) \\ \acute{e}\sin\left(\frac{\theta}{2}\right) \end{pmatrix} \tag{33}$$

10.3 Algorithm Testing Results

In order to validate the scripts created in this part of the project, various tests were completed. A Monte Carlo simulation was performed on the code for 1000 trial runs beginning with image generation and ending with outputting the attitude quaternion result. Out of the 1000

runs, there were 9 cases where a star match was not found. This results in a 99.1 % efficiency in star identification. In some images generated, there may only be 2 stars identified in the FOV and since three are required to form a triangular feature, a star match is not found. Other factors that could have affected this include edge stars that are not fully within the FOV and so their centroids were not resolvable. The attitude error was also found for the Monte Carlo simulation. The average attitude error was 0.001 degrees and 359.999 degrees, where an error of 0 or 360 degrees represents a perfect estimate of the true attitude. Table 10-1 shows the results of the Monte Carlo simulation. An attitude error of exactly 0 degrees or 360 degrees represent zero error due to the fact that a negative quaternion is equal to its positive. The standard deviation values are presented as well.

Table 10-2 summarizes the amount of time spent in both Matlab algorithms – Star Identification and Attitude Determination. Run time for finding a star match is also included. The amount of time spent in each script is dependent on a few parameters including the efficiency of the algorithm, but also on camera specs like integration time and computing power available. Star trackers are extremely expensive due to the computation power required for its performance. The average loss function after the Monte Carlo run is also presented which is an average of 0.000161 and represents a very accurate estimate of the quaternion. The closer the value is to zero, the smaller the difference between the expected and estimated attitude and the more accurate the result.

Table 10-6: Monte Carlo average error and standard deviation

0 degrees		**360 degrees**	
$\theta avg_{att_{error}}$	θ_{stdev}	$\theta avg_{att_{error}}$	θ_{stdev}
0.0010	0.0073	359.9994	0.0053

Table 10-7: Monte Carlo Simulation Summary

Monte Carlo Runs	Avg. # Stars in FOV	Avg. Time in Star Identification [sec]	Avg. Time in Finding Star Match [sec]	Avg. Time in Attitude Determination [sec]	Avg. Loss Function Value
1000	16.3	1.903358	0.054212	0.000095	0.000161

11 Applications and Future Work

The truth model designed in this project can be applied towards a Hardware in the Loop (HITL) simulation, where the star tracker algorithm can be tested over a spacecraft's orbit in time. A Two Line Element (TLE) can be propagated, assuming the spacecraft holds an LVLH position at 0, 0, 0 degrees. The quaternion in this case is constantly changing over the course of the orbit and when plotted versus time would yield a sinusoidal curve.

Another extension of this project is to take real pictures of the night sky with a telescope. Using a long exposure and integration time will allow the telescope to get refined images of the night sky. The same algorithms presented in this project can be used on those images to see if they are robust in handling noise.

Future work includes improving the image generation process to include noise factors. This will create even more useful applications for testing star tracker algorithms. As a result, when noise is incorporated the star identification algorithm will need to be updated. This will yield a better method for star matching that can catch errors atypical of a star tracker in orbit. In effect a

better star tracker algorithm can be created that is more efficient and susceptible to overcoming noise factors.

One limitation of the truth model created here is that noise is not incorporated. In addition, there is a case where a quaternion will output a star image with less than three stars in the FOV. Because a TF requires three stars in order to find a star match, this algorithm will not output an attitude. It may be that the stars are too dim for the detector to detect, with the camera specs defined in this project. This may also suggest that the method of star identification is not robust enough. Although there are limitations, the overall star tracker algorithm presented met the expectations of the proposal and fulfilled the objective with high accuracy.

References

[1] Liebe, Carl C., "Star Trackers for Attitude Determination,"*IEEE AES Systems Magazine,* Department of Electrophysics, Technical University of Denmark, Copenhagen, Denmark, 1995.

[2] Liebe, C. C., "Accuracy Performance of Star Trackers—A Tutorial," *IEEE Transactions on Aerospace and Electronic Systems,* T-AES/38/2/11444, Vol. 38, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 2002, pp. 587-599.

[3] Eisenman, A.R., Liebe, C.C., and Joergensen, J.L., "New generation of autonomous star trackers," Sensors, Systems, and Next-Generation Satellites, Vol. 3221, SPIE - The International Society for Optical Engineering, USA, 1997, pp. 524-35.

[4] Doug Sinclair URL: < http://www.sinclairinterplanetary.com/startrackers>.

[5] Enright, J., Sinclair, D., Grant, Cordell C., McVittie, G., Dzamba, Tom, "Towards Star Tracker Only Attitude Estimation," *24th Annual AIAA/USU Conference on Small Satellites*, SSC10-X-3, Toronto, CAN, 2010.

[6] Spratling, B. B., Mortari, D., "A Survey on Star Identification Algorithms," Algorithms Journal, 2009.

[7] Kandiyil, R., "Attitude Determination Software for a Star Sensor," M.S. Thesis, University of Wurzburg, Wurzburg, Germany, 2009.

[8] Markley, F. L., John, C. L, *Fundamentals of Spacecraft Attitude Determination and Control*, New York, USA, 2014.

[9] Epoch (astronomy), Wikipedia, URL: http://en.wikipedia.org/wiki/Epoch_(astronomy) [cited 2 August 2014].

[10] Bate, R. B., Mueller, D. D., White, J. E., *Fundamentals of Astrodynamics*, New York, USA 1971.

[11] Sidi, Marcel J., *Spacecraft Dynamics & Control, Appendix B*, 1st ed., Cambridge, UK, 2000.

[12] Hipparcos Catalog, "Hipparcos, the new reduction of the Raw data," URL: http://cdsarc.u-strasbg.fr/viz-bin/Cat?cat=I%2F311&target=readme& [cited 9 December 2014].

[13] The HYG Database, "The Astronomy Nexus," URLS: http://www.astronexus.com/hyg [cited 15 January 2015].

[14] Color Index, Wikipedia, URL: http://en.wikipedia.org/wiki/Color_index [cited 10 January 2015].

[15] Bennett, Donahue, Schnieder, Voit, *Stars, Galaxies, and Cosmology*, 2010.

[16] Johnson V Transmission Curve, “Astronomical spectra, filters, and magnitudes,” URL: http://spiff.rit.edu/classes/phys440/lectures/filters/filters.html [cited 10 January 2015].

[17] Liebe, C. C., “Pattern Recognition of Star Constellations for Spacecraft Applications,” *IEEE AES Systems Magazine,* Jan 1993.

Appendix A: Star Catalog Matlab Files

A.1: catalog_mag6.m

```
% Description:  Loads the Hipparcos Star Catalog as a mat file and deletes
%               all entries for stars that have a magnitude > 6
% Inputs:       none
% Outputs:      *Saves* hip_data.mat file with updated 'new' structure

load('hip_data.mat')

i = 1;
while (i <= length(new.mag))

    if (new.mag(i) > 6)
        new.hip(i) = [];
        new.ra(i) = [];
        new.dec(i) = [];
        new.pmRa(i) = [];
        new.pmDec(i) = [];
        new.mag(i) = [];
        new.parallax(i) = [];
        new.bv(i) = [];
        new.vi(i) = [];
    else
        i = i + 1;
    end

end
```

A.2: HipID2HendryDraperID.m

```
% Description:  Gets the Henry Draper ID # (HDID) from csv file and adds
%               to the mag6 structure
% Inputs:       none
% Outputs:      *Saves* mag6_HDID.mat file with updated 'new' structure

function [] = HipID2HenryDraperID()

fid = fopen('hygdata_v3.csv','r');
load('mag6.mat');
mag6.HDID = [];
i = 1;
while (1)
    tline = fgetl(fid);
    if tline == -1
        break;
    end

    if sscanf(tline,'%*d,%d,%*d') == mag6.hip(i)
        if isempty(sscanf(tline,'%*d,%*d,%d'))
            mag6.HDID(i,1) = 0;
```

```
        else
            mag6.HDID(i,1) = sscanf(tline,'%*d,%*d,%d') ;
        end
        i = i + 1;
    else
        continue
    end
    if i == 166,
        mag6.HDID(i,1) = 6595
        i = 167
    end
    if i == 602,
        mag6.HDID(i,1) = 23978
        i = 603
    end
    if i == 901,
        mag6.HDID(i,1) = 34310
        i = 902
    end
    if i == 1115,
        mag6.HDID(i,1) = 41695
        i = 1116
    end
    if i == 2529,
        mag6.HDID(i,1) = 114613
        i = 2530
    end
    if i == 3734,
        mag6.HDID(i,1) = 182416
        i = 3735
    end
    if i == 3868,
        mag6.HDID(i,1) = 189944
        i = 3869
    end
    if i == 4317,
        mag6.HDID(i,1) = 213179
        i = 4318
    end
    if i > numel(mag6.hip)
        break
    end
end

fclose(fid);
save('mag6_HDID.mat');

end
```

A.3: RaDec2_Veci.m

```
% Description:  Converts RA, Dec to unit vectors in the ECI frame, adds
%               to the mag6 structure and saves as a new mat file
% Inputs:       none
% Outputs:      *Saves* mag6_ECI.mat file with updated 'new' structure
```

```
function RaDec2_Veci()

load('mag6_HDID.mat')
mag6.ECIcoord = [];
for i = 1:numel(mag6.ra)

    mag6.ECIcoord(i,1) = cos(mag6.dec(i))*cos(mag6.ra(i));
    mag6.ECIcoord(i,2) = cos(mag6.dec(i))*sin(mag6.ra(i));
    mag6.ECIcoord(i,3) = sin(mag6.dec(i));

end

save('mag6_ECI.mat');

end
```

Appendix B: Constants Matlab Files

B.1: camera_specs.m

```
% Description:  Provides camera's specifications used in generating images
% Inputs:       none
% Outputs:      specs structure


function specs = camera_specs()

    specs.QE = 0.6;           % Quantum Efficiency
    specs.Int = 0.1;          % [sec]
    specs.A = (pi*0.016^2)/4; % [m^2]
    specs.Nsize = 25;         % [pixels] Individual Star Block Size: 25x25
    specs.pp = 2.5e-6;        % [m] Pixel Pitch
    specs.f = 0.016;          % [m] Focal Length
    specs.U = 2000;           % [pixels] Number of pixels vertically
    specs.V = 1000;           % [pixels] Number of pixels in horizontally
    specs.FOVx = 2*atan((specs.V*specs.pp/(2*specs.f)));
% [rad] Horizontal Field of View
    specs.FOVy = 2*atan((specs.U*specs.pp)/(2*specs.f));
% [rad] Vertical Field of View
    specs.FOV = sqrt(specs.FOVx^2 + specs.FOVy^2)/2;
% [rad] Diagonal Field of View
    specs.sigma = 2;
% standard deviation, how much the star spreads over pixels
    specs.QSE = 7.3;
% Quantum Step Equivalence = full well capacity/dynamic range
    specs.bits = 12;          % Bit size
    specs.DG = 10;            % Digital Gain scale factor

end
```

B.2: constants.m

Appendix C: Image Generation Matlab Files

C.1: generate_random_quaternion.m

```
% Description:  Generates a random quaternion
% Inputs:       none
% Outputs:      4x1 quaternion, 1st element is scalar
% Fns used:     optionally use generate_random_ProperRealOrthogonal_matrix()

function [q] = generate_random_quaternion()

%     angle = rand(1)*pi;
%     A = generate_random_ProperRealOrthogonal_matrix();
%     [vec, lambda] = eig(A);
%     e = vec(:,1);
%     q = [cos(angle/2); e(1)*sin(angle/2); e(2)*sin(angle/2);
e(3)*sin(angle/2)];

e = rand(3,1);
e = e/norm(e);
theta = rand*2*pi;
q = [cos(theta/2);e(1)*sin(theta/2);e(2)*sin(theta/2);e(3)*sin(theta/2)];

end
```

C.2: get_Vbohr.m

```
% Description:  Obtain the bore sight vector of the camera
% Inputs:       quaternion
% Outputs:      Bore sight of the camera, along Z axis.

function Vbohr = get_Vbohr(q)

q_3x3 = [q(2)^2-q(3)^2-q(4)^2+q(1)^2 2*(q(2)*q(3)+q(4)*q(1)) 2*(q(2)*q(4)-
q(3)*q(1));...
    2*(q(3)*q(2)-q(4)*q(1)) -q(2)^2+q(3)^2-q(4)^2+q(1)^2
2*(q(3)*q(4)+q(2)*q(1));...
    2*(q(4)*q(2)+q(3)*q(1)) 2*(q(4)*q(3)-q(2)*q(1)) -q(2)^2-
q(3)^2+q(4)^2+q(1)^2]; %bore sight

Vbohr = q_3x3(3,:);

end
```

C.3: Veci_2Vcam.m

```
% Description:  Converts a vector in the ECI frame to a vector in the
%               Camera frame
% Inputs:       quaternion, Vector in ECI
% Outputs:      Vector in Camera frame

function [Vcam] = Veci_2Vcam(q,Veci)

q_3x3 = [q(2)^2-q(3)^2-q(4)^2+q(1)^2 2*(q(2)*q(3)+q(4)*q(1)) 2*(q(2)*q(4)-
q(3)*q(1));...
    2*(q(3)*q(2)-q(4)*q(1)) -q(2)^2+q(3)^2-q(4)^2+q(1)^2
2*(q(3)*q(4)+q(2)*q(1));...
    2*(q(4)*q(2)+q(3)*q(1)) 2*(q(4)*q(3)-q(2)*q(1)) -q(2)^2-
q(3)^2+q(4)^2+q(1)^2]; %bore sight

Vcam = q_3x3*(Veci');

end
```

C.4: CameraStarCoordinates.m

```
% Description:  Obtains all the star coordinates in the camera's FOV and
%               converts them from ECI to camera frame
% Inputs:       quaternion, camera specs
% Outputs:      mag6bore structure with star vector parameters including
%               Hipparcos ID, RA, Dec, mag, bv, HDID, ECI coordinates
% Fns used:     get_Vbohr, Veci_2Vcam

function mag6bore = CameraStarCoordinates(q,specs)

FOV = specs.FOV;
load('mag6_ECI.mat')
Vbore = get_Vbohr(q);
mag6bore = [];
k = 1;

for i=1:length(mag6.hip)
    j = (mag6.ECIcoord(i,:));
    if dot(j,Vbore) >= cos(FOV),
        mag6bore.hip(k,:) = mag6.hip(i,:);
        mag6bore.ra(k,:) = mag6.ra(i,:);
        mag6bore.dec(k,:) = mag6.dec(i,:);
        mag6bore.mag(k,:) = mag6.mag(i,:);
        mag6bore.bv(k,:) = mag6.bv(i,:);
        mag6bore.HDID(k,:) = mag6.HDID(i,:);
        mag6bore.ECIcoord(k,:) = mag6.ECIcoord(i,:);
        k = k + 1;
    else
        continue
    end
end

for i = 1:length(mag6bore.ECIcoord)
```

```
    Veci = mag6bore.ECIcoord(i,:);
    Vcam = Veci_2Vcam(q,Veci);
    mag6bore.CameraCoord(i,:) = Vcam';
end
end
```

C.5: Camera2PixelSpace.m

```
% Description:  Converts the coordinates of stars from camera frame to pixel
space
% Inputs:       camera specs, x,y,z coordinates in camera frame
% Outsputs:     u,v coordnates in Pixel Space

function [u,v] = Camera2PixelSpace(specs,vx,vy,vz)

f = specs.f;
pp = specs.pp;
U = specs.U;
V = specs.V;

uc = (U/2)+1;
vc = (V/2)+1;

d = f/vz;
u = uc - (vx*d)/pp;
v = vc - (vy*d)/pp;

end
```

C.6: PixelSpaceCoordinates.m

```
% Description:  Calculates the pixel space coordinates of the stars in the
%               star tracker's FOV
% Inputs:       quaternion, camera specs, mag6bore structure
% Outsputs:     Updated mag6bore structure with pixel space coordinates
% Fns used:     Camera2PixelSpace

function mag6bore = PixelSpaceCoordinates(q,specs,mag6bore)

listCam = mag6bore.CameraCoord;
n = length(listCam);
for i = 1:n
    [u,v] = Camera2PixelSpace(specs,listCam(i,1),listCam(i,2),listCam(i,3));
    mag6bore.PixelCoord(i,1) = u;
    mag6bore.PixelCoord(i,2) = v;
end

end
```

C.7: johnsonVcurve.m

```
% Description:  Johnson V filter transmission curve for real radiation spectra
%               (ftp://obsftp.unige.ch/pub/mermio/filters/ph01.Vj)
%               Transmission in Unity
% Inputs:       none
% Outputs:      range of wavelengths, Johnson V transmission curve for visual
magnitudes

function [lambda,JVcurve] = johnsonVcurve()

lambda = (5:5:3500)*1e-9;
top = zeros(89,1);
bottom = zeros(559,1);
A=[0.002
0.005
0.005
0.009
0.012
0.016
0.023
0.039
0.1
0.213
0.371
0.548
0.705
0.831
0.916
0.972
0.998
1
0.984
0.954
0.916
0.872
0.826
0.775
0.722
0.668
0.613
0.559
0.503
0.45
0.399
0.346
0.297
0.251
0.209
0.169
0.135
0.104
0.081
0.063
0.049
```

```
0.042
0.037
0.035
0.03
0.028
0.023
0.019
0.016
0.014
0.012
0.009];
JVcurve = [top; A ;bottom];
end
```

C.7: ref_star.m

```
% Description:  Calculates the flux coming from the sun which is used as the
%               reference star for calculating photon numbers in NumPhotons
% Inputs:       none
% Outsputs:     sun's flux
% Fns used:     johnsonVcurve, Optionally check Wein's Displacement Law

function sun_flux = ref_star()

    h = 6.626*10^-34;             % [J s] Planck's constant
    k = 1.38065*10^-23;           % [J/K] Boltzmann's constant
    c = 2.997*10^8;               % [m/s] speed of light
    sunT = 5777;                  % [K]
    AU = 149597871000;            % [m] Astronomical Unit
    Rs = 695800000;               % [m] Sun's radius
    dist = (AU/Rs)^2;             % http://maths.ucd.ie/met/msc/fezzik/Phys-
Met/Ch04-2-Slides.pdf
    [lambda, JV] = johnsonVcurve();

    p = ((2*pi*h*c^2)./(lambda.^5.*(exp((h*c)./(lambda.*k*sunT))-1)))';
    flux = trapz(lambda,p.*JV);   % from sun's photosphere [W/m2]
    sun_flux = (flux/dist);       % from Earth's surface [W/m2/s]
    figure
    plot(lambda,p)
    title('Sun Flux')
    xlabel(' \lambda [m]')
    ylabel(' [W / m^2]')          % WeinsLaw(lambda,p.*JV,sunT);

end
```

C.9: BV2Temp.m

```
% Description:  Get the temperature of a star from its B-V color index
% Inputs:       B-V color index
```

```
% Outputs:      Temperature of a star

function T = BV2Temp(bv)

    T = 4600*( 1/(.92*bv + 1.7) + 1/(.92*bv+.62) );

end
```

C.10: NumPhotons.m

```
% Description:  Calculates the number of photons coming from an X-Magnitude
%               star, optionally plots the flux vs wavelength
% Inputs:       Visual magnitude, BV color index, camera specs
% Outputs:      range of wavelengths, number of photons
% Fns used:     johnsonVcurve

function [lambda,photons] = NumPhotons(Vmag,bv,specs)

    [h,k,c,sunT,solarConstant] = constants();
    Int = specs.Int;
    A = specs.A;
    [lambda,JV] = johnsonVcurve();
    T = BV2Temp(bv);

    I = (((2*pi*h*c^2)./(lambda.^5.*(exp((h*c)./(lambda.*k*T))-1))))';
    Sint = solarConstant;                   % [W/m2] *JV
    Iint = trapz(lambda,I.*JV);             % [W/m2]
    magn = 2.512^(-26.7-Vmag);
    E = ((h*c)./lambda)';

    starFlux = (Sint/Iint)*magn.*I;         % [W/m2/s]
    photonFlux = ((Sint/Iint)*magn.*I)./E; % [photons/m2/s]
    photons = photonFlux*Int*A;             % [photons]
    sum(starFlux);

    sun = (((2*pi*h*c^2)./(lambda.^5.*(exp((h*c)./(lambda.*k*sunT))-1))))';

%     figure(1)
%     plot(lambda,sun,'b')
%     title('Sun Flux - Reference Star')
%     legend('Vmag = -26.7')
%     xlabel('\lambda [m]'),ylabel('[W / m^2]')
%
%     figure(2)
%     plot(lambda,starFlux,'r')
%     title('Star Flux')
%     legend(['Vmag = ',num2str(Vmag)])
%     xlabel('\lambda [m]'),ylabel('[W / m^2]')

%     WeinsLaw(lambda,starFlux,T);

end
```

C.11: GenerateStar.m

```
% Description:  Creates a star image given visual magnitude, optionally
%               plot the star in its own figure
% Inputs:       Visual magnitude, BV color, camera specs, u,v coordinates
%               of star in pixel space
% Outputs:      number of counts from the electrons that get stored  on the
%               detector, ul,vl coordinates of the upper left pixel to be
%               placed on larger star image field
% Fns used:     NumPhotons

function [counts,ul,vl] = GenerateStar(Vmag,bv,specs,u,v)

Nsize = specs.Nsize;
QE = specs.QE;
sigma = specs.sigma;
QSE = specs.QSE;
bits = specs.bits;
DG = specs.DG;

R_l = [round(v - Nsize/2); round(u - Nsize/2)];
vl = R_l(1);
ul = R_l(2);
R_rel = [v; u] - R_l;

K = zeros(Nsize,Nsize);
for i = 1:1:Nsize
for j = 1:1:Nsize
    K(i,j) = (1/(2*pi*sigma^2))*exp((-1/(2*sigma^2))*norm([i;j] - R_rel)^2);
end
end

sum1 = sum(K(:));
[lambda,p] = NumPhotons(Vmag,bv,specs);
photons = trapz(lambda,p);
e = photons*QE;
K = e*(K/sum1);
C = (K/QSE)*DG;
counts = min(max(round(C),0),2^bits-1);

% figure,
% imagesc(counts,[0 2^bits-1]);
% colormap(gray)

end
```

C.12: GenerateStarField.m

```
% Description:  Plots a star field in a figure given a quaternion
% Inputs:       quaternion
% Outputs:      Img matrix with count values, Stars structure with
%               parameters including: pixel coordinates, Visual magnitude,
%               camera coordinates, HDID number
% Fns used:     CameraStarCoordinates, PixelSpaceCoordinates, GenerateStar

function [Img,Stars] = GenerateStarField(q)

specs = camera_specs();
mag6bore = CameraStarCoordinates(q,specs);
star_parameters = PixelSpaceCoordinates(q,specs,mag6bore);

listUV = star_parameters.PixelCoord;
Vmag = star_parameters.mag;
bv = star_parameters.bv;
U = specs.U;
V = specs.V;
Nsize = specs.Nsize;
Img = zeros(V,U);

Stars = [];
Stars.UV = listUV;
Stars.coord = star_parameters.CameraCoord;
Stars.mag = Vmag;
Stars.HDID = star_parameters.HDID;

for i = 1:length(listUV)

    vmag = Vmag(i);
    BV = bv(i);
    u = listUV(i,1);
    v = listUV(i,2);
    [Inorm,ul,vl] = GenerateStar(vmag,BV,specs,u,v);

    vcoord = vl+1:vl+Nsize;
    ucoord = ul+1:ul+Nsize;
    plc_v = vcoord > 0 & vcoord <= 1000;
    plc_u = ucoord > 0 & ucoord <= 2000;
    Img(vcoord(plc_v), ucoord(plc_u)) = Img(vcoord(plc_v), ucoord(plc_u)) +
Inorm(plc_v,plc_u);
    % if a star falls on the edge, will only plot the portion that's in FOV
end

figure
imagesc(Img);
colormap(gray);

end
```

Appendix D: Algorithm Testing Matlab Files

D.1: findCentroid.m

```
% Description:  Finds the centroid of a star given its upperleft coordinate
% Inputs:       k,m coordinates of upperleft pixel on star field, Region of
%               Interest (ROI)
% Outputs:      Centroid in X-dir, Centroid in Y-dir

function [CenX,CenY] = findCentroid(k,m,ROImg)

sx = length(ROImg(:,1));
sy = length(ROImg(1,:));
Cy = zeros(sx,sy);
Cx = zeros(sx,sy);

restartm = m;
for kk = 1:sx
    for mm = 1:sy
        Cy(kk,mm) = ROImg(kk,mm)*k;
        Cx(kk,mm) = ROImg(kk,mm)*m;
        m = m + 1;
    end
    k = k + 1;
    m = restartm;
end

CenY = sum(Cy(:))/sum(ROImg(:));
CenX = sum(Cx(:))/sum(ROImg(:));

end
```

D.2: starIdentification.m

```
% Description:  Identifies the regions of interest (ROI) given a star field
%               and calculates the centroids of each potential star window.
%               Will place red crosses over centroids of stars
% Inputs:       Img Matrix of star field
% Ouputs:       list of Centroids, ROI structure which includes: star's
%               centroid, star's max count value, star's ROI window
% Fns used:     findCentroid

function [ROI,Centroids] = starIdentification(Img)

specs = camera_specs();
Nsize = specs.Nsize;
maxcount_dimstar = 3;

IMG = Img;
ROI = [];
star = 1;
```

```
startnewj = 25;

for ii = 25:length(IMG(:,1))-25;
    for jj = startnewj:length(IMG(1,:))-25;
    i = ii;
    j = jj;

    if IMG(i,j) >= maxcount_dimstar
        countrj = 1;
        countlj = 1;
        countui = 1;
        countdi = 1;
        count2j = 1;
        while IMG(i,j+1) >= IMG(i,j)
            j = j + 1;
            count2j = count2j + 1;
        end
        while IMG(i+1,j) >= IMG(i,j)
            i = i + 1;
        end
        newi = i; newj = j; % right
        while IMG(newi,newj+1) ~= 0
            newj = newj + 1;
            countrj = countrj + 1;
            if newj == 1
                break
            end
        end
        newi = i; newj = j; % left
        while IMG(newi,newj-1) ~= 0
            newj = newj - 1;
            countlj = countlj + 1;
            if newj == 1
                break
            end
        end
        newi = i; newj = j; % up
        while IMG(newi-1,newj) ~= 0
            newi = newi - 1;
            countui = countui + 1;
            if newi == 1
                break
            end
        end
        newi = i; newj = j; % down
        while IMG(newi+1,newj) ~= 0
            newi = newi + 1;
            countdi = countdi + 1;
            if newi == 1
                break
            end
        end

        StarR = ['star',num2str(star)];
        ROI.(StarR).C = IMG(i-countui:i+countdi,j-countlj:j+countrj);
        ROI.(StarR).ulvl(1,1) = i - countui;
```

```
        ROI.(StarR).ulvl(1,2) = j - countlj;
        ROI.maxcounts(star) = max(ROI.(StarR).C(:));
        IMG(i-countui:i+countdi,j-countlj:j+countrj) = 0;
        star = star + 1;
        startnewj = count2j + countrj;
    end

    end
    startnewj = 25;
end


Centroids = zeros(star-1,2);
hold on
for starnum = 1:star-1
    StarR = ['star',num2str(starnum)];
    ROImg = ROI.(StarR).C;
    k = ROI.(StarR).ulvl(1,1);
    m = ROI.(StarR).ulvl(1,2);
    [CenX,CenY] = findCentroid(k,m,ROImg);
    Centroids(starnum,1) = CenX;
    Centroids(starnum,2) = CenY;
    ROI.(StarR).centroid(1,1) = CenX;
    ROI.(StarR).centroid(1,2) = CenY;
    plot(CenX,CenY,'r+')
end
hold off

end
```

D.3: createAnglesDatabase.m

```
% Description:  Creates a database of angular distances between each star
%               vector in ECI coordinates and every other star in the catalog.
% Inputs:       none
% Outputs:      *Saves* all angular distances in a mat file

load ('mag6_ECI.mat')
ECIunitvectors = mag6.ECIcoord;
angulardist = [];

for i = 1:length(ECIunitvectors)
    for j = 1:length(ECIunitvectors)

        v1 = ECIunitvectors(i,:);
        v2 = ECIunitvectors(j,:);
        v = ['v',num2str(i)];
        angulardist.(v)(j,1) = acos(dot(v1,v2)/(norm(v1)*norm(v2)))*180/pi;
        angulardist.(v) (j,2)

    end
end
```

D.4: createStarCatalogDatabase.m

```
% Description:  Creates a database of local star parameters including: HDID
%               numbers of the three stars, smallest angular distance to
%               the 2 closest stars and the spherical angle between the two
% Inputs:       loads the angulardist and mag6_ECI mat files
% Outputs:      *Saves* the Trianular Features in a StarCatalogDatabase.MAT

load('mag6_ECI.mat')
load('angulardist.mat')
StarCatalogDatabase = zeros(4559,6);

for i = 1:4559 % var instead?

    v = ['v',num2str(i)];
    [angles_sorted,b] = sort(angulardist.(v));
    StarCatalogDatabase(i,1) = i;
    StarCatalogDatabase(i,2) = b(2);
    StarCatalogDatabase(i,3) = b(3);
    StarCatalogDatabase(i,4) = angles_sorted(2);
    StarCatalogDatabase(i,5) = angles_sorted(3);
    V1 = mag6.ECIcoord(i,:);
    V2 = mag6.ECIcoord(b(2),:);
    V3 = mag6.ECIcoord(b(3),:);
    a = V1-V2;
    B = V1-V3;
    StarCatalogDatabase(i,6) = acos(dot(a,B)/(norm(a)*norm(B)))*180/pi;

end
```

D.5: findAngularDistance.m

```
% Description:  Finds the angular distance between two vectors
% Inputs:       two vectors
% Outputs:      angular distance between the two vectors

function angdist = findAngularDistance(v1,v2)

angdist = acos(dot(v1,v2)/(norm(v1)*norm(v2))); % [rad]

end
```

D.6: Pixel2CameraSpace.m

```
% Description:  Converts a vector in pixel space to the camera's reference
frame
% Inputs:       Camera specs, Pixel Space vector
% Outputs:      Corresponding vector in camera space

function CameraSpaceVector = Pixel2CameraSpace(specs, PSVec)

uc = (specs.U/2)+1;
vc = (specs.V/2)+1;

CameraSpaceVector = zeros(3,1);
Vx_Vz = (uc - PSVec(1))*specs.pp/specs.f;
Vy_Vz = (vc - PSVec(2))*specs.pp/specs.f;
Vz = sqrt(1- Vx_Vz^2 - Vy_Vz^2);
Vx = Vx_Vz*Vz;
Vy = Vy_Vz*Vz;
CameraSpaceVector(1,1) = Vx;
CameraSpaceVector(2,1) = Vy;
CameraSpaceVector(3,1) = Vz;

end
```

D.7: findAngles.m

```
% Description:  Calculates the spherical angle and angular distances
%               between stars and outputs a triad of star vectors in the
%               camera frame and in pixel space.
% Inputs:       list of star Centroids, pivot star number
% Outputs:      TF = triangular feature includes angular distances from the
pivot
%               star to 2 closest stars and the spherical angle between them

function [starVectors,starVectors_PixelS,TF] = findAngles(Centroids,pivot)

specs = camera_specs();
Diff = zeros(length(Centroids)+2,length(Centroids));
for i = 1:length(Centroids)
    star1 = [Centroids(i,:) -specs.f/specs.pp]';
    star1cam = Pixel2CameraSpace(specs,star1);
    for j = 1:length(Centroids)
        starx = [Centroids(j,:) -specs.f/specs.pp]';
        starxcam = Pixel2CameraSpace(starx);
        Diff(j,i) = findAngularDistance(star1cam,starxcam);
    end
    [~,b] = sort(Diff(1:length(Centroids),i));
    Diff(length(Centroids)+1,i) = b(2); %Centroid(b(1),:) is 1st closest star
    Diff(length(Centroids)+2,i) = b(3); %Centroid(b(2),:) is 2nd closest star
end

pivotstar = Centroids(pivot,:);
firstNeighbor = Centroids(Diff(length(Centroids)+1,pivot),:);
secondNeighbor = Centroids(Diff(length(Centroids)+2,pivot),:);
```

```
starVectors_PixelS = zeros(3,3);
starVectors_PixelS(:,1) = [firstNeighbor -specs.f/specs.pp]';
starVectors_PixelS(:,2) = [pivotstar -specs.f/specs.pp]';
starVectors_PixelS(:,3) = [secondNeighbor -specs.f/specs.pp]';

starVectors = zeros(3,3);
for i = 1:3
starVectors(:,i) = Pixel2CameraSpace(starVectors_PixelS(:,i));
end

TF = zeros(1,3);
V1 = starVectors(:,1);
V2 = starVectors(:,2);  % pivot
V3 = starVectors(:,3);
v1 = starVectors_PixelS(:,2) - starVectors_PixelS(:,1);
v2 = starVectors_PixelS(:,2) - starVectors_PixelS(:,3);
A = findAngularDistance(V2,V1);
B = findAngularDistance(V2,V3);
C = findAngularDistance(v1,v2);
TF(1,1) = A*180/pi;
TF(1,2) = B*180/pi;
TF(1,3) = C*180/pi;

end
```

D.8: findStarMatch.m

```
% Description:  Find star match between star catalog and those in the FOV
% Inputs:       Structure called "Stars" which includes HDID numbers of
%               stars used in generating images, list of Centroids
%               identified in the FOV
% Outputs:      Vectors of stars in body and inertial reference frames

function [starVectors,inertialVectors] = findStarMatch(Stars,Centroids)

load('StarCatalogDatabase.mat')
load('mag6_ECI.mat')
pivot = 1;
iterate = 1;
limit = length(Centroids);

while iterate == 1 && pivot <= limit
[starVectors,starVectors_PixelS,TF] = findAngles(Centroids,pivot);

for i = 1:length(StarCatalogDatabase)

    angdist1 = StarCatalogDatabase(i,4);
    angdist2 = StarCatalogDatabase(i,5);
    sphangle = StarCatalogDatabase(i,6);

    if angdist1 < TF(1)+0.01 && angdist1 > TF(1)-0.01 && angdist2 ...
```

```
            < TF(2)+0.01 && angdist2 > TF(2)-0.01 && sphangle < TF(3)+0.5...
            && sphangle > TF(3)-0.5
        match = StarCatalogDatabase(i,:);
        hold on
        plot(starVectors_PixelS(1,:),starVectors_PixelS(2,:),'r:')
        hold off
        iterate = 0;
        break
    end
end
pivot = pivot + 1;
if pivot > limit
    disp('There was no match found. There may not be enough stars in the FOV')
%     testingMonteCarlo  = 1;
end
end
if pivot <= limit
HDID = [mag6.HDID(match(2)) mag6.HDID(match(1)) mag6.HDID(match(3))];
LIA = ismember(HDID, Stars.HDID);

if sum(LIA) == 3
%     testingMonteCarlo = 0;
    disp(['You have found 3 stars with corresponding HDID numbers:
',num2str(HDID)])
end
inertialVectors = [mag6.ECIcoord(match(2),:)' mag6.ECIcoord(match(1),:)'
mag6.ECIcoord(match(3),:)'];

end

end
```

D.9: AttitudeDeterminationQMethod.m

```
% Description:  Obtains the quaternion given two sets of vectors in body
%               and reference frame using the Q Method
% Inputs:       Vectors in body frame, Vectors in reference frame
% Ouputs:       Quaternion

function quat = AttitudeDeterminationQMethod(starVectors,inertialVectors)

vb = starVectors;
vr = inertialVectors;
ai = 1; % weight

Bi = zeros(3,3);
B = zeros(3,3);
for i = 1:3
    Bi = ai*vb(:,i)*transpose(vr(:,i));
    B = B + Bi;
end

z = [B(2,3)-B(3,2); B(3,1)-B(1,3); B(1,2)-B(2,1)];
```

```
K = [B+transpose(B) - trace(B)*eye(3) z; transpose(z) trace(B)];
[eigV,lambda] = eig(K);
quat = [eigV(4,4); eigV(1:3,4)];
theta = (2*acos(eigV(4,4)))*180/pi;

end
```

D.10: getQuaternionError.m

```
% Description:  Obtain the error in estimated and true quaternion. fourth
%               parameter is the scalar
% Inputs:       Estimated quaternion, True quaternion
% Outputs:      Attitude error [deg]


function attitudeError = getQuaternionError(q_true,q_est)


q_true_inv = [q_true(1) q_true(4) -q_true(3) q_true(2);...
    -q_true(4) q_true(1) q_true(2) q_true(3);...
    q_true(3) -q_true(2) q_true(1) q_true(4);...
    -q_true(2) -q_true(3) -q_true(4) q_true(1)];
q_est = [-q_est(2:4); q_est(1)];

q_err = q_true_inv*q_est;

attitudeError = 2*acosd(q_err(4));


end
```

D.11: getLossFunctionValue.m

```
% Description:  Finds the loss function value solution to Wahba's problem
% Inputs:       Vectors of stars in the body and inertial reference frames
% Outputs:      Loss function value, J


function J = getLossFunctionValue(starVectors,inertialVectors)

vb = starVectors;
vr = inertialVectors;
ai = 1; % weight
```

```
Bi = zeros(3,3);
B = zeros(3,3);
for i = 1:3
    Bi = ai*vb(:,i)*transpose(vr(:,i));
    B = B + Bi;
end

z = [B(2,3)-B(3,2); B(3,1)-B(1,3); B(1,2)-B(2,1)];
K = [B+transpose(B) - trace(B)*eye(3) z; transpose(z) trace(B)];
[eigV,lambda] = eig(K); %largest lambda is lambda(4,4)
lambdamax = lambda(4,4);
lambda0 = ai + ai + ai; %sum of the weights

J = lambda0- lambdamax;%(transpose(q)*K*q);


end
```

D.12: runMonteCarloQMethod.m

```
% Description:  Run code to generate a star image and output the
%               corresponding quaternion. Loads the StarCatalogDatabase and
%               mag6_ECI MAT files, generates a star field using a random
%               quaternion and runs the star identification algorithm to
%               find a match between the stars in the image and those in the
%               catalog. The code will then run attitude determination via
%               the Q Method and outputting the same quaternion used to
%               generate the image. *Saves* as a mat file
% Inputs:       None
% Outputs:      MonteCarloQMethod Structure


for i = 930:1000

    q_ImgGen = generate_random_quaternion();
    [Img,Stars] = GenerateStarField(q_ImgGen);
    [ROI,Centroids] = starIdentification(Img);
    [starVectors,inertialVectors,testingMonteCarlo] =
findStarMatch(Stars,Centroids);

    if testingMonteCarlo == 1

        Run = ['run',num2str(i)];
        MonteCarloQMethod.q.(Run) = [0 0];
        MonteCarloQMethod.Centroids.(Run) = 0;
        MonteCarloQMethod.Vectors.(Run) = 0;
        MonteCarloQMethod.attitudeError(i) = 0;
    else
        q_AttDet = AttitudeDeterminationQMethod(starVectors,inertialVectors);
        attitudeError = getQuaternionError(q_ImgGen,q_AttDet)  ;
```

```
        Run = ['run',num2str(i)];
        MonteCarloQMethod.q.(Run) = [q_ImgGen q_AttDet];
        MonteCarloQMethod.Centroids.(Run) = Centroids;
        MonteCarloQMethod.Vectors.(Run) = [starVectors inertialVectors];
        MonteCarloQMethod.attitudeError(i) = attitudeError;
    end

end
```

D.13: calcMonteCarloAttitudeErrror.m

```
% Description:  Calculate the attitude error from the Monte Carlo results
% Inputs:       None, loads mat file created from runMonteCarloQMethod.m
% Outputs:      Average error in attitude errors, both 0 and 360 degrees

load('MonteCarloQMethod.mat')
AttitudeError = MonteCarloQMethod.attitudeError;
zeroDeg = zeros(1000,1);
threesixtDeg = zeros(1000,1);
j = 1; k = 1;

for i = 1:length(AttitudeError)

   run = AttitudeError(i);
   if run < 1
       zeroDeg(j,1) = run;
       j = j + 1;
   end
   if run > 1
       threesixtDeg(k,1) = run;
       k = k + 1;
   end

end

deletezeros = zeroDeg > 0;
zeroDeg = zeroDeg(deletezeros);
deletezeros = threesixtDeg > 0;
threesixtDeg = threesixtDeg(deletezeros); %0 = 360 degrees in rotation

avgZeroDeg = mean(zeroDeg(:));
stdZeroDeg = std(zeroDeg(:));
avgThreesixtDeg = mean(threesixtDeg(:));
stdThreesixtDeg = std(threesixtDeg(:));
```

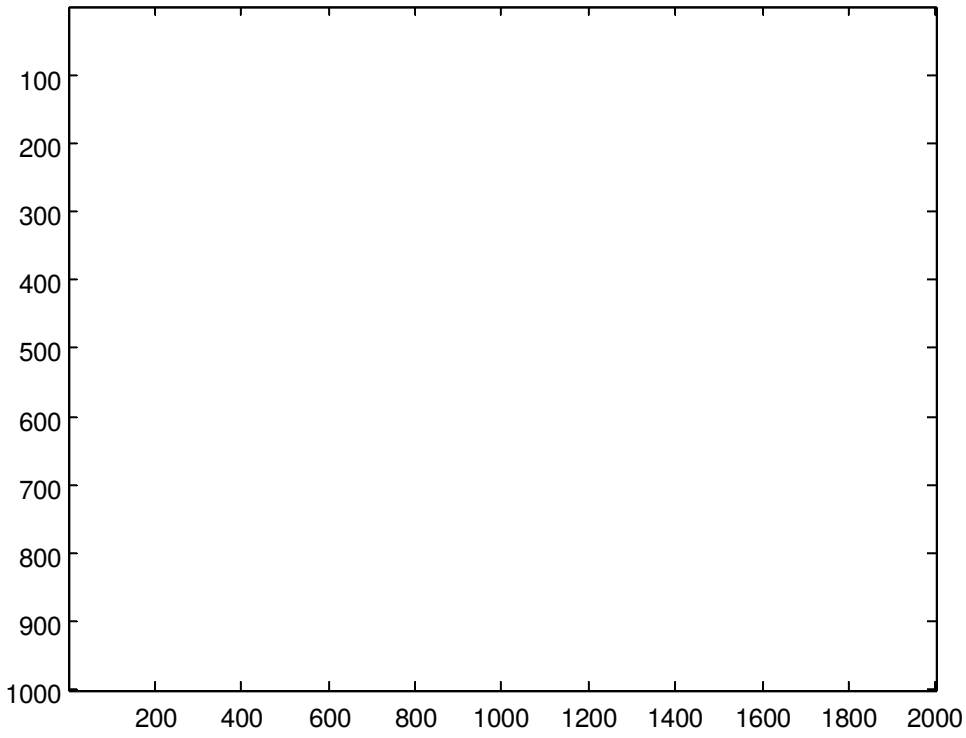
D.14: runScript_StarTrackerforAttitudeDetermination.m

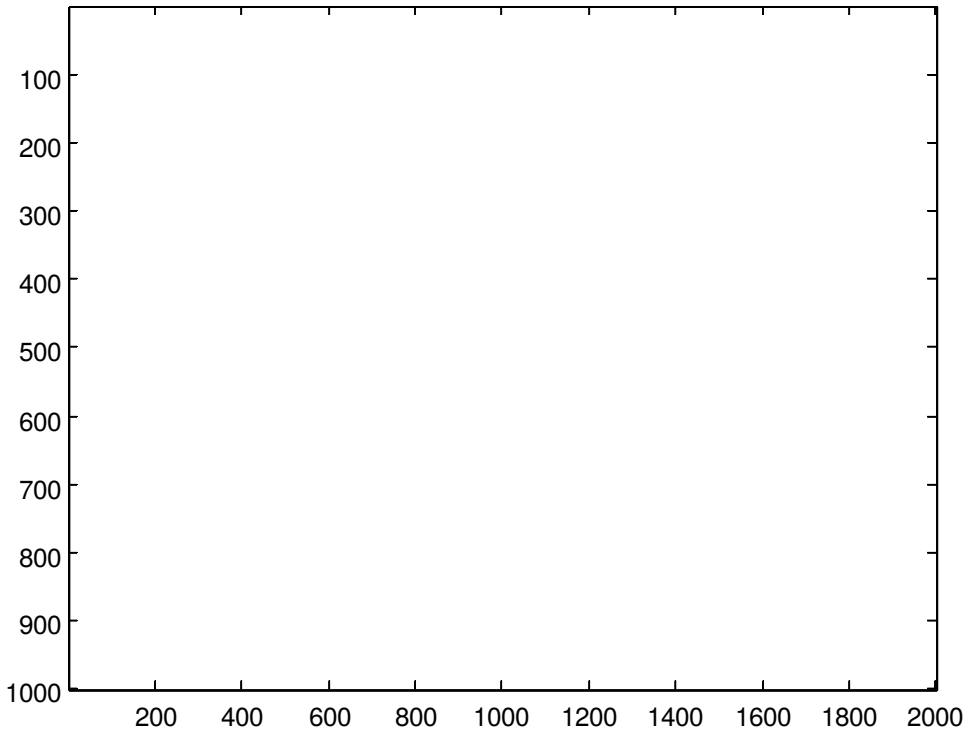
```
% runScript_StarTrackerforAttitudeDetermination.m
```

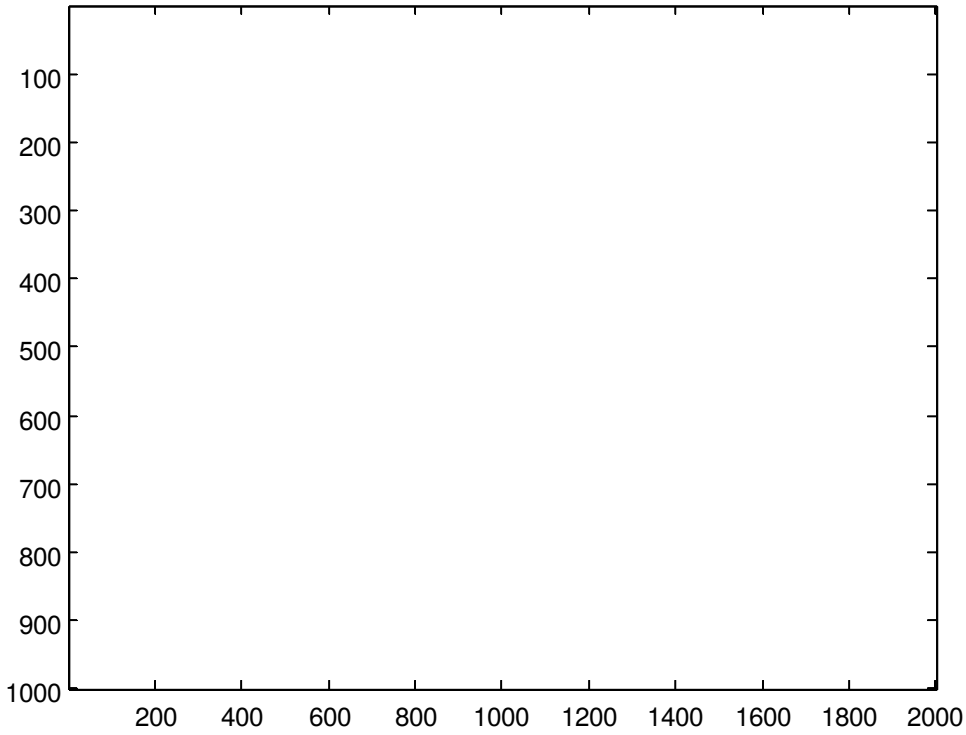
```
% Created: 4/15/15
% Updated: 5/13/15
% Description:  Run code to generate a star image and output the
%               corresponding quaternion. Loads the StarCatalogDatabase and
%               mag6_ECI MAT files, generates a star field using a random
%               quaternion and runs the star identification algorithm to
%               find a match between the stars in the image and those in the
%               catalog. The code will then run attitude determination via
%               the Q Method and outputting the same quaternion used to
%               generate the image.

close all; clear all; clc;
q_ImgGen = generate_random_quaternion()
[Img,Stars] = GenerateStarField(q_ImgGen);
[ROI,Centroids] = starIdentification(Img);
[starVectors,inertialVectors] = findStarMatch(Stars,Centroids);
q_AttDet = AttitudeDeterminationQMethod(starVectors,inertialVectors)
attitudeError = getQuaternionError(q_ImgGen,q_AttDet)
J = getLossFunctionValue(starVectors,inertialVectors)
```

Appendix E: MATLAB Script Figure Outputs

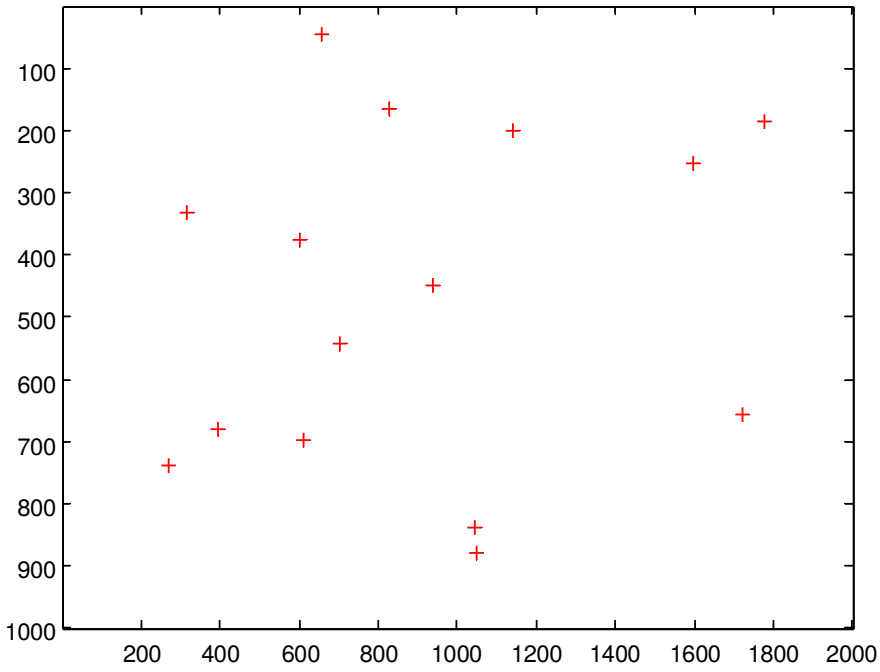
E.1 Example Raw Image Generation Outputs

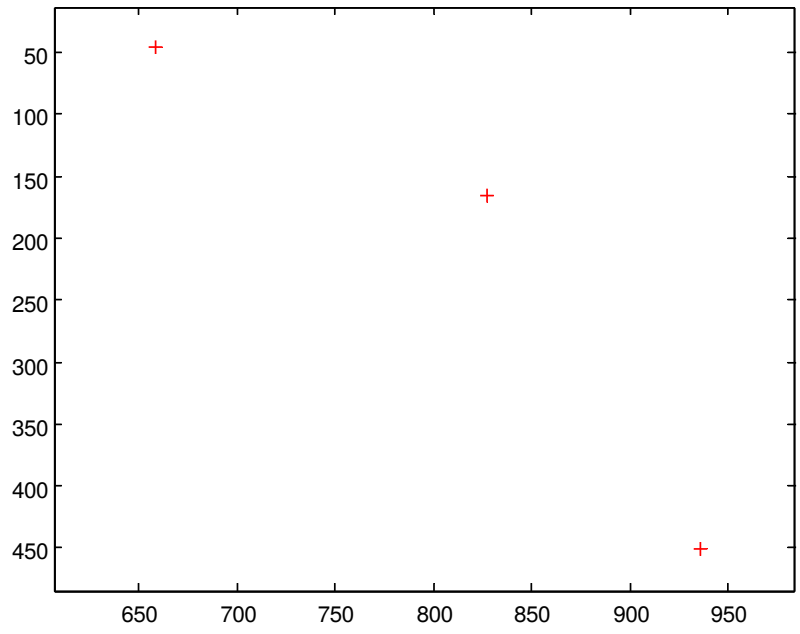


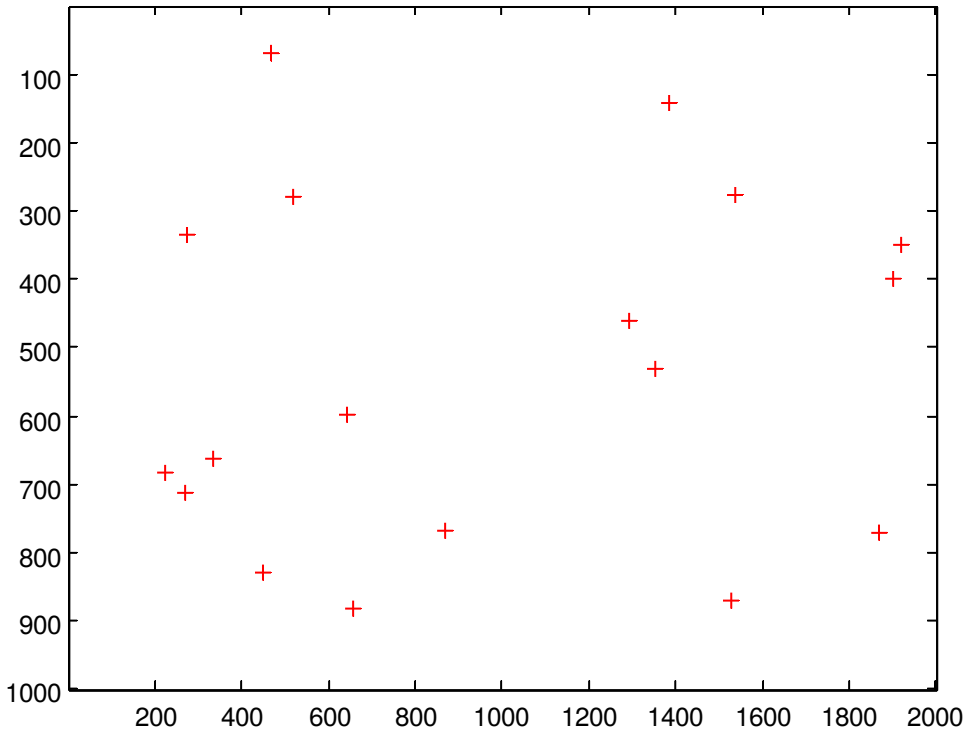

100
200
300
400
500
600
700
800
900
1000
200
400
600
800
1000
1200
1400
1600
1800
2000

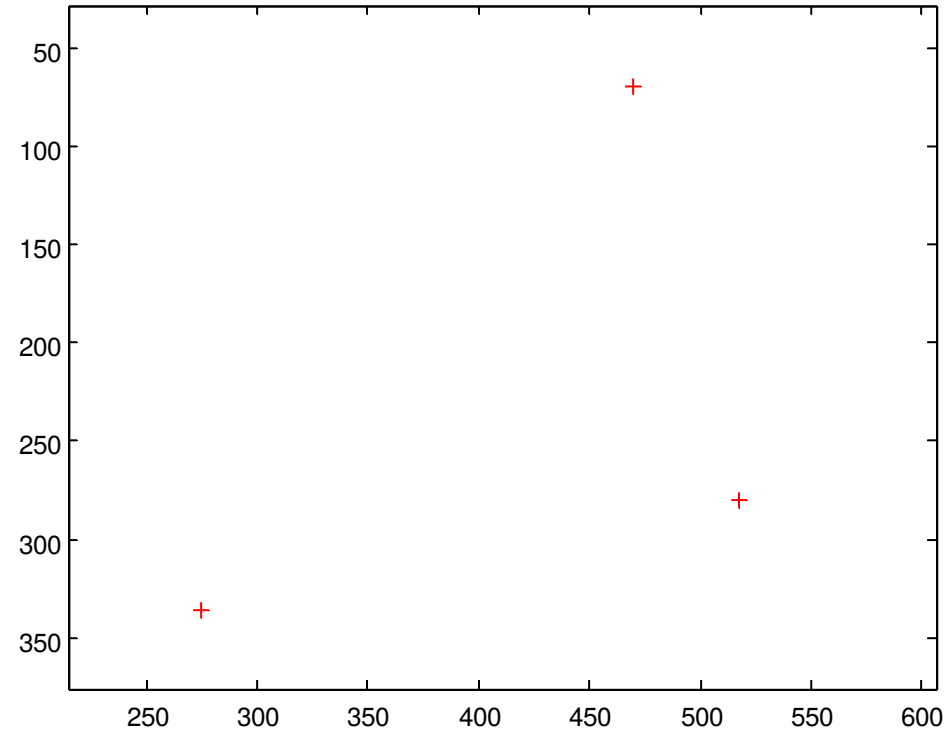

100
200
300
400
500
600
700
800
900
1000
200
400
600
800
1000
1200
1400
1600
1800
2000

E.2 Example Star Identification / Star Matching Outputs






100
200
300
400
500
600
700
800
900
1000
200
400
600
800
1000
1200
1400
1600
1800
2000


50
100
150
200
250
300
350
250
300
350
400
450
500
550
600