Fast Star Tracker Centroid Algorithm for High Performance CubeSat with Air Bearing Validation

Matthew Knutson, David Miller

June 2012 SSL # 5-12

Fast Star Tracker Centroid Algorithm for High Performance CubeSat with Air Bearing Validation

Matthew Knutson, David Miller

June 2012 SSL # 5-12

This work is based on the unaltered text of the thesis by Matthew Knutson submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of Master of Science in Aeronautics and Astronautics at the Massachusetts Institute of Technology.

Fast Star Tracker Centroid Algorithm for High Performance CubeSat with Air Bearing Validation

by

Matthew W. Knutson

Submitted to the Department of Aeronautics and Astronautics
on May 24, 2012 in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Aeronautics and Astronautics

ABSTRACT

State of the art CubeSats such as ExoplanetSat require pointing precision for the science payload on the order of arcseconds. ExoplanetSat uses dual stage control to achieve the pointing requirement. Reaction wheels provide coarse satellite attitude control while a high bandwidth piezoelectric stage performs fine optical stabilization. The optical sensor provides star images from which a centroiding algorithm estimates the star locations on the optical focal plane. The star locations are used for both the optical control loop and satellite attitude determination. The centroiding algorithm requires a short processing time to maximize the bandwidth of the fine control loop.

This thesis proposes a new fast centroiding algorithm based on centroid window tracking. The tracking algorithm utilizes centroid data from previous image frames to estimate the motion of the optical sensor. The estimated motion provides a prediction of the current centroid locations. An image window is centered at each predicted star location. A center of mass calculation is performed on the image window to determine the centroid location. This proposed algorithm is shown to reduce the computation time by a factor of 10 with a novel air bearing hardware testbed.

This thesis also develops a high fidelity optical imager model in MATLAB Simulink. This model can be used to test centroiding algorithms and to simulate optical systems in a spacecraft pointing simulator. The model is validated with the air bearing testbed. Furthermore, the model is autocoded to C-code which is compatible with a rapid Monte Carlo analysis framework.

Thesis Supervisor: Sungyung Lim
Title: Senior Member of the Technical Staff, Charles Stark Draper Laboratory

Thesis Supervisor: David W. Miller
Title: Professor of Aeronautics and Astronautics

DISCLAIMER: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

Acknowledgements

This work was performed primarily under contract with Draper Laboratory as part of the Draper Laboratory Fellowship Program. This work was also performed for contract FA70001020040 with the Air Force Space and Missile Systems Center as part of the Space Engineering Academy. The author gratefully thanks the sponsors for their generous support that enabled this research.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ADCS	Attitude Determination and Control System
ADU	Analog-to-Digital Unit
CCD	Charge-Coupled Device
CMOS	Complementary Metal-Oxide-Semiconductor
COM	Center of Mass
FWHM	Full Width at Half Maximum
ID	Identification
kTC	Thermal
LIS	Lost-in-Space
MBD	Model-Based Design
MEMS	Micro-Electro-Mechanical Systems
PIL	Processor-in-the-Loop
PRNU	Photo Response Non-Uniformity
PSF	Point Spread Function
ROI	Region of Interest
SWIL	Software-in-the-Loop

Chapter 1

1. Introduction

1.1. Background

The CubeSat was originally conceived as a low cost alternative to provide universities with the ability to access space. When CubeSats were first conceived, their primary purpose was to provide hands on education to undergraduate and graduate students on the development of space hardware. Recently, they have evolved to play a more dominant role in scientific research. The scientific objectives often levy the requirement for high precision pointing of the science payload.

There are several examples which highlight the need for high precision pointing space systems [1, 5, 8]. One example is ExoplanetSat which is a 3U (10cm x 10cm x 30cm) CubeSat under development at Draper Laboratory and the Massachusetts Institute of Technology with a mission to search for Earth like exoplanets using the transit method [1]. This mission requires arcsecond level pointing for the optical payload. ExoplanetSat uses a dual stage control algorithm in which reaction wheels provide coarse attitude control at 4 Hz and a piezoelectric stage provides fine optical control at 12 Hz. The piezoelectric stage translates the focal plane with the charge-coupled device (CCD) and complementary metal-oxide-semiconductor (CMOS) detectors so that the images on the detectors are stabilized on the order of arcseconds. The translation command is created from the star centroid locations on the optical detectors. Figure 1-1 below shows the satellite prototype, Figure 1-2 shows the payload optical system, and Figure 1-3 shows the focal plane array layout.

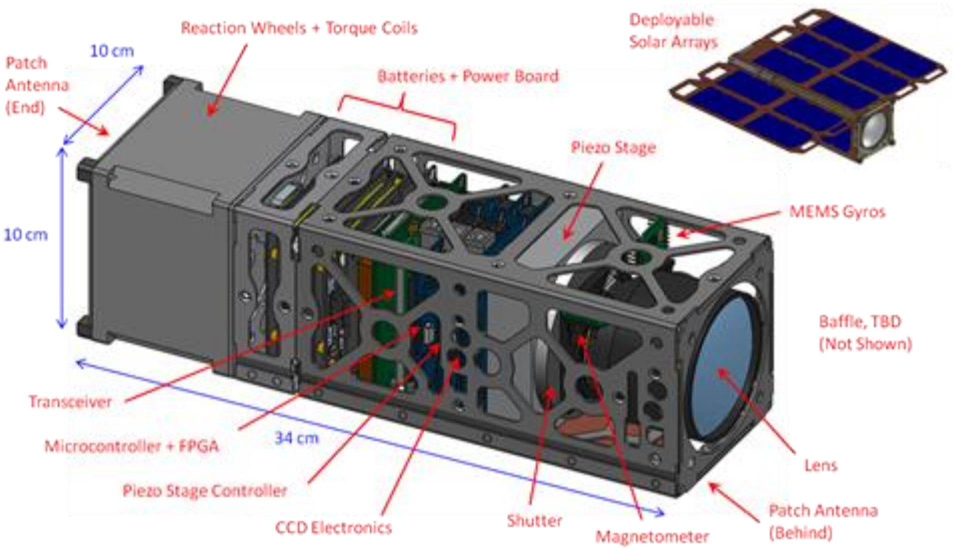


Figure 1-1: ExoplanetSat Prototype [3]

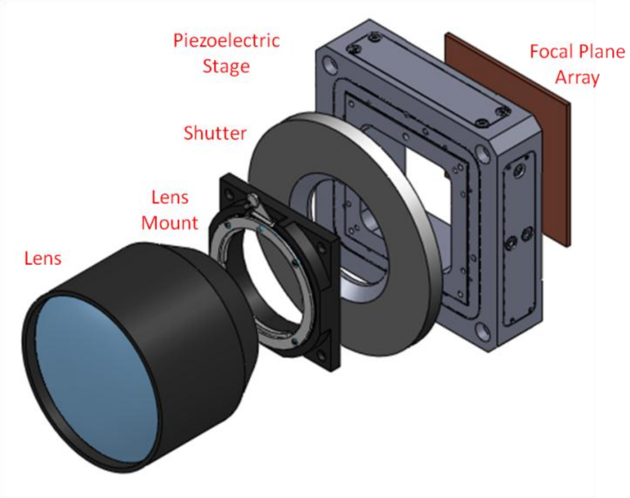


Figure 1-2: ExoplanetSat Payload Optical System [3]

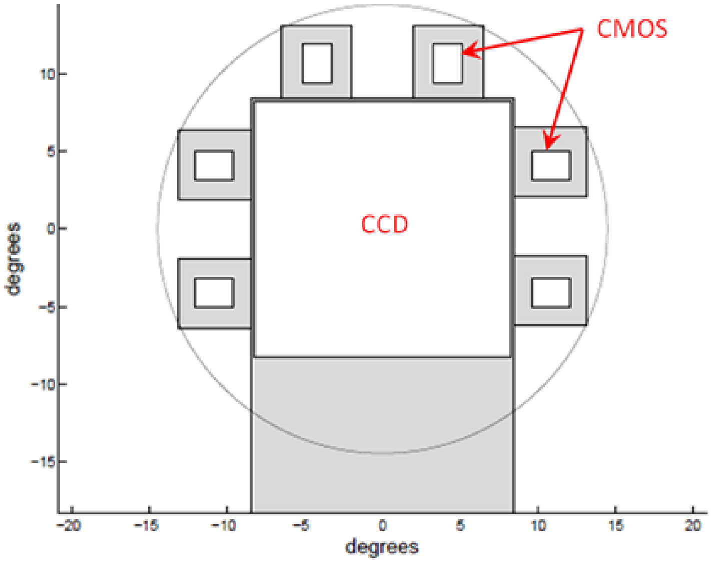


Figure 1-3: ExoplanetSat Focal Plane Array Layout [3]

The centroid locations must be calculated from the detector output at high rate to close the piezo stage control loop and to provide better spacecraft attitude and angular rate estimates using star tracker algorithms [3].

The high precision pointing system can also be used in other satellite applications such as high bandwidth communication [5, 6, 7] and formation flying [8]. High bandwidth communication uses very narrow beam widths requiring fine pointing systems on the order of up to a few arcseconds [5, 6, 7]. An example of formation flying is robotic assembly. Small autonomous satellites cooperatively assemble a large space system such as a modular optical telescope [8]. This assembly requires a precise pointing system to fit each component into place.

1.2. Problem Statement

The centroid algorithm takes significant computation time to produce centroid output from the detected star image. The computation time delay affects the pointing accuracy of the dual stage control. In particular, this time delay limits the bandwidth of the fine optical control loop, thereby reducing pointing precision. Therefore, it is imperative to minimize the processing time of the centroid algorithm in the dual stage control.

Another benefit of a fast centroiding algorithm is more accurate attitude determination with optical star measurements. These measurements are processed by star tracker algorithms to provide attitude estimates. For high precision pointing satellites, attitude estimates are often augmented by gyroscope rate measurements. However, micro-electro-mechanical systems (MEMS) gyroscopes for CubeSats are currently too noisy. A star tracker with a fast centroiding algorithm can provide accurate attitude rate estimates, therefore eliminating the need for a gyroscope.

The processing time of the centroiding algorithm is affected by the processor speed and the algorithm efficiency. A dedicated high performance flight processor can reduce the centroiding algorithm processing time. However, CubeSats may not be able to support the additional cost, size, mass, volume, and power of a dedicated processor. Therefore, the preferred solution for CubeSats is to improve the centroiding algorithm efficiency.

1.3. Literature Review

Ref [9] provides an excellent overview of how star trackers operate. The star tracker consists of three components: image data pre-processing, centroiding, and attitude determination. The centroiding algorithm processes the image data and produces the center locations of each star in the image. The reference proposes a two-phase operation including lost-in-space (LIS) and tracking to reduce computation time. The LIS mode processes the full frame image data to determine the centroid locations. The tracking mode uses the previous centroid data to predict the location of the stars in the current image. Small centroid windows are defined at the predicted star locations. A center of mass algorithm is applied over these windows to measure the centroid locations. This method may speed up the centroiding algorithm and thus star tracker processing. However, the reference does not provide a tracking algorithm.

Ref [15] presents a tracking algorithm based on an iterative least square estimation of star tracker attitude with optical measurements. The attitude is then used to define centroid windows. The reference assumes that the rotation between frames is small and the star tracker is modeled as a pin-hole camera. However, there is no end-to-end hardware implementation.

Refs [13, 14] propose estimation of attitude and attitude rate with optical star measurements via an extended Kalman filter. This approach improves the attitude rate estimate. However, it is not applicable to the dual stage control which requires fast algorithms to achieve high precision optical pointing.

There are many references which focus on accuracy of the centroid algorithms and modeling of the optical detectors. Refs [10, 11, 12] characterize the systematic errors of centroid algorithms and analyze the effect of the centroid point spread function (PSF). Refs [17, 18, 19, 20] describe modeling of the CCD and CMOS detectors.

1.4. Thesis Objectives and Approach

The primary objective of this thesis is to develop a fast centroiding algorithm and demonstrate the performance on end-to-end hardware. The fast centroiding algorithm uses the two-stage approach [9] with the tracking algorithm developed by [15]. The tracking algorithm uses centroid data from previous images to estimate the slew rate of the optical system. The rate estimate propagates the centroid locations forward to predict their locations in the current image. Small image windows are centered at the predicted locations. A center of mass (COM) centroid algorithm is calculated over the windows to determine the estimated centroid locations. Computation time is reduced because the centroid algorithm is not executed over the entire image.

The tracking algorithm is further optimized to reduce the computation time. First, the suggested iteration is removed because the satellite attitude rate is small relative to the high image frame rate of the optical system. Second, the least square solution is found by a computationally efficient analytic formula of the pseudo-inverse. The tracking algorithm is also enhanced with a simple and robust star identification algorithm to track the same star between image frames. This algorithm is not based on the standard star tracking algorithm which matches stars from the image to a star catalog but a feature matching algorithm which directly compares stars from successive image frames.

More importantly, the fast centroiding algorithm developed in this thesis is integrated with a hardware testbed assembled with camera, imager, artificial star field, and embedded processor. The algorithm is tested to analyze the real time performance in a dynamic slew environment similar to what the satellite will experience in space. This enables inexpensive validation of flight performance necessary to ensure mission success.

The second objective of this thesis is to develop a high-fidelity star tracker image model in MATLAB Simulink. High-fidelity simulations offer a low cost platform for conducting hardware trade studies and predicting system performance before building engineering models and testbeds. The star tracker model simulates various noise sources which affect the optical system [19, 20]. This model is validated with the hardware testbed data. The star tracker model is then integrated with the developed centroiding algorithm to analyze the performance in a software-in-the-loop (SWIL) testbed. Future work can use the SWIL testbed to validate further algorithm development before hardware implementation.

The star tracker model is written in MATLAB Simulink for inexpensive development and maintenance. However, it is computationally expensive to simulate. Therefore, it is autocoded to C-code using a model-based design (MBD) framework developed at Draper Laboratory. This framework enables rapid development and test because algorithm updates are made in Simulink then quickly implemented and tested in C.

1.5. Thesis Outline

This thesis is organized as follows. Chapter 2 provides the fundamentals of centroiding algorithms and develops the new fast centroiding algorithm. Chapter 3 develops the high-fidelity star tracker model which simulates the optical system. The model outputs simulated star field images which incorporate several optical and detector noise sources in order to mimic the actual hardware. Chapter 4 presents the software architecture for the fast centroiding algorithm. The software architecture integrates the fast centroiding algorithm, star tracker model, and star identification algorithm. This chapter also presents results of the centroiding algorithm performance analysis. Chapter 5 describes the air bearing testbed and performance of the fast centroiding algorithm. This chapter also presents validation of the star tracker model using the testbed results. Finally, Chapter 6 presents the MBD framework which enables rapid Monte Carlo analysis in a SWIL testbed.

Chapter 2

2. Algorithm Development

2.1. Centroiding Algorithm Development

2.1.1. Centroiding Algorithm Motivation and Purpose

A centroiding algorithm measures the star locations in the optical detector. The fine optical control loop uses the star locations to determine the piezo stage commands to stabilize the science star image. The centroids are also used for satellite attitude determination. Attitude determination is based on matching stars from the camera image to the star catalogue.

Several sources of noise are introduced in the process of converting an image to an attitude measurement including detector noise, time delay, and star catalogue errors [9]. Errors in the centroiding algorithm along with detector noise will produce errors in the measured centroid location. Centroid errors can also cause improper matches to the star catalogue leading to attitude determination error. Therefore, it is important to utilize an accurate centroiding algorithm robust to detector noise.

The processing speed of the centroiding algorithm is also an important consideration. Time delays limit the optical pointing precision. Some image processing techniques may improve the accuracy of the centroiding algorithm at the expense of increased processing time. Therefore, it may be necessary to trade centroid error and centroiding algorithm processing speed.

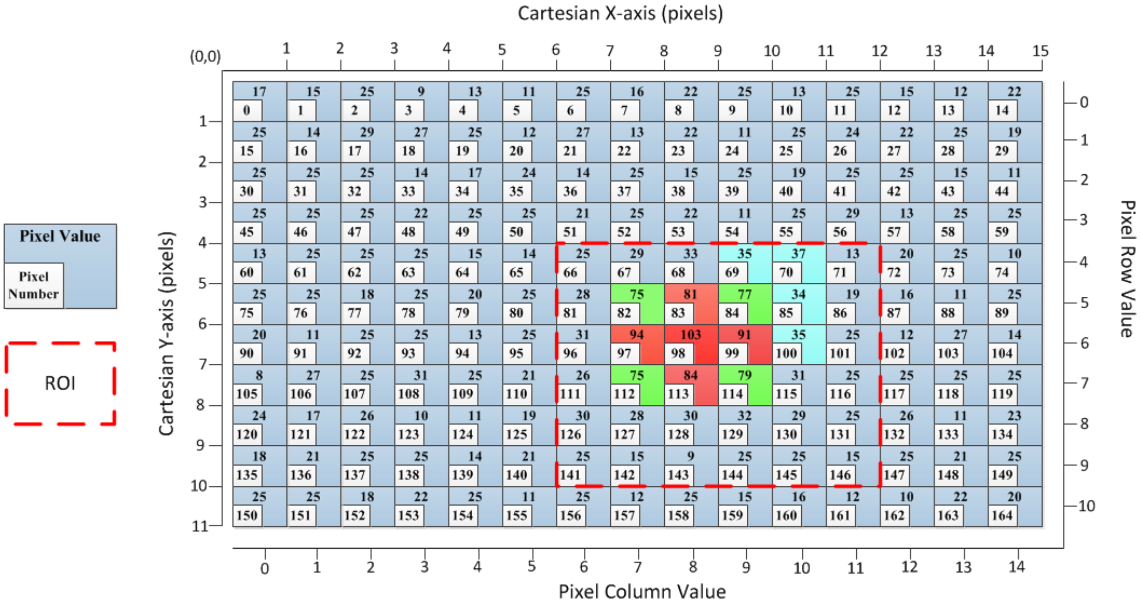
2.1.2. Centroiding Algorithm Traditional Approach

The most common approach to determining the star locations is a simple COM algorithm [9]. The reference recommends a centroiding algorithm which checks every pixel in the image and compares it to a threshold value. If a pixel is above the threshold, a square region of interest (ROI) is identified with the detected pixel at the center. The value of the ROI border pixels are averaged and subtracted from each pixel within the ROI. This process subtracts out the background noise. A COM calculation is then used to find the centroid location within the ROI.

2.1.3. Centroiding Algorithm Implementation

2.1.3.1. Centroiding Algorithm Visualization

The centroiding algorithm utilized in this thesis is similar to Ref [9]. Figure 2-1 below shows an example of the centroiding algorithm used to find the center of a star on a camera image.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	17 0	15 1	25 2	9 3	13 4	11 5	25 6	16 7	22 8	25 9	13 10	25 11	15 12	12 13	22 14
1	25 15	14 16	29 17	27 18	25 19	12 20	27 21	13 22	22 23	11 24	25 25	24 26	22 27	25 28	19 29
2	25 30	25 31	25 32	14 33	17 34	24 35	14 36	25 37	15 38	25 39	19 40	25 41	25 42	15 43	11 44
3	25 45	25 46	25 47	22 48	25 49	25 50	21 51	25 52	22 53	11 54	25 55	29 56	13 57	25 58	25 59
4	13 60	25 61	25 62	25 63	15 64	14 65	25 66	29 67	33 68	35 69	37 70	13 71	20 72	25 73	10 74
5	25 75	25 76	18 77	25 78	20 79	25 80	28 81	75 82	81 83	77 84	34 85	19 86	16 87	11 88	25 89
6	20 90	11 91	25 92	25 93	13 94	25 95	31 96	94 97	103 98	91 99	35 100	25 101	12 102	27 103	14 104
7	8 105	27 106	25 107	31 108	25 109	21 110	26 111	75 112	84 113	79 114	31 115	25 116	25 117	25 118	25 119
8	24 120	17 121	26 122	10 123	11 124	19 125	30 126	28 127	30 128	32 129	29 130	25 131	26 132	11 133	23 134
9	18 135	21 136	25 137	25 138	14 139	21 140	25 141	15 142	9 143	25 144	25 145	15 146	25 147	21 148	25 149
10	25 150	25 151	18 152	22 153	25 154	11 155	25 156	12 157	25 158	15 159	16 160	12 161	10 162	22 163	20 164

Figure 2-1: Example Camera Image with Centroid Code

The white square boxes in Figure 2-1 represent the pixel number while the number in the center of each box represents the pixel value. The dotted red line represents the ROI. Two coordinate systems are shown: the pixel row-column value and the x-y Cartesian value. It is important to note that the centroid code is written in the C programming language. The pixel-column values begin with zero because array indexing in C begins with zero.

2.1.3.2. Centroiding Algorithm Parameters

The centroiding code has three pre-defined variables including the signal threshold, noise threshold, and ROI size. The signal threshold is set to a bright pixel value indicating the presence of a star. The noise threshold is set to a value just above the noise floor. Simulation, lab, and calibration tests during flight can be used to determine threshold and ROI values. The ROI must be an even number. In the example shown in Figure 2-1 the signal threshold is 95, the noise threshold is 34, and the ROI is 6 pixels. The image size is 15 columns by 11 rows. The image pixel values are stored in an array in C which can be visualized as a matrix. Each element in the matrix is equal to the corresponding pixel value.

2.1.3.3. Centroiding Algorithm Detailed Steps

The centroiding code first checks each pixel to see if it is above the signal threshold. If the checked pixel is above the signal threshold, the ROI is centered on the pixel. The following center of mass (COM) calculation is used to find the centroid of the star. Only pixels that are above the nose threshold are included in the COM calculation. Notice that the noise threshold is subtracted from each pixel value.

$$DN = \sum_{i=Row_o}^{Row_{end}} \sum_{j=Col_o}^{Col_{end}} I_{(j,i)} - \eta \tag{2-1}$$

$$x = \frac{\sum_{j=Col_o}^{Col_{end}} \sum_{i=Row_o}^{Row_{end}} j * (I_{(j,i)} - \eta)}{DN} + 0.5 \tag{2-2}$$

$$y = \frac{\sum_{i=Row_o}^{Row_{end}} \sum_{j=Col_o}^{Col_{end}} i * (I_{(j,i)} - \eta)}{DN} + 0.5 \tag{2-3}$$

where the centroid location is defined as (x, y) with origin at the top left corner of the image; *DN* is the ROI brightness; Row_o, Row_{end}, Col_o, and Col_{end} refer to the starting pixel row value, ending pixel row value, starting pixel column value, and ending pixel column value of the region of interest; $I(j,i)$ refers to the pixel intensity value at location *(j,i)* where *j* is the column value and *i* is the row value; η is the noise threshold.

Eqs 2-2 and 2-3 add 0.5 pixels to the x and y centroid positions in order to transform the coordinate system from the pixel row and column values to the Cartesian x and y values. Because C-language treats matrices as arrays the image is stored in an array with the number of elements equal to the image width times the image height. In this example the image is stored in an array called I with 15 x 11 = 165 elements. The first pixel is at index 0 and the last pixel is at index 164 as shown in Figure 2-1. Therefore, the value of each pixel $I(j, i)$ is accessed with an array element *index* as in $I[index]$. The index is calculated from:

$$index = (i * W) + j \tag{2-4}$$

where W is the image width defined as the number of pixel columns

For example, the brightest pixel in Figure 2-1 has a value of 103 and is located at column 8 row 6. The index value is (6 x 15) + 8 = 98. In C, this pixel would be referenced as *I[98]*.

After the centroid code finds the centroid location within the ROI, the algorithm continues to check the remainder of the image for more centroids. Each time a pixel above the signal threshold is identified, the COM is calculated over the respective ROI. A separate array called *ischecked* equal to the size of the image keeps track of which pixels have already been checked. This array is initialized to all zeros. Once a pixel has been checked to see if it is above the signal threshold or included in the COM calculation, the corresponding array value is changed from 0 to 1. Therefore, as the centroiding algorithm searches for stars, a pixel must be above the signal threshold and must have a value of 0 in the *ischecked* array before the COM algorithm is calculated at its respective ROI.

2.1.4. Traditional and Current Approach Comparison

There are two key differences between the centroiding algorithm presented in this thesis and Ref [9]. First, the algorithm in the reference determines the noise threshold by calculating the average value of the pixels lining the border of the ROI. In order to save computation time by not having to calculate a new threshold value for each detected star, the centroiding algorithm presented in this thesis uses one pre-determined noise threshold value. The average value of the noise floor should remain consistent between images. Therefore, image calibration can set the noise threshold ahead of time. The second difference between the centroiding algorithms is that the reference subtracts the noise threshold from each pixel in the ROI then calculates the COM over the entire ROI. The COM algorithm in this thesis only includes pixels that are within the ROI and above the noise threshold. If a pixel is below the noise threshold, it is ignored. As shown in Eqs 2-2 and 2-3, the centroiding algorithm in this thesis saves computation time by subtracting the noise threshold and performing the COM calculation in one step.

2.2. Tracking Algorithm Development

2.2.1. Tracking Algorithm Motivation and Purpose

The centroiding algorithm processing time is crucial for achieving high precision optical pointing. Checking each pixel in the image to compare to the signal threshold takes significant computation time when dealing with detector sizes on the order of a megapixel or more. The method proposed in this thesis to speed up the centroiding algorithm is to use a centroid tracking algorithm. The centroid tracking algorithm uses the centroids from previous frames to estimate the star tracker slew. The slew estimate is used to predict the centroid locations in the current frame. The COM algorithm is calculated over a small window or ROI centered at the predicted locations in order to determine the measured centroid location. Computation time is reduced because the centroiding algorithm is not calculated over the entire image frame.

The centroid tracking algorithm provides two capabilities. First, the algorithm predicts the location of the centroids in the current image by propagating the previous centroid positions with the estimated rate. Predicting the centroid locations for each frame provides the capability to run the COM algorithm on small centroid windows therefore speeding up the centroiding algorithm. The prediction is fundamentally based on the optical detector estimated roll, pitch, and yaw angular rates. Second, the estimated angular rates are used in the fine optical control loop to stabilize the science star image. These rates can also be used in the spacecraft attitude Kalman filter, thereby eliminating the need for a gyro which is necessary for high precision pointing.

2.2.2. Computationally Intensive Approach Overview

Ref [13] proposes two methods for estimating angular rate using a star tracker. The first method called the “attitude dependent angular velocity estimation algorithm” feeds the star tracker’s attitude estimate into a Kalman filter which estimates the spacecraft angular rate. The second method called the “attitude independent approach” estimates the spacecraft angular rate based on the movement of the star centroids. At each frame, the star centroids can be used to calculate a star unit vector. See Figure 2-2 below. Assuming the star camera has a sufficiently high update rate compared to the spacecraft slew rate, the time derivative of the change in the star unit vector from frame to frame along with a least squares algorithm can be used to estimate attitude rate. First order and second order attitude independent approaches are considered. The reference concludes that the attitude independent approach is more accurate because it is not affected by attitude determination errors.

The attitude dependent approach will require the attitude determination and control system (ADCS) algorithm to match stars from each frame to the star catalogue to determine the spacecraft attitude and then use a Kalman filter or back differencing to determine rate. This approach is too slow. The attitude independent approach is also too complex given its use of least squares filters. In order to reduce the computation time, this thesis will utilize a modified version of the centroid tracking algorithm developed in Ref [15] which is essentially a simplified version of the attitude independent approach of Ref [13].

2.2.3. Simplified Tracking Algorithm Derivation

Ref [15] proposes a method for attitude estimation which can be used to estimate spacecraft rates. The equation for *dQ* on page 5 of Ref [15] contains a sign error. Therefore, the equations for the centroid tracking algorithm are derived again. The equations reference Figure 2-2 below.

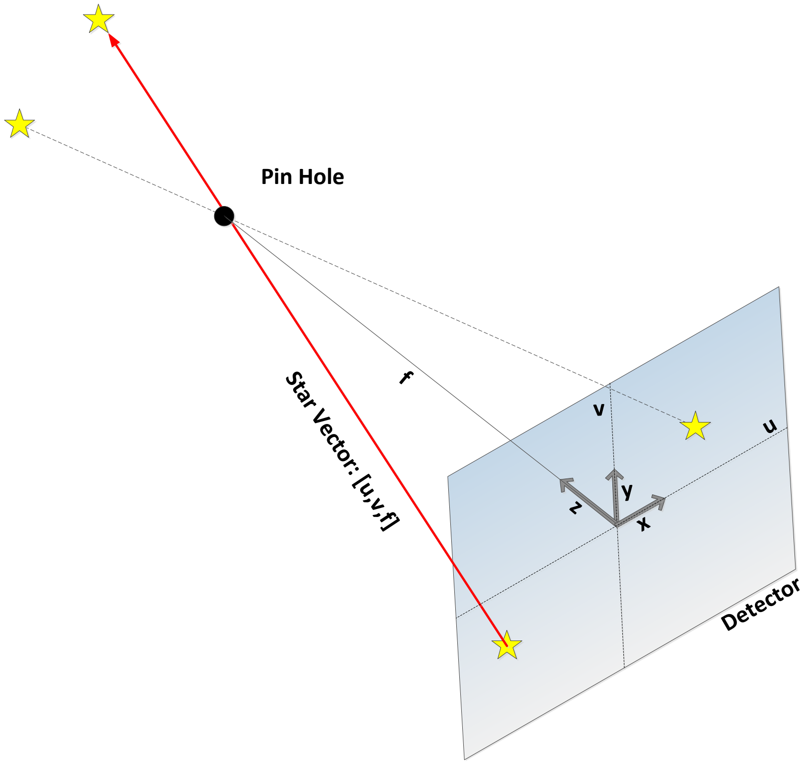


Figure 2-2: Pin Hole Camera Model [13]

Each star centroid is measured in meters by its (u,v) coordinate in the detector fixed frame. Assuming a perfect camera pin-hole model, the unit vector for each star is calculated with Eq 2-5:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{1}{\sqrt{u^2 + v^2 + f^2}} \begin{bmatrix} u \\ v \\ f \end{bmatrix} \tag{2-5}$$

where f is the lens focal length

Recall Euler's rotation from one reference frame to another:

$$DCM = \begin{bmatrix} \cos(\Psi)\cos(\theta) & \sin(\Psi)\cos(\theta) & -\sin(\theta) \\ -\sin(\Psi)\cos(\Phi) + \cos(\Psi)\sin(\theta)\sin(\Phi) & \cos(\Psi)\cos(\Phi) + \sin(\Psi)\sin(\theta)\sin(\Phi) & \cos(\theta)\sin(\Phi) \\ \sin(\Psi)\sin(\Phi) + \cos(\Psi)\sin(\theta)\cos(\Phi) & -\cos(\Psi)\sin(\Phi) + \sin(\Psi)\sin(\theta)\cos(\Phi) & \cos(\theta)\cos(\Phi) \end{bmatrix} \tag{2-6}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = DCM \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{2-7}$$

where Φ is roll in radians about the x-axis, θ is pitch in radians about the y-axis, and Ψ is yaw in radians about the z-axis.

Assuming the small angle approximation ($\sin(\omega) = \omega$; $\cos(\omega) = 1$; $\omega\psi = 0$; $\omega\ and\ \psi \ll 1$), Euler's equation can be reduced:

$$DCM \approx \begin{bmatrix} 1 & \Psi & -\theta \\ -\Psi + \theta\Phi & 1 + \Psi\theta\Phi & \Phi \\ \Psi\Phi + \theta & -\Phi + \Psi\theta & 1 \end{bmatrix} = \begin{bmatrix} 1 & \Psi & -\theta \\ -\Psi & 1 & \Phi \\ \theta & -\Phi & 1 \end{bmatrix} \tag{2-8}$$

Therefore, the simplified formula for rotating unit vectors from one coordinate frame to another is:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & \Psi & -\theta \\ -\Psi & 1 & \Phi \\ \theta & -\Phi & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{2-9}$$

Multiplication reveals formulas for the x, y and z component of each star:

$$x = x_0 + \Psi y_0 - \theta z_0 \tag{2-10}$$

$$y = -\Psi x_0 + y_0 + \Phi z_0 \tag{2-11}$$

$$z = \theta x_0 - \Phi y_0 + z_0 \tag{2-12}$$

Eqs 2-10 through 2-12 are used to find equations for u and v:

$$u = \frac{x}{z} f = \frac{x_0 + \Psi y_0 - \theta z_0}{\theta x_0 - \Phi y_0 + z_0} f \tag{2-13}$$

$$v = \frac{y}{z} f = \frac{-\Psi x_0 + y_0 + \Phi z_0}{\theta x_0 - \Phi y_0 + z_0} f \tag{2-14}$$

Eqs 2-13 and 2-14 are linearized by taking partial derivatives:

First recall the quotient rule:

$$\frac{d}{dx}\left(\frac{Num}{Den}\right) = \frac{(Den)\left(\frac{d}{dx}Num\right) - (Num)\left(\frac{d}{dx}Den\right)}{(Den)^2} \tag{2-15}$$

Next assume that for slow slew rates and a high camera frame rate: $x_0 \approx x$, $y_0 \approx y$, and $z_0 \approx z$.

Take partial derivatives:

$$\frac{\partial u}{\partial \Phi} = \frac{(z)\left(\frac{\partial}{\partial \Phi}x\right) - (x)\left(\frac{\partial}{\partial \Phi}z\right)}{z^2} f = \frac{(z)(0) - x(-y_0)}{z^2} f = \frac{xy_0}{z^2} f \approx \frac{xy}{z^2} f = \frac{x}{z}\frac{y}{z} f = \frac{uv}{f} \tag{2-16}$$

$$\frac{\partial u}{\partial \theta} = \frac{-zz_0 - xx_0}{z^2} f \approx \frac{-z^2}{z^2} f - \frac{x^2}{z^2} f = -f - \frac{u^2}{f} \tag{2-17}$$

$$\frac{\partial u}{\partial \Psi} = \frac{zy_0 - \ (x)(0)}{z^2} f \ \approx \frac{y}{z} f = v \tag{2-18}$$

$$\frac{\partial v}{\partial \Phi} = \frac{zz_0 + yy_0}{z^2} f \ \approx f + \frac{y^2}{z^2} f = f + \frac{v^2}{f} \tag{2-19}$$

$$\frac{\partial v}{\partial \theta} = \frac{(z)(0) - yx_0}{z^2} f \ \approx \frac{-yx}{z^2} f = \frac{-uv}{f} \tag{2-20}$$

$$\frac{\partial v}{\partial \Psi} = \frac{-zx_0 - (y)(0)}{z^2} f \ \approx \frac{-x}{z} f = -u \tag{2-21}$$

The partial derivatives are formed into the Jacobian matrix:

$$J = \begin{bmatrix} \frac{\partial u}{\partial \Phi} & \frac{\partial u}{\partial \theta} & \frac{\partial u}{\partial \Psi} \\ \frac{\partial v}{\partial \Phi} & \frac{\partial v}{\partial \theta} & \frac{\partial v}{\partial \Psi} \end{bmatrix} = \begin{bmatrix} \frac{uv}{f} & -f - \frac{u^2}{f} & v \\ f + \frac{v^2}{f} & \frac{-uv}{f} & -u \end{bmatrix} \tag{2-22}$$

Using the Jacobian, it is possible to find the change in the u and v positions of the i_{th} stars given the rotation angles:

$$\begin{bmatrix} \Delta u_i(k) \\ \Delta v_i(k) \end{bmatrix} = \begin{bmatrix} \frac{u_i(k-1)v_i(k-1)}{f} & -f - \frac{u_i(k-1)^2}{f} & v_i(k-1) \\ f + \frac{v_i(k-1)^2}{f} & \frac{-u_i(k-1)v_i(k-1)}{f} & -u_i(k-1) \end{bmatrix} \begin{bmatrix} \Phi(\mathrm{k}) \\ \theta(k) \\ \Psi(\mathrm{k}) \end{bmatrix} \tag{2-23}$$

where $u_i(k-1)$ is *u* of the i_{th} star in the previous image; $v_i(k-1)$ is *v* of the i_{th} star in the previous image; $\Delta u_i(k) = u_i\,(k) - u_i(k-1)$; $\Delta v_i(k) = \ v_i(k) - \ v_i\,(k-1)$.

The goal of Eq 2-23 is to solve for roll, pitch, and yaw given Δu and Δv for each star. The system has three unknowns (Φ, θ, Ψ) and $(2 * n)$ knowns where *n* is the number of stars. Therefore, assuming at least two stars in each image, recall from linear algebra that the system is over-determined. The best estimate of the solution for an over-determined system is found with the following equations [16]:

$$b = Au \tag{2-24}$$

$$\hat{u} = \ (A^T A)^{-1} A^T b \tag{2-25}$$

where: *u* is an [*n* x 1] matrix of unknowns, *b* is an [*m* x 1] matrix of knowns, *A* is an [*m* x *n*] matrix with more equations than unknowns (*m* > *n*), and $\hat{u}$ is an [*n* x 1] matrix of estimated unknowns.

Therefore, the following equation solves for estimated roll, pitch, and yaw:

$$\begin{bmatrix} \hat{\Phi} \\ \hat{\theta} \\ \hat{\Psi} \end{bmatrix} = \left(\begin{bmatrix} \frac{u_i v_i}{f} & -f - \frac{{u_i}^2}{f} & v_i \\ f + \frac{{v_i}^2}{f} & \frac{-u_i v_i}{f} & -u_i \end{bmatrix}^T \begin{bmatrix} \frac{u_i v_i}{f} & -f - \frac{{u_i}^2}{f} & v_i \\ f + \frac{{v_i}^2}{f} & \frac{-u_i v_i}{f} & -u_i \end{bmatrix} \right)^{-1} \begin{bmatrix} \frac{u_i v_i}{f} & -f - \frac{{u_i}^2}{f} & v_i \\ f + \frac{{v_i}^2}{f} & \frac{-u_i v_i}{f} & -u_i \end{bmatrix}^T \begin{bmatrix} \Delta u_i \\ \Delta v_i \end{bmatrix} \tag{2-26}$$

Eq 2-26 is solved in MATLAB using the analytic formula for the inverse matrix. Define $A = \begin{bmatrix} \frac{u_i v_i}{f} & -f - \frac{{u_i}^2}{f} & v_i \\ f + \frac{{v_i}^2}{f} & \frac{-u_i v_i}{f} & -u_i \\ \dots & \dots & \dots \end{bmatrix}$. *A* will always have three columns and the number of rows equal to twice the number of stars because each star adds two rows to *A*. Define another matrix $M = A^T A$. *M* will always be a 3 x 3 matrix because *A* is *n* x 3. Define each element in matrix *M*:

$M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$. The analytic inverse of matrix *M* is found as follows:

$$\det(M) = (aei) - (afh) - (bdi) + (bfg) + (cdh) - (ceg) \tag{2-27}$$

$$M^{-1} = \begin{bmatrix} \frac{ei - fh}{\det(M)} & \frac{-bi + ch}{\det(M)} & \frac{bf - ce}{\det(M)} \\ \frac{-di + fg}{\det(M)} & \frac{ai - cg}{\det(M)} & \frac{-af + cd}{\det(M)} \\ \frac{dh - eg}{\det(M)} & \frac{-ah + bg}{\det(M)} & \frac{ae - bd}{\det(M)} \end{bmatrix} \tag{2-28}$$

Therefore, Eq 2-26 is solved analytically as follows:

$$\begin{bmatrix} \widehat{\Phi} \\ \hat{\theta} \\ \widehat{\Psi} \end{bmatrix} = (A^T A)^{-1} A^T \begin{bmatrix} \Delta u_i \\ \Delta v_i \\ \cdots \end{bmatrix} = \begin{bmatrix} \frac{ei-fh}{\det(M)} & \frac{-bi+ch}{\det(M)} & \frac{bf-ce}{\det(M)} \\ \frac{-di+fg}{\det(M)} & \frac{ai-cg}{\det(M)} & \frac{-af+cd}{\det(M)} \\ \frac{dh-eg}{\det(M)} & \frac{-ah+bg}{\det(M)} & \frac{ae-bd}{\det(M)} \end{bmatrix} \begin{bmatrix} \frac{u_i v_i}{f} & -f - \frac{{u_i}^2}{f} & v_i \\ f + \frac{{v_i}^2}{f} & \frac{-u_i v_i}{f} & -u_i \\ \cdots & \cdots & \cdots \end{bmatrix}^T \begin{bmatrix} \Delta u_i \\ \Delta v_i \\ \cdots \end{bmatrix}.$$

2.2.4. Tracking Algorithm Steps

The centroid tracking algorithm is summarized as follows:

1. Find the change in each stars' position from the previous image *(k-1)* to the current image *(k)*:

   $\Delta u_i(k) = u_i\,(k) - u_i(k-1);\ \Delta v_i(k) = v_i(k) - v_i\,(k-1).$

2. Build the Jacobian matrix (see Eqs 2-22 and 2-23) where *u* and *v* are the positions of the stars in the previous image *(k-1)*.
3. Solve Eq 2-26 for the estimated roll, pitch, and yaw of the optical detector where *u* and *v* are the positions of the stars in the previous image *(k-1)*. This solves for the estimated rotation angles from the previous image position to the current image position. Dividing the rotation angles by the detector frame rate determines the estimated angular velocity about the optical detector.
4. Build the Jacobian matrix again this time using the *u* and *v* values for the stars in the current image *(k)*.

5. Solve equation 2-23 using the newly build Jacobian and the estimated roll, pitch, and yaw rotation angles to find Δu and Δv of each star. These values are the estimated change in each star's position from the current image *(k)* to the next image *(k+1)*.
6. Add Δu and Δv to the current star locations to find the estimated star locations in the next image. The estimated star locations can be used to focus the centroiding code on portions of the detector where the stars are expected to land instead of centroiding the entire image wasting computation time. Δu and Δv can also be used for the fine optical control loop.

2.3. Tracking Algorithm Analysis

2.3.1. Tracking Simulation Overview

A numerical analysis of the centroid tracking algorithm was conducted in MATLAB Simulink to determine its accuracy at estimating the roll, pitch, and yaw attitude angles of the optical detector from the previous image frame to the current image frame and predicting the centroid locations in the next image frame. Figure 2-3 shows a block diagram of the simulation set-up.

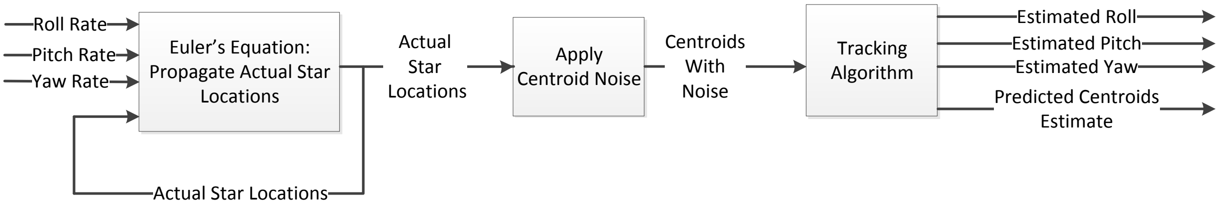


Figure 2-3: Block Diagram of Simulink Tracking Algorithm Analysis

The simulation is given a constant roll, pitch, and yaw attitude rate as well as initial centroid locations. Running at 12 Hz (the star tracker frame rate), the centroids are converted to unit vectors, propagated, and converted back to centroid coordinates using Eqs 2-5, 2-7, 2-13, and 2-14. Gaussian random noise blocks with a mean of 0 pixels and a standard deviation of 0.1

pixels add random centroid noise to the actual star locations. The tracking code uses persistent variables to store the centroids from the previous image frame. As described in steps 1 through 6 of the tracking algorithm in section 2.2.4, the centroids from the current frame and previous frame are used to find an estimate of the roll, pitch, and yaw of the optical detector. This roll, pitch, and yaw estimate is used to predict the location of the centroids in the next image frame (predicted centroids estimate). The tracking algorithm is calculated using the actual centroids and the centroids with noise in order to analyze the effect of centroid errors.

2.3.2. Analysis: Number of Centroids

The tracking algorithm is first analyzed to determine how the number of centroids on the detector affects the angular velocity estimation error. Simulations are run for 5 seconds with a spacecraft rotation rate of 30 degrees per minute. Each case varies the number of centroids which are randomly placed on the detector. Each case is run for 1,000 trials. The three sigma angular velocity estimation error (average plus three times the standard deviation of the 1,000 trials) is plotted for each case. Roll and pitch are defined as the off-boresight axes. Yaw is defined as the boresight axis. Figure 2-4 shows the results without centroid noise and Figure 2-5 shows the results with 0.1 pixel standard deviation centroid noise.

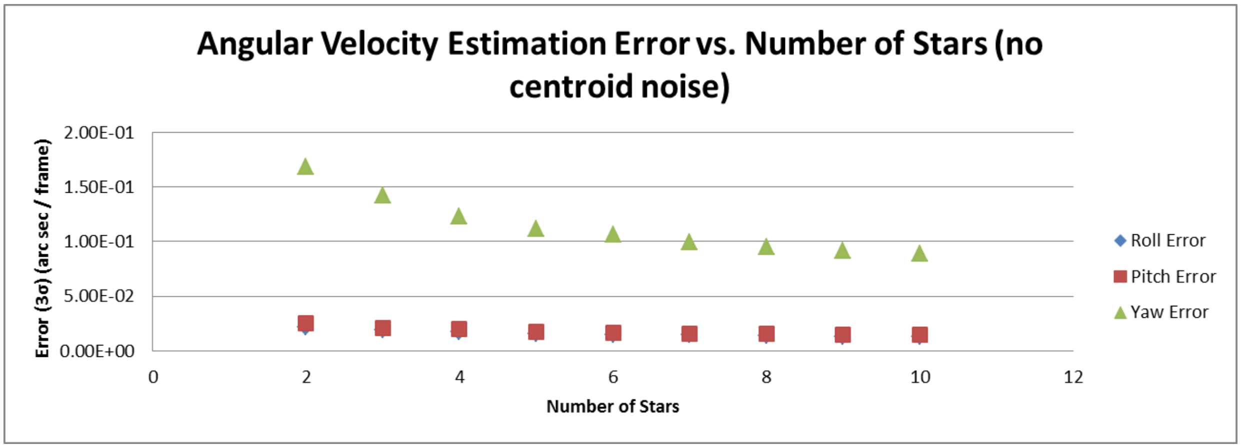


Figure 2-4: Tracking Code Angular Velocity Estimation Error vs. Number of Stars without Centroid Noise

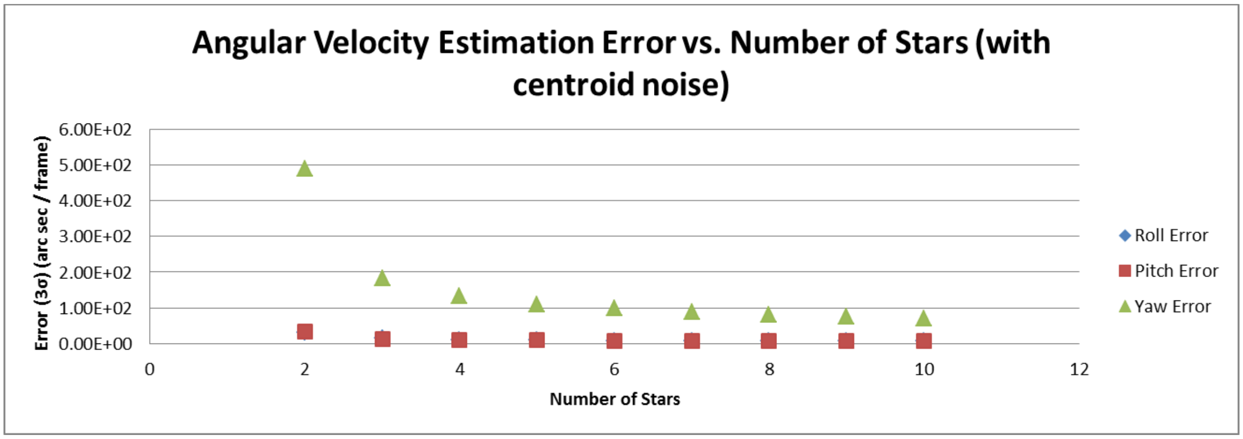


Figure 2-5: Tracking Code Angular Velocity Estimation Error vs. Number of Stars with 0.1 Pixel Standard Deviation Centroid Noise

As the number of stars on the detector increases the angular velocity estimation error decreases non-linearly. When more than eight stars are present on the detector, adding additional stars has little effect on improving the angular velocity estimation error. The estimation error with 0.1 pixel centroid error is about 2.5 to 3 orders of magnitude greater than the error without centroid noise. Therefore, centroid noise contributes significantly to angular velocity estimation error. The results show that estimation error about the yaw axis is up to one order of magnitude greater than roll and pitch. This result is not surprising because the yaw axis is the boresight axis, and star trackers are least accurate about boresight [9]. When centroid noise is considered, the error about boresight improves significantly with more than four stars. Further analysis will assume eight stars are on the detector.

2.3.3. Analysis: Star Tracker Slew

Next, the tracking algorithm is analyzed to determine how the spacecraft rate affects the attitude rate estimation error and centroid prediction error. Each trial is run with 8 centroids randomly placed at initial locations on the detector. Each case varies the spacecraft rate. The simulation was not set-up to drop stars as they fall off the detector. Therefore, stars can move an infinite distance from the center of the detector and still be included in the tracking algorithm. So, as the spacecraft rate increases the simulation time was decreased to prevent stars moving too far from the center of the detector. Simulation time is an important consideration because the tracking algorithms were developed by linearizing the combined Euler and camera pin-hole equation (Eqs 2-13 and 2-14). As stars move further from the center of the detector, errors increase due to the non-linear nature of the equation. Because the stars were randomly placed on the detector, some stars were initially placed at the edge and therefore do move far enough from the center of the detector that they would normally fall off. However, these errors are minimized due to the 1,000 Monte Carlo trials. The three sigma attitude error is plotted for each case. Figure 2-6 and Figure 2-7 show the angular velocity estimation error. Figure 2-8 and Figure 2-9 show the errors in predicting the centroid locations.

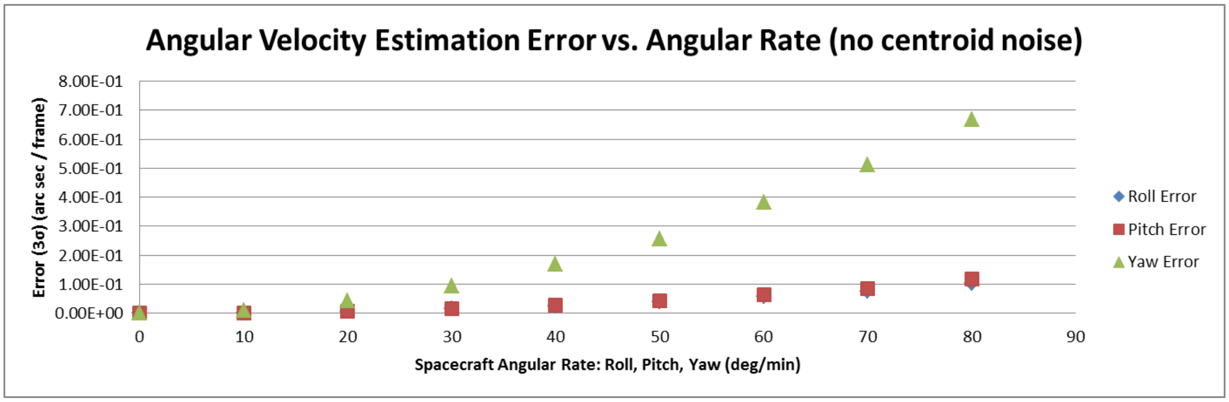


Figure 2-6: Tracking Code Angular Velocity Estimation Error vs. Angular Rate without Centroid Noise

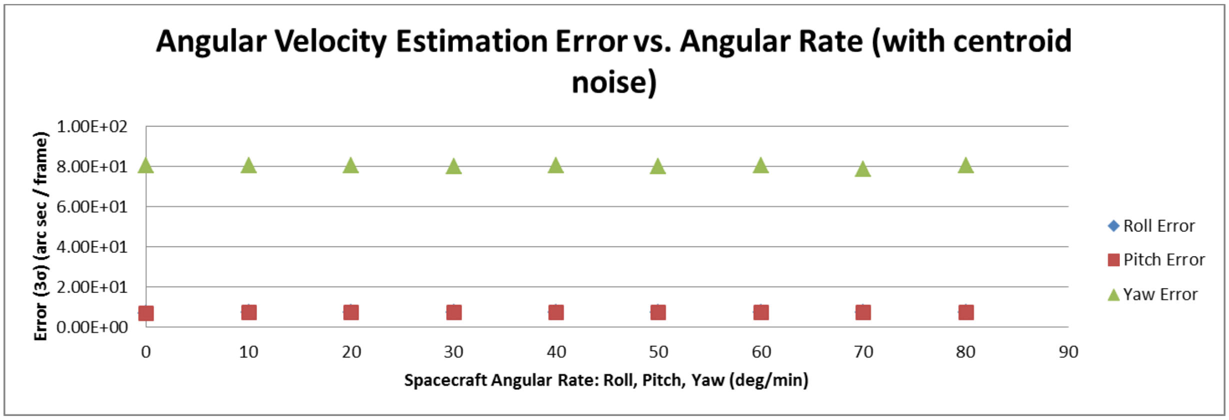


Figure 2-7: Tracking Code Angular Velocity Estimation Error vs. Angular Rate with 0.1 Pixel Standard Deviation Centroid Noise

Without centroid noise, attitude estimation error increases non-linearly. When centroid noise is factored into the tracking algorithm, the attitude estimation error is nearly constant as rate increases. Even with zero spacecraft rate, the attitude estimation error with centroid noise is several times greater than the attitude estimation error without centroid noise. Therefore, a centroid noise of just 0.1 pixels has a much larger effect on attitude estimation error than the spacecraft rate. The figures below show similar results for centroid prediction error.

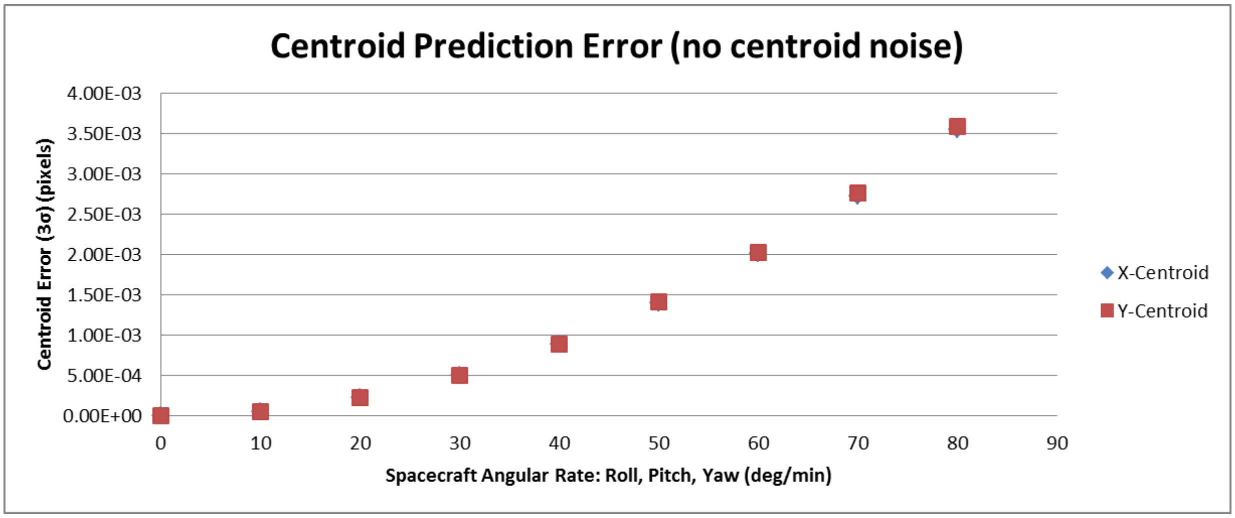


Figure 2-8: Tracking Code Centroid Prediction Error vs. Angular Rate without Centroid Noise

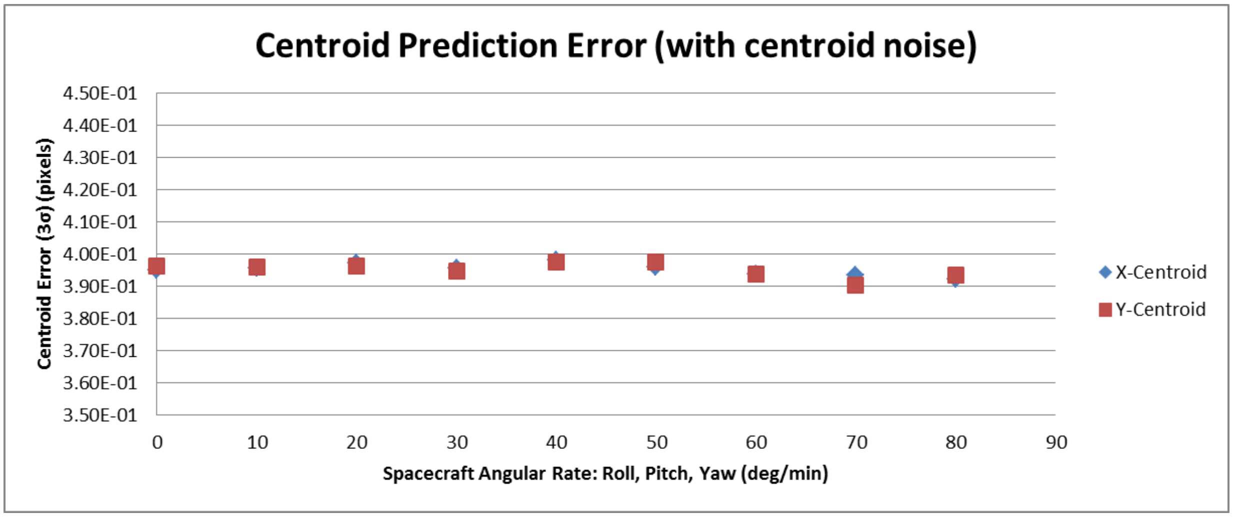


Figure 2-9: Tracking Code Centroid Prediction Error vs. Angular Rate with 0.1 Pixel Standard Deviation Centroid Noise

An estimate of roll, pitch, and yaw rate is necessary to predict the new centroid locations as shown in Eq 2-23. Therefore, the centroid prediction error is related to the attitude rate estimation error. Without centroid noise, as spacecraft rate increases, the prediction error increases non-linearly. With centroid noise, the prediction error is nearly constant. Again, centroid noise is the dominant error source. With 0.1 pixel centroid error, the new centroid locations are predicted to within 0.4 pixels.

Chapter 3

3. Star Tracker Simulation

3.1. Motivation and Purpose

When designing an optical high precision pointing system many factors contribute to the pointing performance. It may be necessary to consider several system architectures and trade several hardware options. One cost effective method to perform hardware trade studies and to analyze performance is to create a computer simulation of the optical hardware. There are several noise sources from the selected detector and theoretical optical limitations which affect the system performance. The simulation must be robust to capture the dominant noise sources which impact the downstream pointing precision. Therefore, the goal is to develop a robust model of the camera optics and detector in MATLAB Simulink. The model should generate simulated star field images which include optical and detector noise. The images can be analyzed with simulations to characterize the centroid error.

3.2. Star Tracker Noise Sources

3.2.1. Noise Sources Overview

There are several optical and detector noise sources that must be simulated in the camera model. For the purpose of organizing the simulation, the noise sources can be placed into categories as shown in the following error tree. The error tree is based on Refs [9, 18, 19, 20].

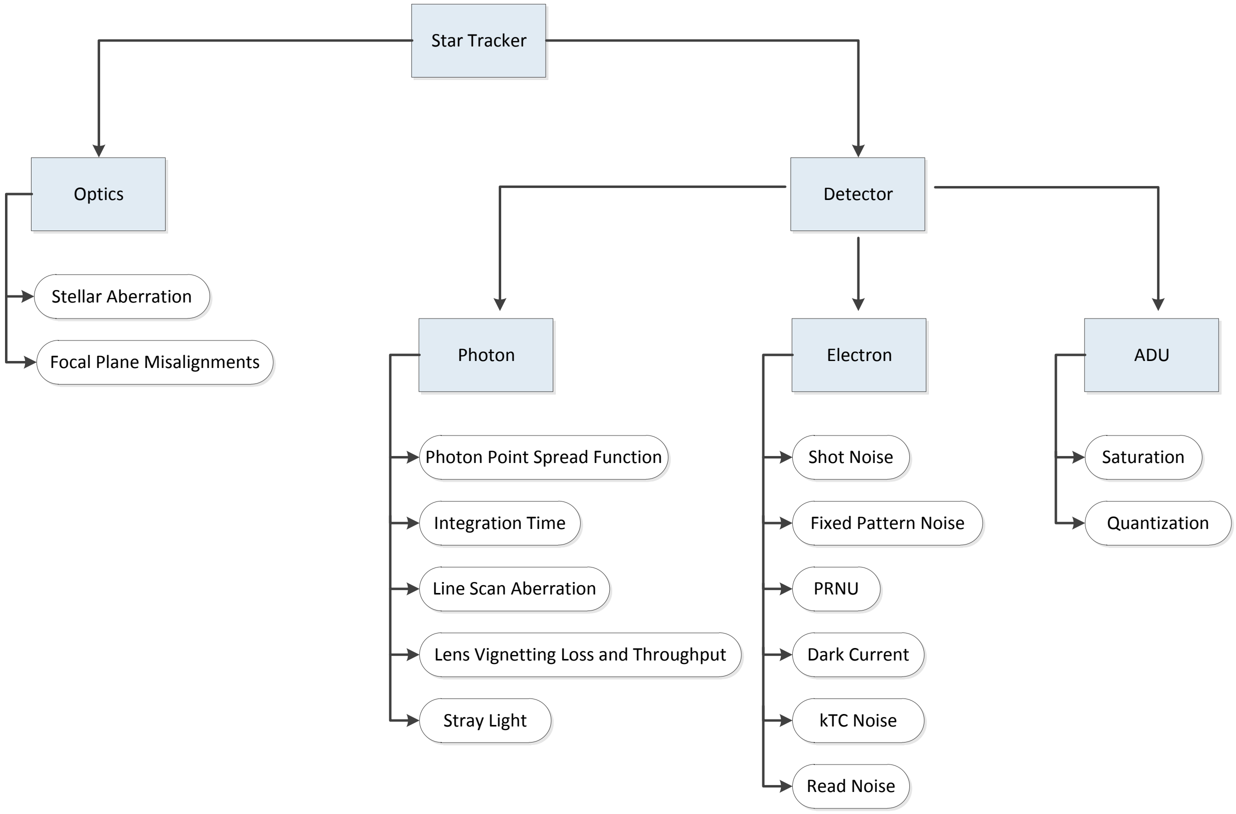


Figure 3-1: Star Tracker Model Error Tree

The noise sources are split into two main groups including optics and detector.

3.2.2. Optics Model Overview

The optics model includes stellar aberration and focal plane misalignments. These error sources determine where a star lands on the detector. Stellar aberration is a distortion caused by the spacecraft absolute velocity (spacecraft orbital velocity plus Earth's orbital velocity) compared to the speed of light. Focal plane misalignments are errors between the actual location the detector is mounted and the modeled location. When building the spacecraft optical system, there will be small rotational and linear misalignment errors between the designed location and built location. These errors should have little effect on the pointing performance because the software and star catalogue can be adjusted to model the actual location of the detector. The error that does matter is the error between where the detector is actually located and where we think it is located. This is a misalignment calibration error which does add noise to the model. The stellar aberration and misalignment calibration error add a bias to the centroid positions because these errors affect the actual location of the star on the detector before detector noise is applied.

3.2.3. Detector Noise Overview

The detector model includes noise that affects the individual pixel values from the image. The detector group is split into three categories including photon, electron, and ADU. The pixels on a CMOS detector act like wells. When a photon strikes a pixel, an electron is produced. Throughout the duration of the integration time, the pixel wells fill up with electrons as more and more photons strike the pixel. When integration terminates, the CMOS electronics measure each pixel charge as a voltage. An analog-to-digital converter outputs the voltage reading as an analog-to-digital unit (ADU). The ADU count for each pixel is a measure of how much light fell on each pixel during the integration time [17, 18]. The image data is stored as a matrix with each element containing the ADU value of the corresponding pixel. Each group in the error tree detector category corresponds to one step in the detector imaging process. The photon group determines how many photons strike each pixel within the integration time. The electron group includes the error from electrical interference and noise as the photons are converted to electrons. The ADU group captures the error in converting from voltages to ADU counts.

The photon group includes the photon point spread function (PSF), integration time, line scan aberration, lens vignetting loss and throughput, and stray light. Star light is defocused over several pixels. The PSF is modeled as a Gaussian function which spreads the star light over several pixels. The integration time is similar to the shutter on a traditional camera. It determines how long the period of photon collection lasts for each image. A longer integration time results in more photons collected and a brighter image. Some detectors have a rolling shutter. This means that the camera is read out one row at a time. Therefore, there is a slight time delay from when the top and bottom rows are integrated. This effect is referred to as line scan aberration. Lens vignetting and throughput refer to loss of photons as they pass through the lens considering the lens is thicker at the edges compared to the center. Reflections from the moon, Earth, and other parts of the spacecraft will result in additional photons striking the detector. These photons are classified as stray light.

The electron group includes shot noise, fixed pattern noise, photo response non-uniformity (PRNU), dark current, thermal (kTC) noise, and read noise. These error sources are all caused by electrical noise on the sensitive imager electronics. Some of the noise sources such as shot noise and dark current are Poisson distributions. However, a Gaussian distribution is a close approximation. Therefore, for simplicity, all the electrical noise sources are modeled as Gaussian. These noise sources depend on the specific detector and can be found in a data sheet or through hardware lab tests.

The ADU group includes saturation and quantization. Each pixel can only accept a limited number of photons which results in a maximum ADU count. Long integration times can cause overexposure resulting in saturated pixels. Saturation increases centroid error because the shape of the star is degraded as the top of the Gaussian is essentially chopped off. Quantization occurs when the electrons are converted to ADU counts and rounded to the nearest ADU.

3.3. Star Tracker Model Implementation

3.3.1. Implementation Overview

The star tracker model is written in MATLAB Simulink for quick implementation and testing. In Chapter 6 it will be autocoded to C-code with a MBD framework developed at Draper Laboratory. The algorithm for simulating images is split into the two main categories from the error tree including the optics model and detector model. Figure 3-2 through Figure 3-4 show block diagrams of the star tracker simulation.

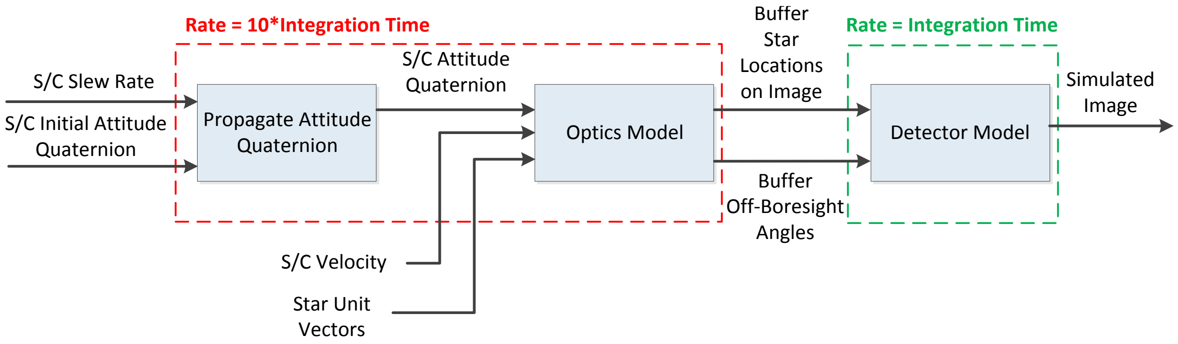

Rate = 10*Integration Time
S/C Slew Rate
S/C Initial Attitude Quaternion
Propagate Attitude Quaternion
S/C Attitude Quaternion
Optics Model
S/C Velocity
Star Unit Vectors
Buffer Star Locations on Image
Buffer Off-Boresight Angles
Rate = Integration Time
Detector Model
Simulated Image

Figure 3-2: High Level Block Diagram of Star Tracker Simulation

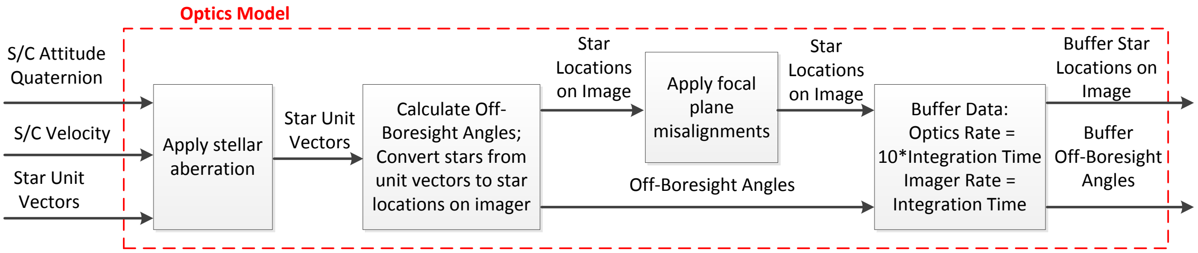

Optics Model
S/C Attitude Quaternion
S/C Velocity
Star Unit Vectors
Apply stellar aberration
Star Unit Vectors
Calculate Off-Boresight Angles; Convert stars from unit vectors to star locations on imager
Star Locations on Image
Apply focal plane misalignments
Star Locations on Image
Off-Boresight Angles
Buffer Data: Optics Rate = 10*Integration Time Imager Rate = Integration Time
Buffer Star Locations on Image
Buffer Off-Boresight Angles

Figure 3-3: Detailed Block Diagram of Optics Model

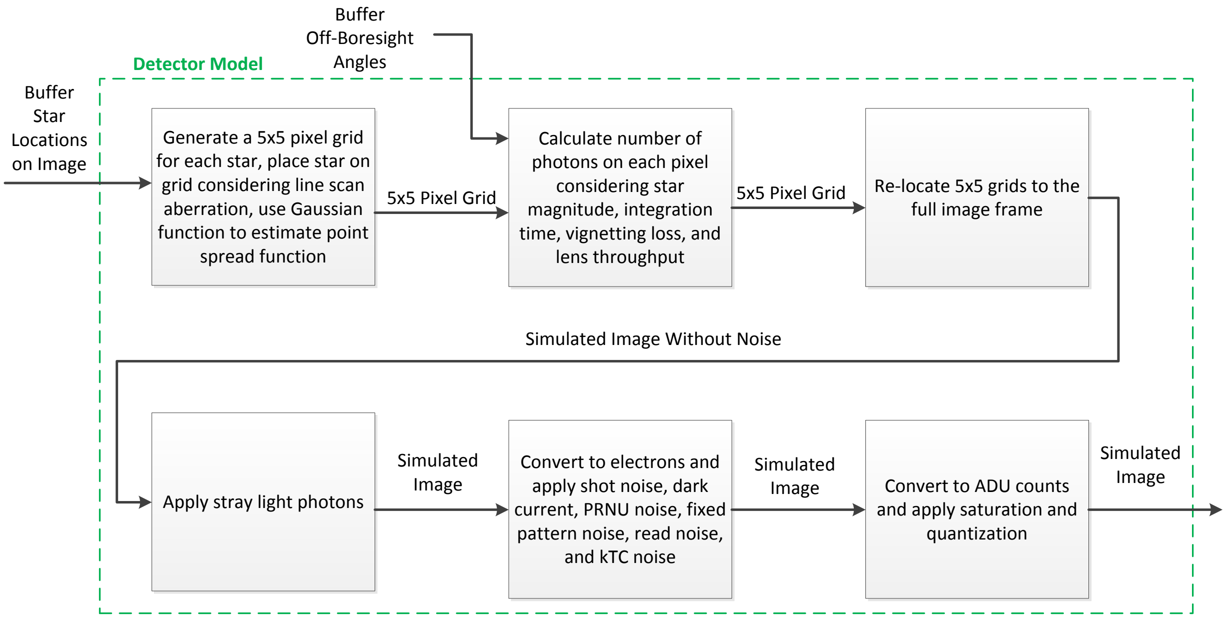

Buffer
Off-Boresight
Angles
Detector Model
Buffer
Star
Locations
on Image
Generate a 5x5 pixel grid for each star, place star on grid considering line scan aberration, use Gaussian function to estimate point spread function
5x5 Pixel Grid
Calculate number of photons on each pixel considering star magnitude, integration time, vignetting loss, and lens throughput
5x5 Pixel Grid
Re-locate 5x5 grids to the full image frame
Simulated Image Without Noise
Apply stray light photons
Simulated
Image
Convert to electrons and apply shot noise, dark current, PRNU noise, fixed pattern noise, read noise, and kTC noise
Simulated
Image
Convert to ADU counts and apply saturation and quantization
Simulated
Image

Figure 3-4: Detailed Block Diagram of Detector Model

Start with Figure 3-2. The star tracker simulation first runs a MATLAB script which initializes a constant spacecraft slew rate and constant spacecraft absolute velocity. These values are initialized as vectors with three elements for the three spacecraft axes. The absolute velocity should change as the spacecraft orbits the earth. However, for the purpose of simplifying the simulation for parametric analysis, the velocity is specified as a constant. The MATLAB script also specifies the initial spacecraft attitude quaternion and the initial star locations. The star locations are unit vectors referenced to the detector. The first step of the Simulink model is to use the slew rate to update the spacecraft attitude quaternion. The second step is to run the optics model to determine the star centroid locations on the detector and off-boresight angles considering stellar aberration and detector misalignments. The off-boresight angles are used later to calculate vignetting. The last step is to run the detector model to produce a simulated image.

Notice the simulation runs at two different rates. Here rates are defined as simulation time steps in Hz. The detector model runs at the integration time while the propagate attitude quaternion and optics model run at 10 times the integration time. For example, assume the star tracker integration time is 83.3 milliseconds or 12 Hz. The detector model runs at 12 Hz and the optics model runs at 120 Hz. The purpose of the two rates is to account for stars smearing across the detector as the spacecraft slews during the integration time. The optics model runs 10 iterations (120 Hz / 12 Hz = 10) for every time the detector model generates a simulated image. The centroid locations and off-boresight angles produced from the optics model are stored in a buffer. The buffer holds up to 10 propagated star locations for each star on the detector. The buffer is then passed at 12 Hz to the detector model. Each time the detector model runs it has access to all the star locations as the stars slewed across the detector for the current image. In other words, for each image the stars slew across the detector for 83.3 milliseconds. By running the optics model at 120Hz we can discretize each smeared star into 10 discrete locations. The 10 discrete locations are passed to the detector model so that it can simulate the star smear.

3.3.2. Detector Model

3.3.2.1. Detector Model Overview

The detector model uses small pixel grids for each star in the buffer to simulate the PSF and photon count from the star. The grid is then placed in its corresponding location in the full frame image. Once all the grids for each centroid position are placed in the full frame image, the remaining detector noise is calculated on the full image.

3.3.2.2. Point Spread Function

The first step in the detector model is to calculate the PSF of each propagated star location in the buffer. Refer to the first block in Figure 3-4. A small grid is used rather than the full image because the PSF of a star only covers a few pixels. The baseline grid size is 5x5 pixels. The PSF is simulated with a Gaussian function which is a close approximation to the actual star PSF.

$$G(X) = e^{-J} \tag{3-1}$$

$$J = \frac{1}{2}(X - \bar{X})^T \Sigma^{-1} (X - \bar{X}) \tag{3-2}$$

where $G(X)$ is the Gaussian function solved at discrete points $X = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$ on the grid; $\bar{X} = \begin{bmatrix} x_c \\ y_c \end{bmatrix}$ where x_c and y_c are the coordinates for the center of the star; $\Sigma = \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix}$ where σ is the standard deviation calculated as $\sigma = \frac{FWHM}{2\sqrt{2\log(2)}}$; *FWHM* is the PSF full width at half maximum

So, for each star location in the buffer, a Gaussian function is centered at the decimal location of the star at the middle of the 5x5 pixel grid. For example, if one centroid location in the buffer is (102.23, 54.78), the PSF is simulated in a 5x5 pixel grid at location (0.23, 0.78) in the center pixel. The goal is to solve the Gaussian function at each pixel in the grid. For improved accuracy, each pixel in the 5x5 grid is also divided into a sub-grid nominally 5x5. The Gaussian function is solved at the discrete sub-grid locations within each pixel. The value of each pixel equals the average value of the discrete locations within each pixel. Finally the pixel values are normalized by dividing each pixel by the sum of the pixels. In MATLAB code, the grid is saved as a 5x5 matrix. The value of each element corresponds to the pixel value. The pixels are normalized because the Gaussian grid is used later as a weighting function to determine how many photons strike each pixel.

3.3.2.3. Star Light Photon Count

The second step in the detector model is to calculate the number of photons that fall on each pixel in the 5x5 grid. Refer to the second block in Figure 3-4. Eqn 3-3 calculates the total number of photons that fall on each 5x5 grid.

$$\mathrm{P} = \frac{\phi * 10^{-0.4*Mag} * A * n * \gamma * \tau}{\rho} \tag{3-3}$$

$$\gamma = 1 - \nu * \beta \tag{3-4}$$

where P is the total number of photons, ϕ is the flux vega, *Mag* is the star magnitude, *A* is the aperture area, n is the throughput, γ is the vignetting loss, τ is the integration time, ρ is the number of discrete centroid locations which equals 10 or (optics model rate / detector model rate), ν is the vignetting, and β is the off-boresight angle

Dividing by the number of buffered locations is necessary because a 5x5 grid is calculated for each propagated star location in the buffer. The numerator in Eq 3-3 calculates the total number of photons collected for each star on the full frame image. Dividing by the number of discrete locations that make up a star in the full frame image calculates the number of photons at each discrete propagated location. The next step is to create a 5x5 grid with each pixel value equal to the correct number of photons that make up a star. The photon grid is created by multiplying each pixel in the PSF grid by the total number of photons:

$$\mathrm{P}_G = G * \mathrm{P} \tag{3-5}$$

where P_G is the photon grid, G is the PSF grid calculated previously, P is the total number of photons

Therefore, the PSF grid is essentially used as a unique scale factor for each pixel in the grid to find the proportion of the total photons that land on each pixel.

3.3.2.4. Clean Full Frame Image

Step three in Figure 3-4 is to relocate each photon grid to the proper star location in the full frame image. In the example above, one of the centroid locations in the buffered centroid list was at location (102.23, 54.78). The PSF was calculated with the Gaussian centered at location (0.23, 0.78) in the center pixel of the grid. The 5x5 photon grid needs to be placed such that the center pixel in the grid is located at (102, 54) in the full frame image. The full frame image is initialized as a matrix of zeros with row and column size corresponding to the full frame detector size. Each element in the matrix corresponds to a pixel value in the detector. Steps 1 through 3 in Figure 3-4 are repeated for each propagated centroid location in the buffered centroid list. Once complete, the full frame image is a matrix of zeros except at the centroid locations where the photon grids were injected. This matrix simulates what a star image would look like if there was no detector noise or stray light.

3.3.2.5. Stray Light Photon Count

Step four of the detector model adds stray light to the clean image. The total number of stray light photons is calculated as follows:

$$\zeta = \kappa * \phi * 10^{-0.4*zMag} * A * n * \tau * cols * rows * \left(\frac{1}{\ell * a2R}\right)^2 \qquad (3\text{-}6)$$

where ζ is the total number of stray light photons, κ is the stray light multiplier, ϕ is the flux vega, *zMag* is the zodiacal light magnitude, *A* is the aperture area, $\mathcal{n}$ is the throughput, τ is the integration time, *cols* is the total number of pixel columns and *rows* is the total number of pixel rows in the full frame image, ℓ is the focal length, and $a2R$ is a unit conversion for the number of radians in an arc second calculated as: $a2R = 1\, arcsec * \frac{1\, deg}{3600\, arcsec} * \frac{\pi\, rad}{180\, deg} = 4.8481 * 10^{-6}\, rad$

The total number of stray light photons is added to every pixel in the clean image.

3.3.2.6. Electron Noise

Step five is to convert the image to electrons then apply the detector noise shown in block 5 of Figure 3-4. The image is converted to electrons by multiplying each pixel by the efficiency value h found in the data sheet. Once converted to electrons, various noise sources are modeled.

$$I = h * I_o \tag{3-7}$$

Shot noise:

$$I = I_o + \ sqrt(I_o).* randn(size(I_o)) \tag{3-8}$$

Dark current:

$$I = I_o + (i_d * \tau * ones(size(I_o))) + sqrt(i_d * \tau) * randn(size(I_o)) \tag{3-9}$$

PRNU:

$$I = I_o.* \left(1 + (\alpha * Randn(size(I_o)))\right) \tag{3-10}$$

Fixed pattern noise:

$$I = I_o + (\Omega * Randn(size(I_o))) \tag{3-11}$$

Read noise:

$$I = I_o + \Gamma * randn(size(I_o)) \tag{3-12}$$

kTC noise:

$$I = I_o + \Lambda * randn(size(I_o)) \tag{3-13}$$

where I is a matrix in which each element represents the corresponding pixel value in the full frame image, I_o distinguishes the matrix before a specific noise is added, h is the photon to electron conversion efficiency, i_d is the dark current, τ is the integration time, α is the PRNU noise, Ω is the fixed pattern noise, Γ is the read noise, Λ is the kTC noise.

The noise sources are modeled one at a time. For example, in order to simulate shot noise and dark current only, use Eq 3-8 to calculate shot noise then set I_o equal to I and use Eq 3-9 to calculate dark current. Notice the noises are not calculated on separate images then added together. Instead, after shot noise was calculated, the shot noise image was used as I_o in the calculation for dark current.

The equations for detector noise introduced above are written in MATLAB syntax for ease of understanding. The multiplier $(.*)$ refers to element by element multiplication and $ones(size(I_o))$ creates a matrix the same size as the image with each element equal to one. The detector noise is simulated as Gaussian random noise. The *randn* command in MATLAB generates a random number with mean and standard deviation equal to one. Multiplying *randn* by a constant creates a random number with standard deviation equal to the constant. Adding a constant to *randn* creates a random number with mean equal to the constant.

$$Gaussian\ Noise = Mean + \ \sigma * randn(size(I_o)) \tag{3-14}$$

The equation above creates a matrix the same size as the image called *Gaussian Noise*. Each element in the matrix was created from a random number generator with mean equal to *Mean* and standard deviation equal to σ. The standard deviation values used in Eqs 3-8 through 3-13 can be found in the detector data sheet provided by the manufacturer. Table 3-1 below shows values from the Cypress HAS 2 CMOS detector which are used as nominal values in the detector model.

Table 3-1: Cypress HAS 2 CMOS Data Sheet Values [23]

Cypress HAS 2 CMOS Data Sheet Values					
Characteristic	**Min**	**Typ**	**Max**	**Unit**	**Remarks**
Image sensor format	N/A	1024 x 1024	N/A	pixels	
Pixel Size	N/A	18	N/A	µm	
Spectral Response (Quantum efficiency x Fillfactor)	N/A	33.3	N/A	%	Measured average over 400 - 900nm
Global fixed pattern noise standard deviation (Hard reset)	N/A	115	180	e-	With DR/DS
Global photo response non uniformity, standard deviation	N/A	1.8	5	%	Of average response
Average dark signal	N/A	190	400	e-/s	At 25 ± 2°C die temp, BOL
Temporal noise (NDR Soft reset)	N/A	75	100	e-	read noise
kTC noise	N/A	0	N/A	e-	zero when running in Correlated Double Sampling-Non-Destructive Readout mode
Charge to voltage conversion factor	13	14.8	15.6	µV/e-	
Output amplifier gain	N/A	1	N/A		
ADC ideal input range	0.85	N/A	2.0	V	voltage swing: 2.0 - 0.85
ADC resolution	N/A	12	N/A	bit	10 bit accuracy at 5 Msamples / sec

When the simulation is run, new random numbers are generated for each image with the exception of PRNU and fixed pattern noise. PRNU and fixed pattern noise are characterized as random noise however they remain constant for each image once the detector is turned on. Therefore, the random number matrices for PRNU and fixed pattern noises are generated once at the start of the simulation using persistent variables. Eqs 3-10 and 3-11 show the MATLAB syntax *Randn* with a capital R to indicate this difference.

3.3.2.7. ADU Noise

Step 6 is to convert the image from electrons to ADU counts and apply saturation and quantization. The image is converted to ADU counts as follows:

$$I = \frac{2^{\mu} - 1}{V_s} * A_g * I2V * I_o \qquad (3\text{-}15)$$

where μ is the resolution, V_s is the voltage swing, A_g is the amplifier gain, and $I2V$ is the charge to voltage conversion

Saturation is accounted for by setting the minimum pixel value to zero and the maximum pixel value to $(2^{\mu} - 1)$. Quantization is simulated by rounding each pixel value to the nearest integer.

3.3.3. Star Tracker Model Output

The output of the star tracker model is a simulated star field image. This image is passed to the software which runs a centroiding algorithm to determine the centroid locations. The noise values that contributed to building the image will affect the centroid error. In order to accurately characterize the centroid error, the images need to be validated for accuracy. Validation can occur through the use of hardware testbeds by taking actual images and comparing to the simulated images.

Chapter 4

4. Fast Centroiding Algorithm

4.1. Fast Centroiding Algorithm Software Architecture

4.1.1. Fast Centroiding Algorithm Challenges

In order to develop a fast centroiding algorithm based on the tracking algorithm described in Section 2.2, several factors had to be considered. First, the tracking algorithm uses the centroid locations from the previous images to predict the centroid locations in the current image. The tracking code must be initialized with centroids. Second, the tracking code requires at least four stars for adequate attitude rate accuracy. The tracking code must handle cases in which too few stars land on the detector as stars are added and subtracted due to the star tracker slew. Third, the tracking code must subtract the current centroid locations from the previous locations. The centroid code reads out centroids in order from the top of the image to the bottom of the image. With this constraint, the tracking code must match stars from the current frame to the previous frame when the order of the centroids changes and stars are added or subtracted due to the star tracker slew.

4.1.2. Fast Centroiding Algorithm Overview

Figure 4-1 below shows a high level block diagram of the fast centroiding algorithm software architecture. Figure 4-2 below shows the software architecture in more detail.

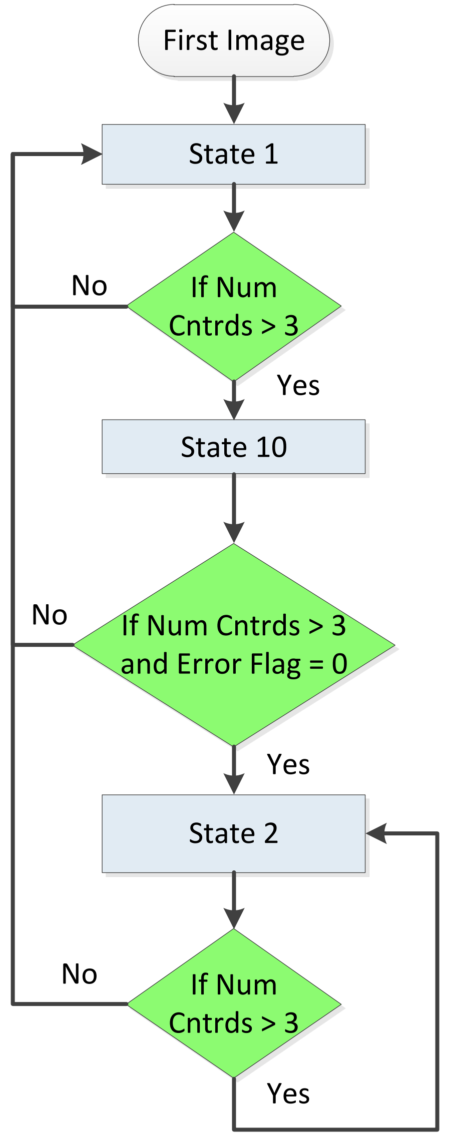
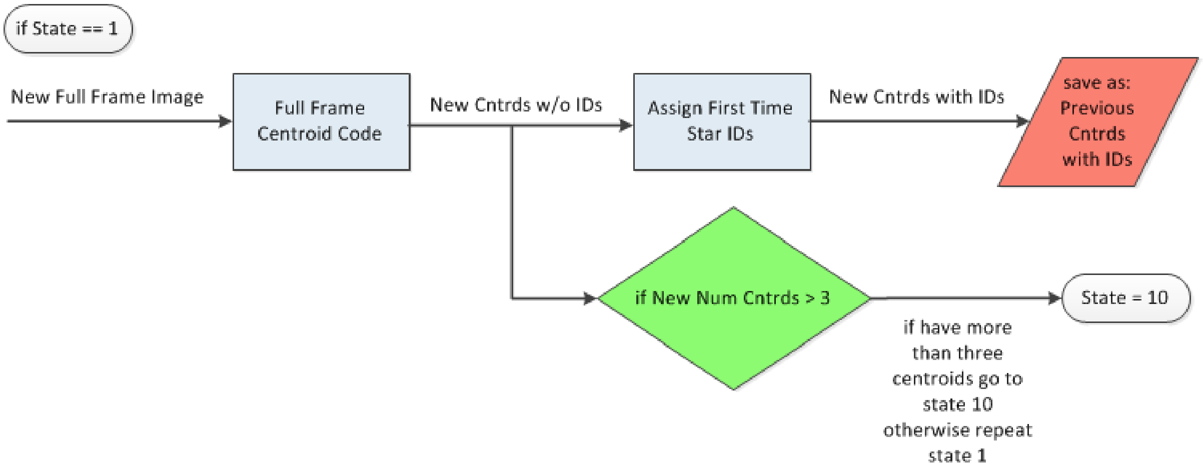
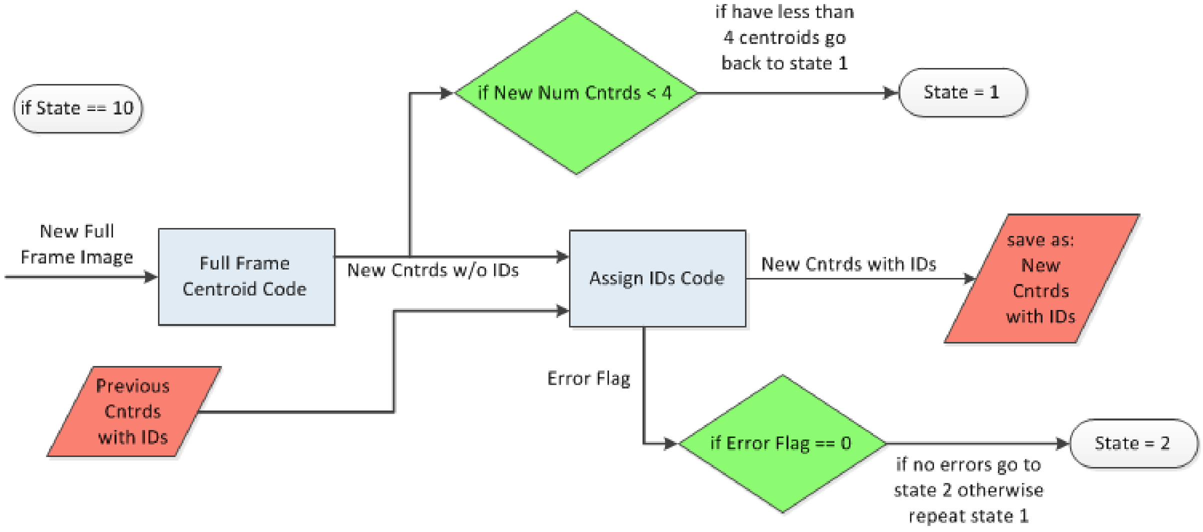


Figure 4-1: High Level Software Architecture for Fast Centroiding Algorithm


if State == 1
New Full Frame Image
Full Frame Centroid Code
New Cntrds w/o IDs
Assign First Time Star IDs
New Cntrds with IDs
save as: Previous Cntrds with IDs
if New Num Cntrds > 3
State = 10
if have more than three centroids go to state 10 otherwise repeat state 1


if have less than 4 centroids go back to state 1
if State == 10
if New Num Cntrds < 4
State = 1
New Full Frame Image
Full Frame Centroid Code
New Cntrds w/o IDs
Assign IDs Code
New Cntrds with IDs
save as: New Cntrds with IDs
Previous Cntrds with IDs
Error Flag
if Error Flag == 0
State = 2
if no errors go to state 2 otherwise repeat state 1

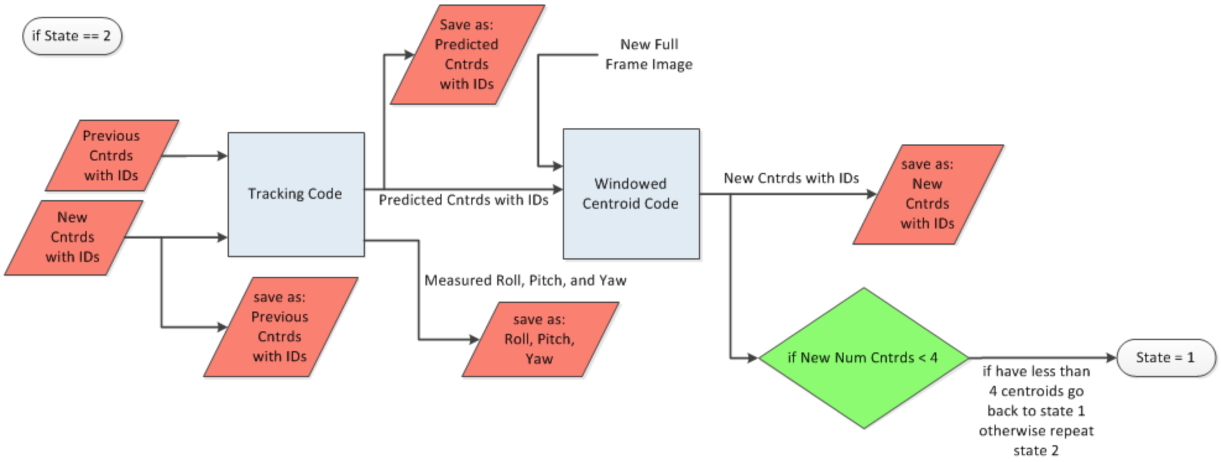

if State == 2
Save as: Predicted Cntrds with IDs
New Full Frame Image
Previous Cntrds with IDs
New Cntrds with IDs
Tracking Code
Predicted Cntrds with IDs
Windowed Centroid Code
New Cntrds with IDs
save as: New Cntrds with IDs
save as: Previous Cntrds with IDs
Measured Roll, Pitch, and Yaw
save as: Roll, Pitch, Yaw
if New Num Cntrds < 4
State = 1
if have less than 4 centroids go back to state 1 otherwise repeat state 2

Figure 4-2: Detailed Software Architecture for Fast Centroiding Algorithm

The fast centroiding algorithm has three states that are called by a switch case statement in C-code: state 1, state 10, and state 2. The state value is saved as a global variable. Each time the star tracker takes an image, the current state is executed. Organizing the code into multiple states adds the ability for the fast centroiding algorithm to respond to the dynamic environment of an Earth orbiting spacecraft. As stars are added and subtracted from the detector, the algorithm can determine which state should be executed in the next image frame.

Two new functions are included in this architecture. First, the centroiding code is split into a full frame centroiding algorithm and a windowed centroiding algorithm. The full frame centroiding algorithm used for the LIS mode searches the entire image frame for centroids as described in section 2.1.3.3. The tracking algorithm executed during the tracking mode uses centroid data from previous image frames to determine the predicted centroid locations in the current image frame. The windowed centroiding algorithm runs the COM calculation on image windows centered at the predicted star locations. Therefore, the windowed centroiding algorithm differs from the full frame centroiding algorithms because it does not use a signal threshold to determine each ROI location. Because the windowed algorithm does not search the entire image for centroids, the processing time required to output centroids from each image is significantly reduced.

Another function added to the software architecture is a star identification algorithm. This algorithm assigns an identification (ID) number to each star read out from the centroid code. As stars are retired, the IDs for those stars are retired and as stars are added they are assigned new IDs. The tracking code uses the star IDs to accurately match stars from the current frame to the previous frame when estimating star tracker slew rate.

4.1.3. Fast Centroiding Algorithm Step-by-Step Details

The algorithm initializes to state 1 (LIS mode). State 1 runs the full frame centroiding algorithm. The new centroids are passed to a function called “Assign First Time Star IDs”. The centroid data is saved in a global variable called “Previous Centroids”. If the centroid code found four or more stars, the next image will move on to state 10 (transition mode from LIS to tracking mode). If there are less than four centroids, the next image will repeat state 1. State 10 runs the full frame centroid code then passes the new centroids and the previous centroids to the “Assign IDs” function. This function matches the stars in the current image to the stars in the previous image. Matching stars are given identical ID numbers, IDs for retired stars are dropped, and new IDs are assigned for new stars. The new centroid data and IDs are saved in a global variable called “New Centroids”. If there are four or more new centroids and no error flag from the ID code, the state is set to 2, otherwise the state is re-set to 1. State 2 (tracking mode) passes the centroids and IDs from the previous two images to the tracking code which predicts the location of the stars in the current image. The windowed centroid code uses the predicted centroid locations to find the new centroids in the current image. If the windowed centroid code finds fewer than four stars, the state is reset to one. Otherwise, the state remains at 2 and the new centroids and previous centroids are saved.

4.2. Star ID Algorithm

4.2.1. Star ID Algorithm Motivation and Purpose

The purpose of assigning star IDs is to handle cases in which stars are added or subtracted from the image. The centroid code saves centroid data in an array with 20 elements as follows: $C = [u_1, v_1, u_2, v_2 \ldots u_{10}, v_{10}]$. The first element in the array is the u centroid value of the first star, the second element is the v centroid value of the first star, the third position is the u centroid value of the second star, and the fourth position is the v centroid value of the second star. This pattern continues for up to 10 stars. The centroid code stops saving centroid data if more than 10 stars are found. Because the centroid code always reads from the top of the image to the bottom, as stars are added or subtracted during spacecraft slew the order of the centroids changes. Figure 4-3 below demonstrates this problem.

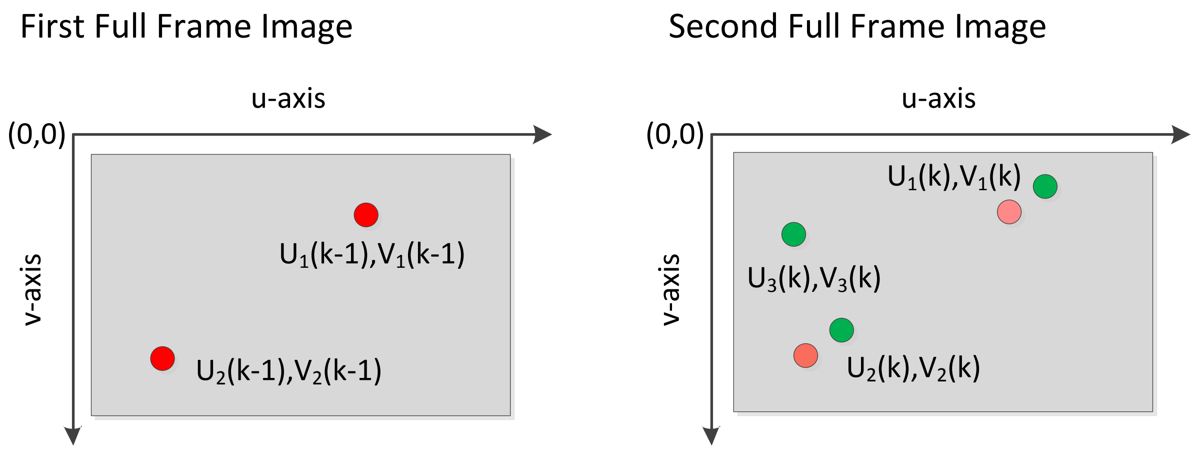


Figure 4-3: Example of Star ID Algorithm

As the star tracker slews between reading out the first and second image, the first and second star move positions and a third star is added. The centroid code will read out the centroid positions for each frame as follows: $C(k-1) = [u_1(k-1), v_1(k-1), u_2(k-1), v_2(k-1)]$ and $C(k) = [u_1(k), v_1(k), u_3(k), v_3(k), u_2(k), v_2(k)]$. The tracking code needs to find the change in position for each star (Δu and Δv) from the first image frame to the second image frame. Because the order of the centroids changes, the tracking code cannot simply subtract the *C(k)* array from the *C(k-1)* array. This thesis develops a simple and robust star ID algorithm based on angular distance of images without a priori information such as a star catalog.

4.2.2. Star ID Algorithm Details

State 1 uses the "Assign First Time Star IDs" algorithm to create star IDs for the first full frame image. This function simply assigns IDs in numerical order. The first centroid is given ID number 1, the second centroid is assigned ID number 2, and so on, for each star the centroid code finds up to a max of 10. In the example shown in Figure 4-3, for the first frame image, star $(u_1(k-1), v_1(k-1))$ is given ID 1 and star $(u_2(k-1), v_2(k-1))$ is given ID 2.

State 10 uses the "Assign IDs" algorithm to assign IDs to the centroids found in the second full frame image. This algorithm must correctly match each star in the current image frame to a star from the previous image. Assuming the spacecraft slews at a slow rate relative to the star tracker frame rate, the stars will only move a small distance between images. Therefore, the Assign IDs algorithm subtracts each star's current location from all the star locations in the previous frame. Whichever star pair has the smallest displacement is considered a match.

For the example shown in Figure 4-3, the star ID for star $(u_1(k), v_1(k))$ in the second full frame image is found by calculating the following displacements: $D_{11} = (u_1(k) - u_1(k-1))^2 + (v_1(k) - v_1(k-1))^2$ and $D_{12} = (u_1(k) - u_2(k-1))^2 + (v_1(k) - v_2(k-1))^2$. The smallest distance represents a match to star $(u_1(k), v_1(k))$. In this example, D_{11} will be smaller than D_{12}, therefore star $(u_1(k), v_1(k))$ matches star $(u_1(k-1), v_1(k-1))$. Star $(u_1(k), v_1(k))$ is assigned ID 1 to match the ID assigned to star $(u_1(k-1), v_1(k-1))$ in the "Assign First Time Star IDs" function explained above. There is no need to take the square root in the distance equation because the ID algorithm does not have to find the actual distance between stars in order to identify matches. Leaving out the square root saves computation time.

In order to account for cases in which stars are added or retired between images, a threshold called "maxDeltaUV" is used to specify the maximum distance allowed between star matches. If the distance between a star in the current frame and star in the previous frame is greater than the threshold, the pair is ruled out as a match. For example, the Assign IDs code will find the distances between star $(u_3(k), v_3(k))$ in the second image and both stars in the first image. If both distances are greater than the threshold, it is assumed that star $(u_3(k), v_3(k))$ was added due to the spacecraft slew and is assigned ID 3.

There are two cases in which the ID algorithm could fail triggering the output of an error flag. The first error occurs if fewer than two star matches are detected because the tracking code needs at least two star pairs in order to estimate slew rate. The second error occurs when one star from the previous frame is matched to more than one star in the new frame. This can happen if stars become clustered together. As shown in Figure 4-2, any errors will trigger the software architecture to return to state 1 and start over.

4.3. Combined Star Tracker Model and Algorithms

The fast centroiding algorithm was combined with the star tracker model into a MATLAB Simulink simulation as shown in Figure 4-4 below.

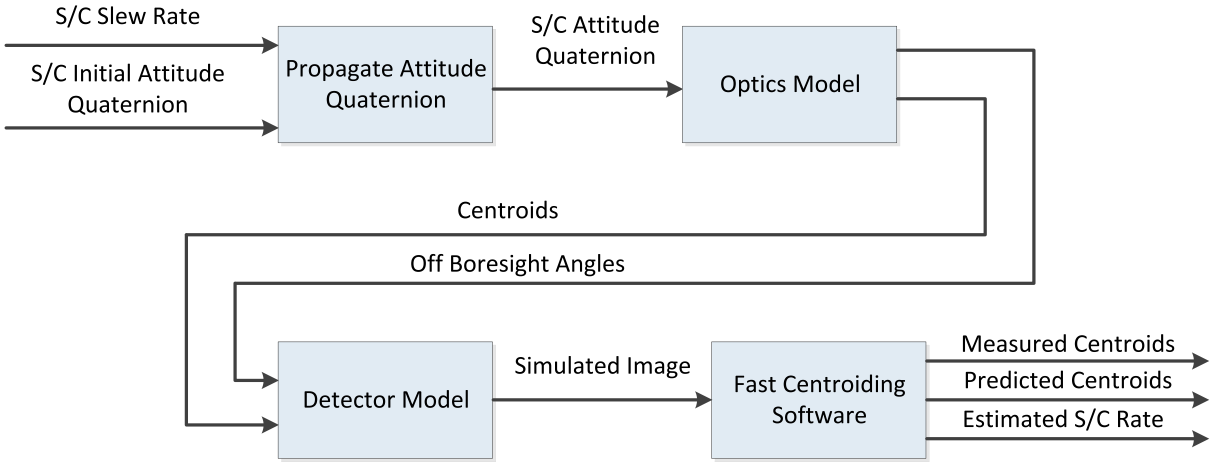


Figure 4-4: Combined Star Tracker Model and Fast Centroiding Algorithm

The simulation initializes a constant slew rate, spacecraft attitude quaternion, and star locations on the detector. The spacecraft attitude is propagated with the slew rate at each simulation time step. The optics model and detector model create a simulated camera image as explained in

Chapter 3. The simulated image is passed to the fast centroiding software to output the measured centroids, predicted centroids, and estimated star tracker slew rate.

4.4. Centroid Accuracy Analysis

The centroiding algorithm accuracy was tested with Monte Carlo runs from the star tracker simulation. For each trial, one star was placed at a random location on the detector. The optics and detector model simulated a camera image which was passed to the fast centroiding code. The fast centroiding algorithm was modified so that only the full frame centroiding algorithm was used to calculate the measured centroid location. A parametric analysis was conducted varying the slew rate from 0 to 0.5 deg/sec. Each trial was run 100 times in order to provide enough data for statistical analysis of the centroid error. Centroid error is defined as the difference between the actual centroid location and the measured centroid location. Two different cases were tested: stellar aberration on and off. A constant spacecraft orbital velocity was used in the optics model to determine the stellar aberration. The results are shown in the tables below.

Table 4-1: Centroid Error with Stellar Aberration Off

Star Tracker Model Centroid Error Analysis (stellar aberration off)						
Slew Rate (deg/sec)			**X Centroid Error (pixels)**		**Y Centroid Error (pixels)**	
Roll	**Pitch**	**Yaw**	**Mean**	**Std Dev**	**Mean**	**Std Dev**
0.0000	0.0000	0.0000	3.3788E-03	3.3943E-02	3.4454E-03	3.1481E-02
0.1000	0.1000	0.1000	-9.7878E-03	6.8563E-02	7.7379E-03	7.7198E-02
0.2000	0.2000	0.2000	-4.8337E-02	1.1536E-01	5.4238E-02	1.2139E-01
0.3000	0.3000	0.3000	-3.7934E-02	1.9361E-01	5.3088E-02	1.8822E-01
0.4000	0.4000	0.4000	-2.9840E-02	2.5794E-01	3.3704E-02	2.5913E-01
0.5000	0.5000	0.5000	n/a	n/a	n/a	n/a

Table 4-2: Centroid Error with Stellar Aberration On

Star Tracker Model Centroid Error Analysis (stellar aberration on)						
Slew Rate (deg/sec)			X Centroid Error (pixels)		Y Centroid Error (pixels)	
Roll	Pitch	Yaw	Mean	Std Dev	Mean	Std Dev
0.0000	0.0000	0.0000	-4.8891E-01	3.0688E-02	1.9131E-01	3.4208E-02
0.1000	0.1000	0.1000	-5.0562E-01	7.1719E-02	2.1617E-01	7.3437E-02
0.2000	0.2000	0.2000	-5.1052E-01	1.3452E-01	2.2836E-01	1.3387E-01
0.3000	0.3000	0.3000	-5.5435E-01	1.9542E-01	2.5367E-01	1.9897E-01
0.4000	0.4000	0.4000	-5.4426E-01	2.6653E-01	2.6683E-01	2.7598E-01
0.5000	0.5000	0.5000	n/a	n/a	n/a	n/a

Table 4-1 and Table 4-2 show the centroid error with and without the stellar aberration respectively. Figure 4-5 below shows the standard deviation of the centroid error with the stellar aberration turned off.

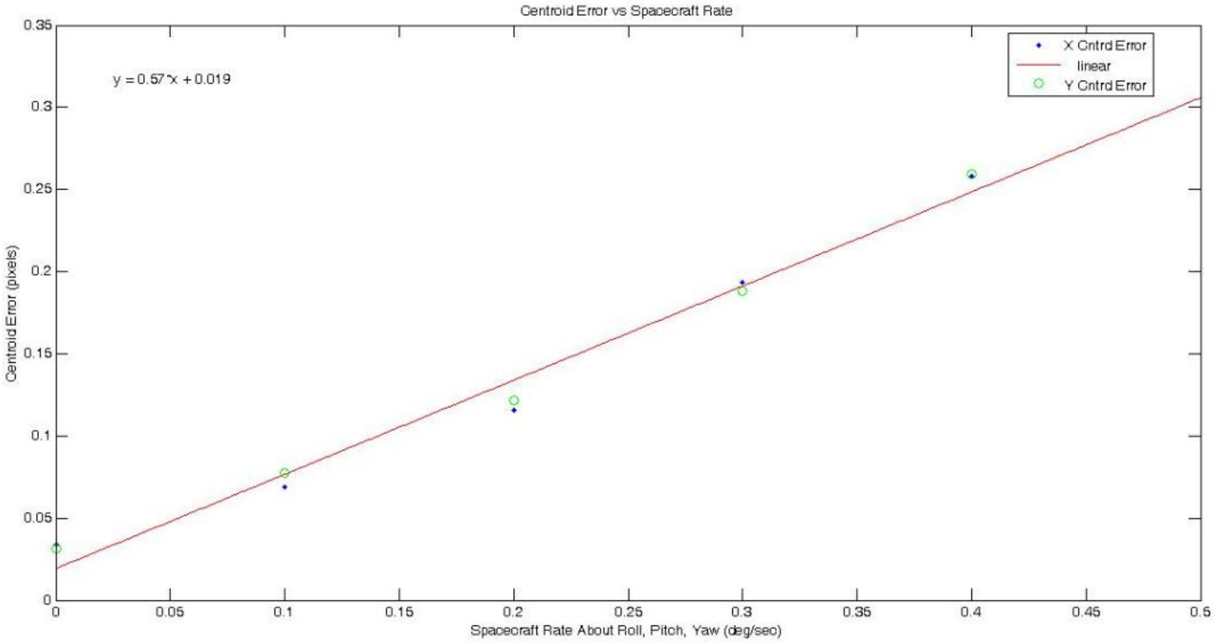


Figure 4-5: Centroid Error vs. Spacecraft Rate with Stellar Aberration Off

The data shows that the centroid error standard deviation increases as rate increases. As the spacecraft slew rate increases, the star smears across the detector. This smear increases the centroid error because of two effects. First, the elongated shape reduces the star's symmetry. Second, as smear increases, the star becomes dimmer reducing the signal to noise ratio. If the brightest pixel drops below the centroiding algorithm signal threshold, the centroiding algorithm will fail to identify a star. This effect is shown in the results for 0.5 deg/sec which are n/a because the centroid code failed to detect the star.

Stellar aberration does not affect the standard deviation as it is nearly the same for both cases. However, stellar aberration does affect the mean centroid error. The mean is nearly zero when the stellar aberration is off and on the order of 0.5 pixels when the stellar aberration is on. This result makes intuitive sense because stellar aberration shifts the star location on the detector in the optics model. This has the effect of introducing a bias in the centroid error. The stellar aberration centroid error is greater in the x direction due to the constant velocity chosen for the simulation. In flight, the spacecraft velocity would change depending on the satellite's orbital location resulting in a varying mean centroid error. The goal of these results was to use a constant velocity representative of flight to provide typical error values.

Chapter 5

5. Air Bearing Testbed

5.1. Purpose and Objectives

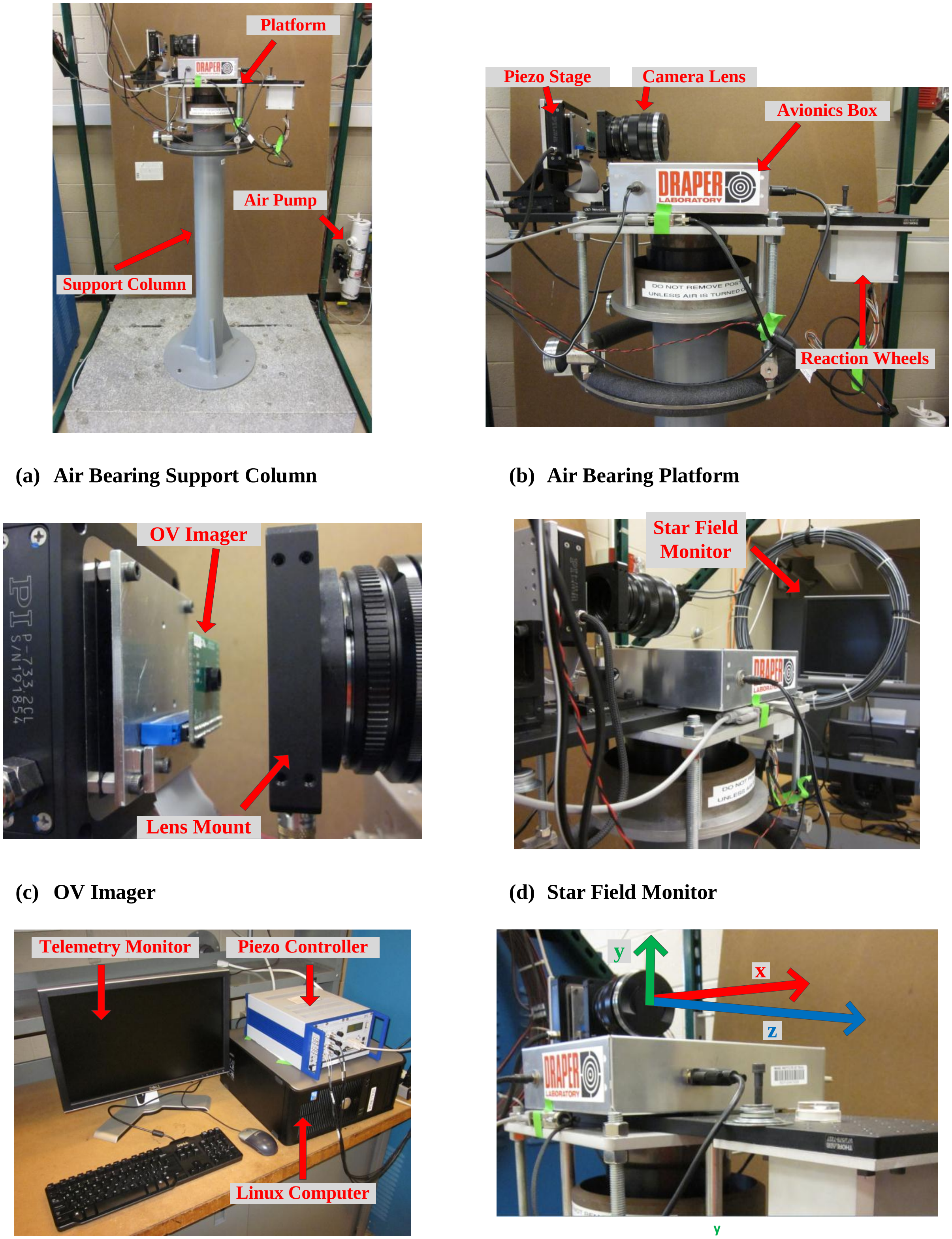
A hardware testbed was developed in order to complete two objectives. The first objective was to measure the critical performance metrics of the fast centroiding algorithm such as processing time and centroid prediction error in a flight-like dynamic slew environment. The second objective was to use a relatively low cost testbed to validate the star tracker model. Once validated, the model can be used in a SWIL testbed in order to develop and analyze further attitude control algorithms before hardware validation. For example, the model can be combined with a high fidelity spacecraft simulation which provides a low cost platform for conducting trade studies, developing flight software via MALAB AutoCoder, and analyzing the optical pointing performance to determine if it can meet scientific objectives.

5.2. Testbed Set-Up

5.2.1. Hardware Architecture

Demonstrating the performance of the fast centroiding algorithm requires the ability to rotate the star tracker in order to displace stars to different locations on the detector. It was determined that a three degree of freedom air bearing could fulfill this requirement. An air bearing is a device often used to test spacecraft attitude determination and control systems. A platform is mounted on a support column. Air is pumped through the column to levitate the platform allowing for 360 degree rotation about the axis running through the support column and approximately 45 degree rotation about the cross axes.

The following hardware was mounted to the top of the air bearing platform: 35mm camera lens, OV 5642 CMOS imager, piezoelectric stage, MAI 200 reaction wheels, and Draper avionics box with DM 355 processor. A Linux computer and piezo controller were connected to the Draper avionics box. A Windows PC and monitor displayed a star field. The piezoelectric stage and reaction wheels were mounted on the platform from previous lab tests. They remained on the testbed in order to keep the platform balanced. The OV 5642 CMOS detector was chosen as the representative star tracker imager due to its low cost and availability. The detector was mounted approximately 85 millimeters from the lens. The lens pointed to a monitor which displayed a star field centered on Alpha Centauri B.   The hardware set-up is shown in Figure 5-1 below. Roll, pitch, and yaw angular rotation are defined as rotation about the x, y, and z axes respectively. The z axis is camera boresight. More information about air bearing operation including pressure settings and platform balancing can be found in [24].



(a) Air Bearing Support Column

(b) Air Bearing Platform

(c) OV Imager

(d) Star Field Monitor

(e) Command and Control

(f) Coordinate Frame

Figure 5-1 (a-f): Air Bearing Testbed Set-Up

The star field was generated from a MATLAB script which processed a star catalogue to select stars magnitude 8 or greater. The MATLAB script projected the stars onto the CMOS detector then back to the screen assuming a focal length of 85 millimeters and a monitor distance of 1.2322 meters from the lens. A bmp image was created from the projected centroid locations such that each star represented a single bright monitor pixel. The bmp file was displayed on the monitor during testbed operation.

There were several electrical connections to pass commands and collect data. The DM 355 sent commands and received images from the OV imager through a ribbon cable. The DM 355 received commands from the Linux computer through a serial cable, sent image data and telemetry to the Linux computer through an Ethernet cable, and communicated with the piezo controller through a serial cable. The piezo controller sent commands to and received telemetry from the piezo stage. The embedded avionics system software was developed by engineers at Draper Laboratory for a previous experiment. Although the piezo stage was not used in this thesis, it had to remain connected because the embedded system initialized assuming the piezo was connected. Figure 5-2 below shows a block diagram of the avionics hardware architecture.

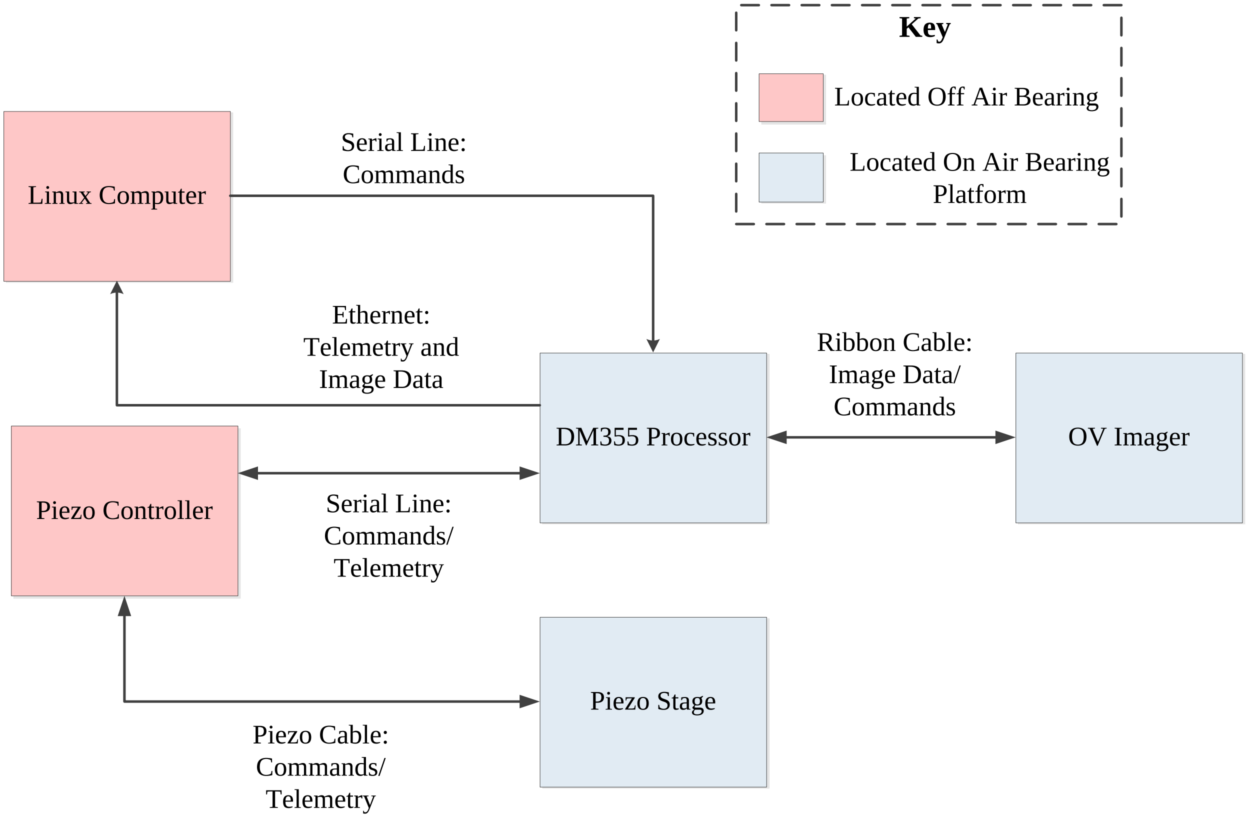

Key
Located Off Air Bearing
Located On Air Bearing Platform
Linux Computer
Serial Line: Commands
Ethernet: Telemetry and Image Data
DM355 Processor
Ribbon Cable: Image Data/ Commands
OV Imager
Piezo Controller
Serial Line: Commands/ Telemetry
Piezo Stage
Piezo Cable: Commands/ Telemetry

Figure 5-2: Avionics Hardware Architecture

5.2.2. Software Architecture

Engineers at Draper Laboratory developed most of the embedded avionics software including the Linux computer command and control, the piezo stage and OV imager application programming interfaces, and the DM355 interfaces. The image processing software architecture including centroiding code, tracking code, and star ID code were developed separately for this thesis. The centroiding code was hand coded in C. The star ID algorithm and tracking algorithm were developed in MATLAB Simulink then autocoded to C-code. The testbed was controlled through a program written in the Python language on the Linux computer while the DM 355 ran U-Boot for its operating system. The image processing software was stored and compiled on the

Linux computer. When the DM 355 booted up, the compiled code was automatically downloaded to the DM 355.

The Python program could initialize the testbed to run in several different modes. The user entered the mode at the Linux terminal during runtime. The file called “cmd_cam.c” read the command entered in the python application and called the corresponding loop command. The code for the loop command was written in the file “csdltest.c”. This file commanded the OV detector on and off and called the proper image processing functions. The two image processing functions were named “cinit” and “cproc”. The code for these functions was maintained in the file “FinePointingLoop.c”. The tracking algorithm and star ID algorithm were initialized in “cinit”. The fast centroiding code was located in the “cproc” function. Figure 5-3 below shows a high level block diagram of the software architecture.

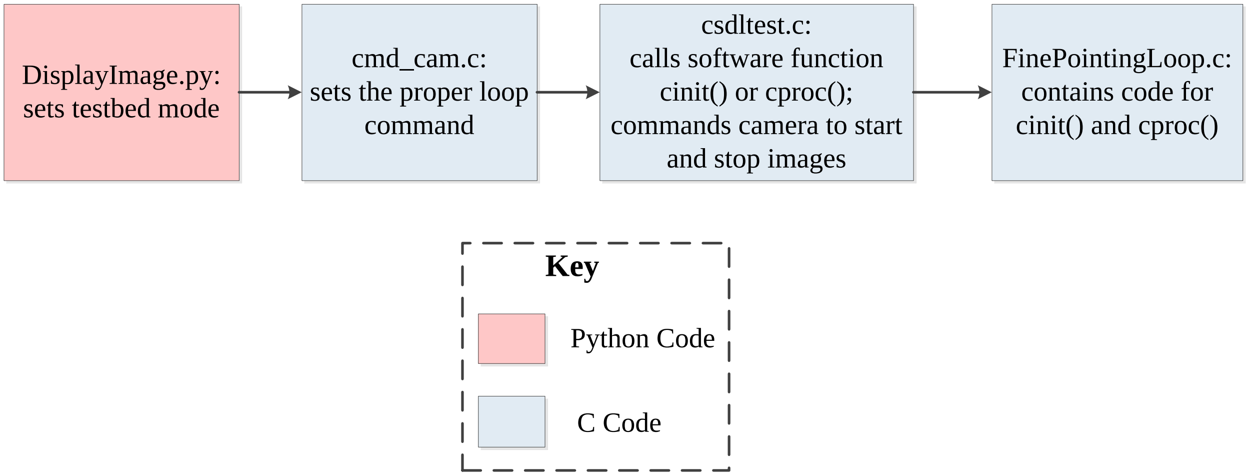

DisplayImage.py:
sets testbed mode
cmd_cam.c:
sets the proper loop command
csdltest.c:
calls software function cinit() or cproc();
commands camera to start and stop images
FinePointingLoop.c:
contains code for cinit() and cproc()
Key
Python Code
C Code

Figure 5-3: Air Bearing High Level Software Architecture

There were several operating modes for the testbed. The testbed could be commanded to take a single windowed image, a series of 80 windowed images, a series of 80 full frame images, infinite windowed images, or infinite full frame images. When the camera was commanded to take infinite images, only an abort command from the python application could stop the camera. The windowed mode images were 256 x 256 pixels. The full frame mode captured full 2560 x 1920 pixel images. The mode most pertinent to this thesis was called “Lost-in-Space (LIS) Loop Forever”. When this mode was called, the camera was set to full frame mode with a 16 Hz integration time and 4 Hz frame rate. After initialization, the camera began taking images which were passed to the fast centroiding code. The fast centroiding algorithm outputs were displayed in the Python application at the Linux terminal. The system ran in a continuous loop with images passed to the fast centroiding code and data output to the monitor at 4Hz. An abort command in the Python application would break the loop.

5.3. Testbed Data Collection Procedures

The purpose of using the air bearing was to mimic spacecraft slews. In theory, this could be accomplished by floating the platform then imparting a small force causing it to slowly spin. When the camera field of view passes over the star field monitor, stars would slew across the detector much like a star tracker operating in space. However, slew tests revealed that the cables leading to the avionics box created enough tension on the platform to create a restoring force about an equilibrium point much like a pendulum. Pushing the platform to impart a force caused it to oscillate about equilibrium. Nevertheless, the air bearing testbed proved effective.

Air bearing data was collected as follows. The air bearing was turned on to float the platform. The platform was balanced so that the camera pointed level at the star field monitor. Next, the avionics were commanded through the Linux computer to begin taking images in the LIS Loop Forever mode. The air bearing was given a light push which caused it to oscillate about the equilibrium point. The avionics box fed the following telemetry at each time step to the computer monitor: frame number, state, centroid code processing time, tracking code processing time, total time to run the fast centroiding code (tracking algorithm plus centroid algorithm), number of predicted centroids, the x and y locations of each predicted centroid (max 10), the number of centroids output from the centroid code, the x and y locations of each measured centroid from the centroid code (max 10), and the tracking code estimated roll, estimated pitch, and estimated yaw.

Deciding when to turn off the testbed and stop data collection was a little more complicated because only about 120 time steps of data could be collected at the Linux command window before the data overwrote itself. After the initial push, the air bearing slewed too fast to capture centroid data. The telemetry read to the screen showing that the software was stuck in state 1 until the air bearing began to slow down. Once the air bearing slowed down, centroid data was captured and the software entered state 2. When the telemetry began showing state 2, the frame number was noted. After approximately 100 more frames were read out, the software was aborted to ensure data was not overwritten.

Several tests were conducted varying the centroid code ROI size. The ROI size in the full frame centroid code and the window size in the windowed centroid code were always set to the same value. For example, at test with an ROI of 200 means that the full frame centroid code used an ROI size of 200 x 200 pixels and the windowed centroid code used a window size of 200 x 200 pixels. An additional set of tests were conducted with the air bearing turned off. These tests were used to analyze the centroid code processing time and centroid location prediction error as a function of ROI size.

5.4. Testbed Data Analysis

5.4.1. Analysis: Centroid Processing Time

The first goal of the testbed was to show that the fast centroiding algorithm reduces the computation time compared to the full frame centroiding algorithm. Recall the fast centroiding algorithm is comprised of the centroid tracking algorithm and windowed centroiding algorithm. Table 5-1 below shows the full frame and windowed centroiding algorithm processing times for various ROIs with the air bearing turned off. The camera was aligned with the star field monitor such that five stars landed on the detector for each trial. It is important to note that the centroiding algorithm time will increase as more stars land on the detector. The testbed data showed that the centroid tracking code for all cases ran in 2 milliseconds.

Table 5-1: Centroid Processing Time

Centroid Time		
ROI (pixels)	**Full Frame (milliseconds)**	**Windowed (milliseconds)**
200	211	70
150	188	40
100	111	18
80	171	12
50	103	5
40	169	3

The windowed centroiding code is approximately 10 times faster than the full frame centroiding code for an ROI of about 80 to 100. Smaller ROIs have even lower processing times. The tracking algorithm processing time of 2 milliseconds is negligible compared to the centroiding code processing time. The goal is to be able to run the fine optical control loop with a camera operating at high rate. Using the ExoplanetSat fine control loop as a baseline, the centroiding code must run faster than 12 Hz or 83.3 milliseconds. Even with a large ROI (100 to

200 pixels), the fast centroiding algorithm can run faster than 83.3 milliseconds. Lower processing times result in shorter time delays and better pointing performance.

5.4.2. Analysis: Air Bearing Motion

In order to conduct further analysis, it is important to understand the air bearing motion and how it affects the tracking code prediction error. As previously explained, when force is exerted on the platform, the air bearing oscillates about equilibrium like a pendulum. Plotting the x and y centroid position output from the centroid code over time can reveal this motion. Figure 5-4 below plots the actual and predicted location of the first centroid output from the centroid algorithm for the ROI 200 test case. The prediction error is also plotted. The actual centroid location is defined as the centroid location measured from the centroiding code. The prediction error is defined as the tracking algorithm predicted centroid location minus the measured centroid location.

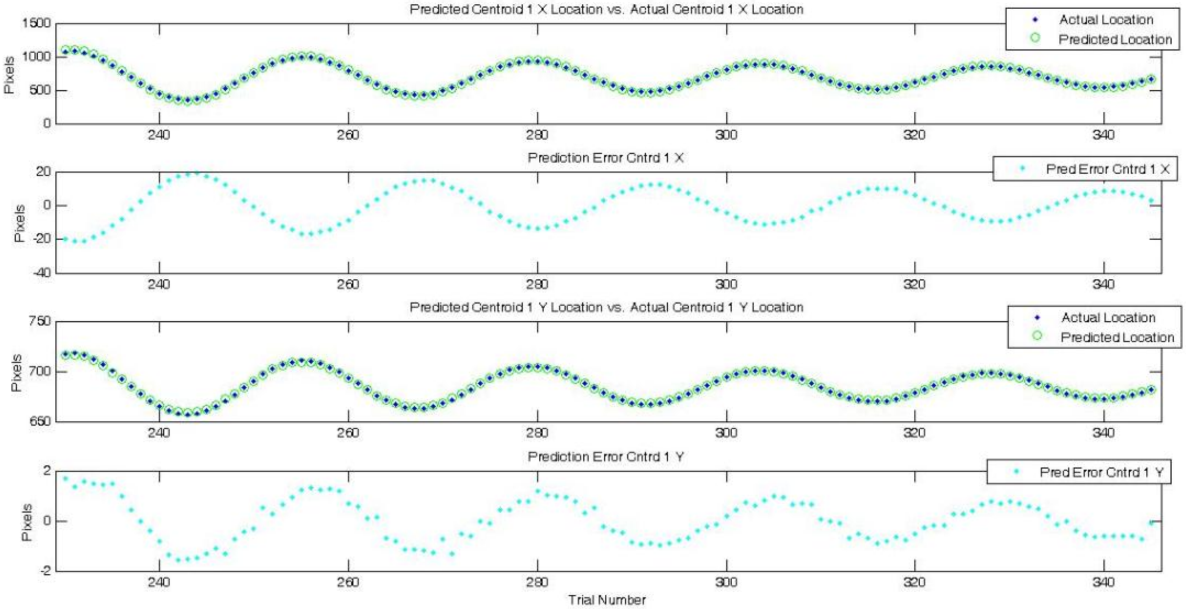


Figure 5-4: Position of First Centroid over Time for ROI 200 Case

A smaller prediction error is best for optical pointing. A more accurate prediction algorithm allows for the use of a smaller window size in the windowed centroiding code which increases the speed of the centroiding algorithm. The damped sinusoidal plots of the x and y centroid positions confirm that the air bearing is oscillating with damped motion about the equilibrium point. The prediction error is also a damped sinusoid in phase with the air bearing motion. The prediction error is near zero when the air bearing is passing through equilibrium and at a maximum when the air bearing is at a maximum. The star oscillates more in the x position than the y position because the air bearing oscillated more about the pitch axis than the roll axis. The air bearing was more constrained about the roll axis due to the wire placement restricting movement. This motion results in greater centroid prediction error in the x direction than the y direction.

To better understand the air bearing motion and why the prediction error is at a minimum at equilibrium, Figure 5-5 and Figure 5-6 plot the prediction error in the x direction of the first centroid vs. the estimated angular velocity and angular acceleration over time. The angular velocity estimate at each time step was output from the tracking code during data collection. Angular acceleration was found by post processing the angular velocity data using the centered difference approximation [16].

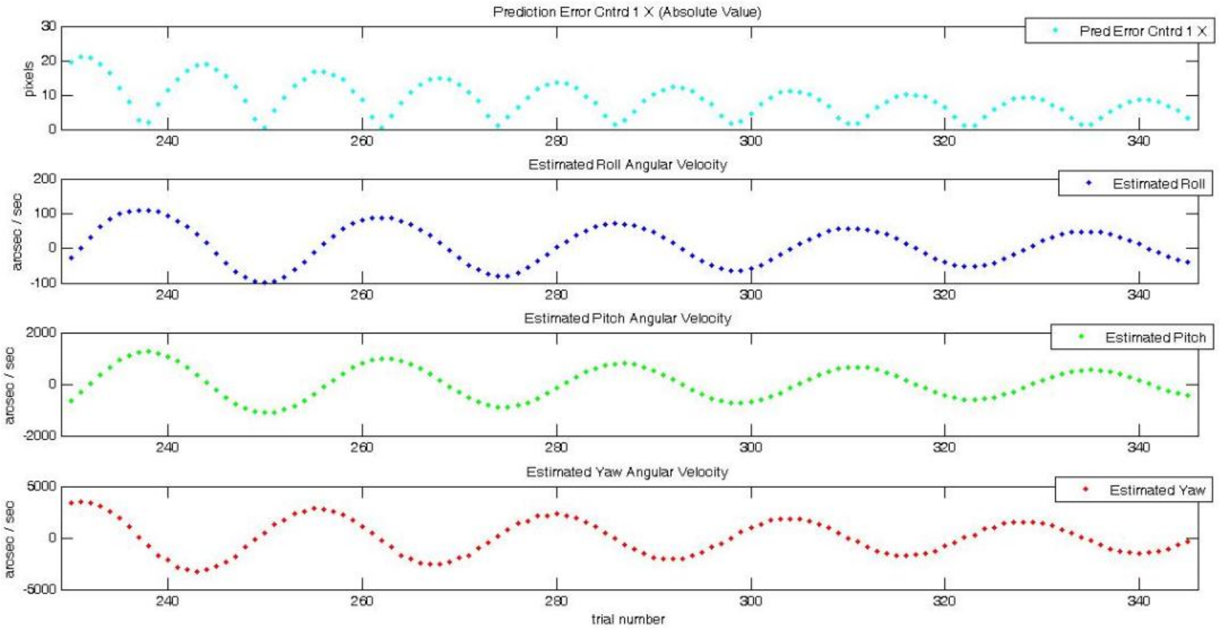


Figure 5-5: Centroid 1 X Prediction Error vs. Estimated Angular Velocity

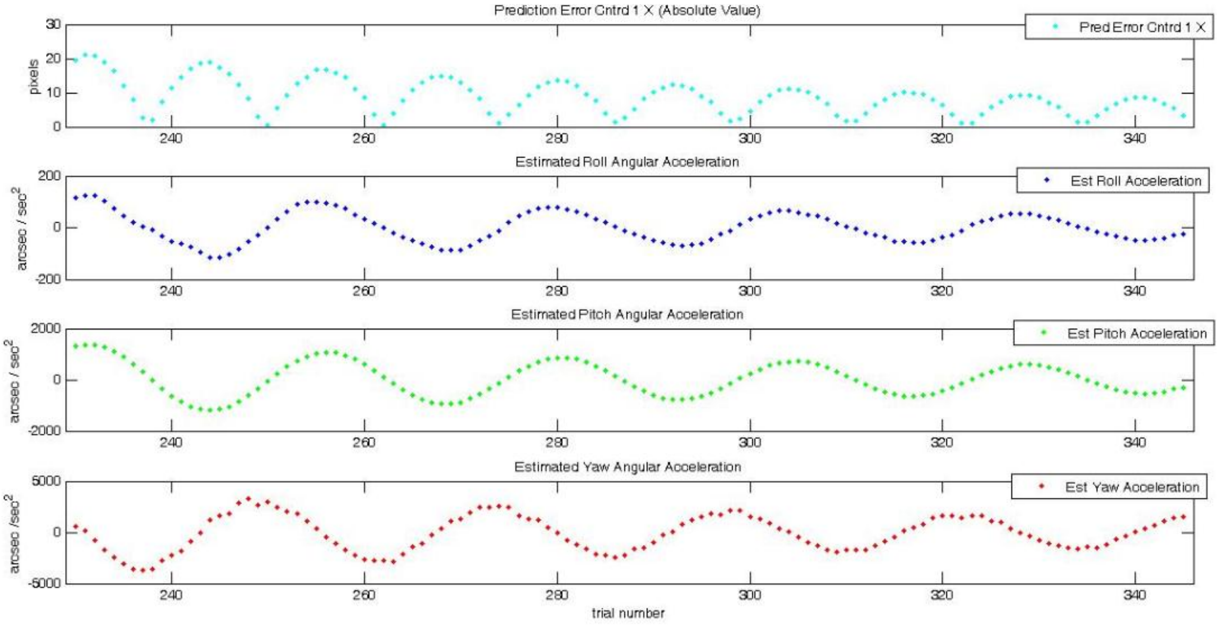


Figure 5-6: Centroid 1 X Prediction Error vs. Estimated Angular Acceleration

Figure 5-5 and Figure 5-6 show the sinusoidal motion of the estimated roll, pitch, and yaw velocity and acceleration. In both figures, yaw is approximately 90 deg out of phase with roll and pitch. Whenever roll and pitch velocities are at a local maximum, the prediction error is at a local minimum. When roll and pitch velocities are at a local minimum, the prediction error is at a local maximum.

Initially this result does not seem to make sense when considering the results of the tracking algorithm analysis in section 2.3.3 and centroid algorithm analysis conducted in section 4.4. Recall that centroid error causes errors in roll, pitch, and yaw estimation which lead to prediction error. Also, recall that centroid error increases as slew velocity increases. Therefore, as slew velocity increases, centroid error should increase leading to an increase in attitude rate estimation error and centroid prediction error. Prediction error should be smallest when slew velocity is smallest. However, the tracking code assumes constant angular velocity when estimating future star locations. In the air bearing testbed, this is not the case.

The air bearing pendulum motion means that the camera velocity is constantly changing. When the camera passes through equilibrium, velocity is at a maximum but acceleration should be near zero. At the extreme points in the pendulum motion, the velocity is zero but acceleration is at a maximum as the camera starts swinging back toward equilibrium. Therefore, the prediction error should be at a minimum when the camera is passing through equilibrium causing a maximum velocity but minimum acceleration. This conclusion is supported by Figure 5-5 and Figure 5-6. When prediction error is at a minimum (passing thru equilibrium), the roll and pitch acceleration are at a minimum. The yaw 90 degree phase difference is not alarming. Recall that the yaw angle is the rotation about camera boresight. The tracking code analysis revealed that yaw estimation error is greater than roll and pitch. Because the dominant air bearing motion is about pitch, further analysis and star tracker model validation will only consider the pitch axis.

5.4.3. Analysis: Centroid Prediction Error

The analysis of the air bearing motion gives insight into how to analyze the data to characterize the centroid prediction error. In order to demonstrate the effect of star tracker slew on prediction error, the angular acceleration must be minimized. For each trial, the mean and standard deviation of the pitch angular acceleration was calculated. From this data a threshold value was calculated by taking the mean angular acceleration minus 1.5σ. Next the data was sifted to find the prediction error and pitch angular velocity at time steps where the angular acceleration was less than or equal to the threshold. Therefore, the data is limited to when the camera is slewing near equilibrium. 1.5σ was chosen because a larger threshold value eliminates too many data points for statistical analysis. Figure 5-7 below shows a plot of prediction error vs. estimated pitch angular velocity when the pitch angular acceleration is below the threshold.

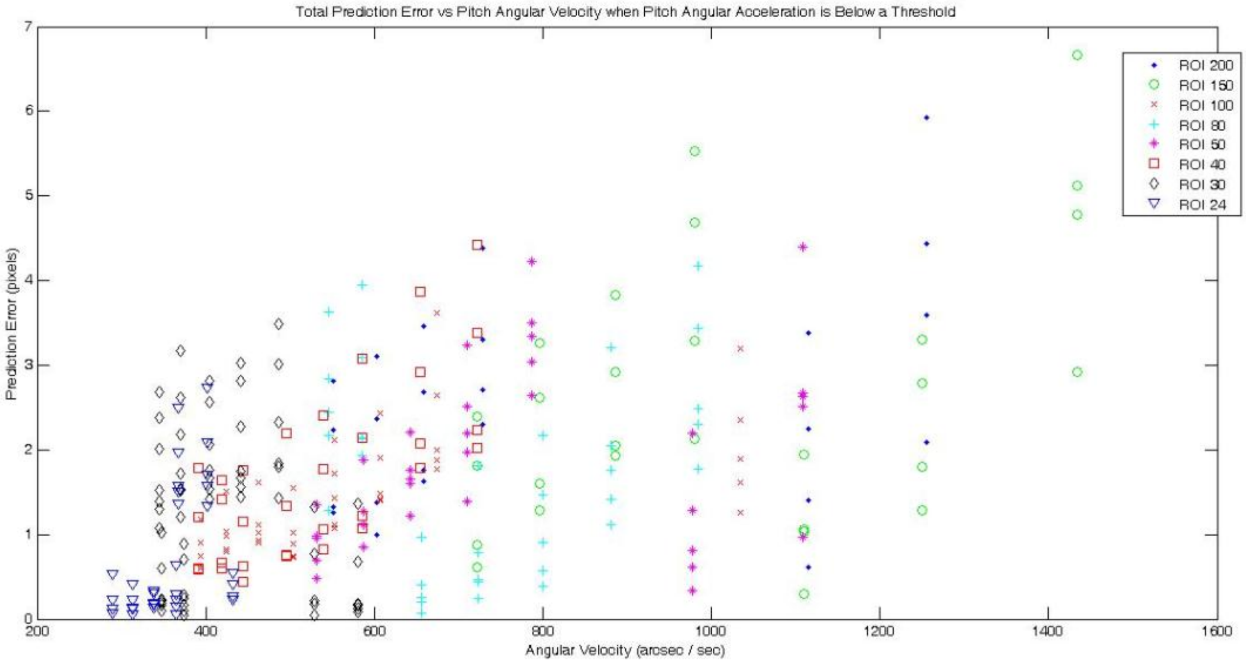


Figure 5-7: Centroid Prediction Error vs. Pitch Angular Velocity when Pitch Angular Acceleration is Below a Threshold

The plot of prediction error vs. pitch angular velocity shows that, in general, as angular velocity increases the prediction error increases. However, several noisy data points make it difficult to really characterize the trend. Notice the data points are clustered within specific velocity ranges and data for larger ROIs appear at higher velocities while data for smaller ROIs appear at lower velocities. As velocity increases, prediction error increases and a larger ROI size is required to capture the centroids. The air bearing had to slow down enough for the software to accurately track centroids and remain in state 2. This means that larger ROIs were able to track centroids when the air bearing was slewing faster. Smaller ROIs could only track centroids at slower angular velocities. Recall only 100 data points total were collected for each trial. For larger ROIs, data collection stops before the air bearing slows down enough to collect data at slower velocities. Limiting the data to the equilibrium point results in most of the data clustered at the same velocity bands.

In an attempt to filter out noise, the data was binned at different velocity groups. The mean and standard deviation of the prediction error was calculated at each bin. Figure 5-8 marks the bins and Figure 5-9 plots the mean, mean plus standard deviation, and mean minus standard deviation prediction error at each bin.

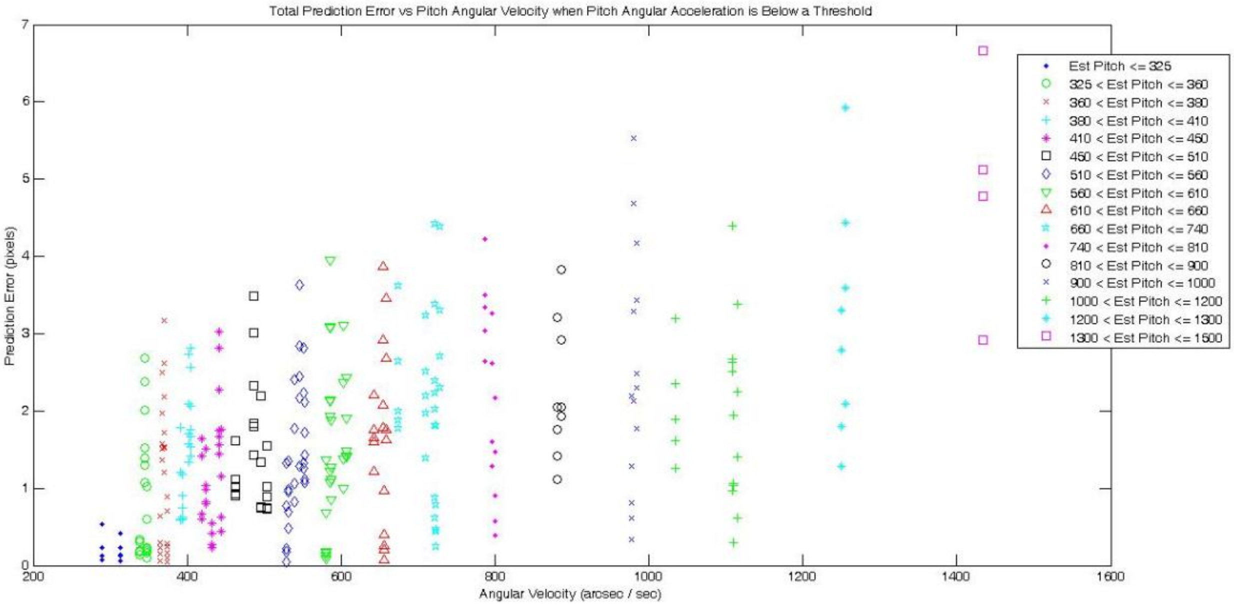


Figure 5-8: Angular Velocity Bins

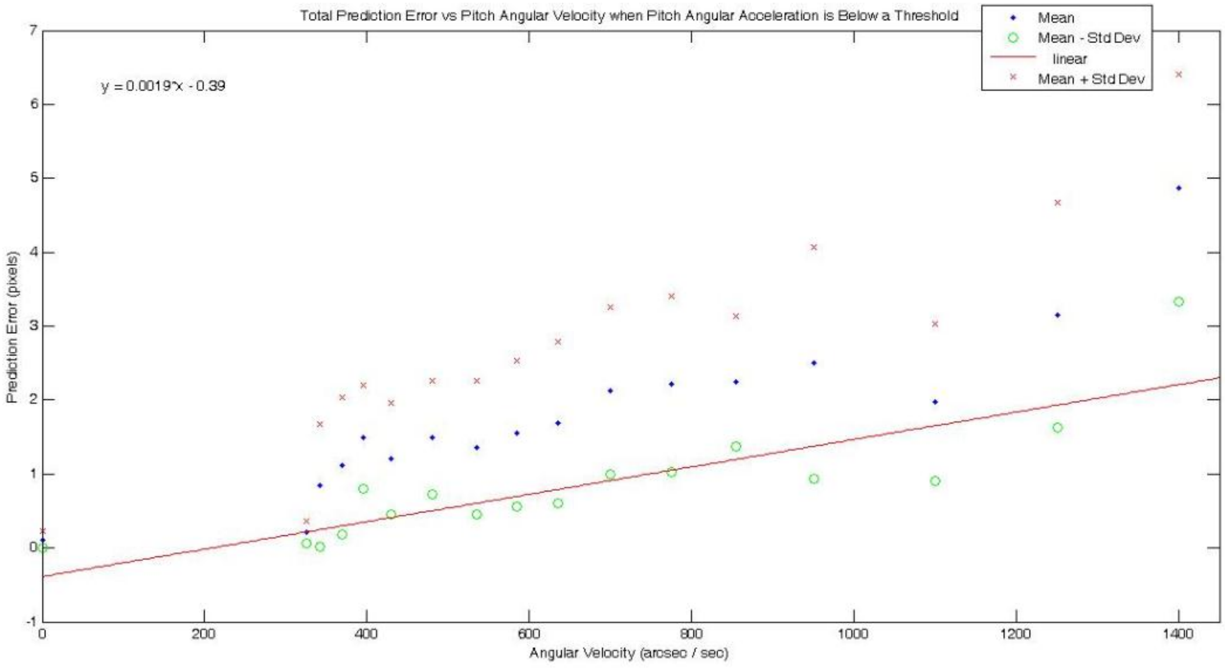


Figure 5-9: Data Statistics for each Angular Velocity Bin with a Linear Best Fit Line through the Mean – Standard Deviation

A linear best fit line running through the mean minus standard deviation shows that as angular velocity increases prediction error increases by approximately 0.0019 pixels / (arcsec/sec). A best fit line through the mean has a slope of 0.0028 pixels / (arcsec/sec). The mean plus standard deviation slope is 0.0038 pixels / (arcsec/sec). This data can be used to validate the imager simulation by comparing the air bearing prediction error trend to the prediction error found in simulation.

5.5. Star Tracker Model Validation

The simulation output can be compared to the air bearing testbed data in order to produce a preliminary validation of the star tracker simulation. The first step in validating the air bearing data is to adjust the simulation parameters so that the OV air bearing images match the simulation images. The overall star shape and detector noise need to approximately match. An air bearing image from the stationary case was compared to an image generated in the simulation. The centroid full width at half maximum (FWHM) and star magnitude were adjusted in the simulation until the star size and pixel values were approximately equal. The dark current and PRNU noise were also increased until the simulated centroid prediction error at zero slew velocity approximately matched the air bearing centroid prediction error at zero velocity. The simulated and actual centroid prediction error at zero slew velocity was about 0.1 pixels.

A Monte Carlo parametric analysis was conducted varying the pitch slew rate because the dominant motion in the air bearing testbed was about pitch. For each slew rate, 20 runs were conducted with each run simulating 2 seconds. The mean, mean plus standard deviation, and mean minus standard deviation prediction error were plotted and compared to Figure 5-9. The mean plus standard deviation best matched the air bearing mean minus standard deviation results. Figure 5-10 below plots the simulation results of the mean plus standard deviation prediction error.

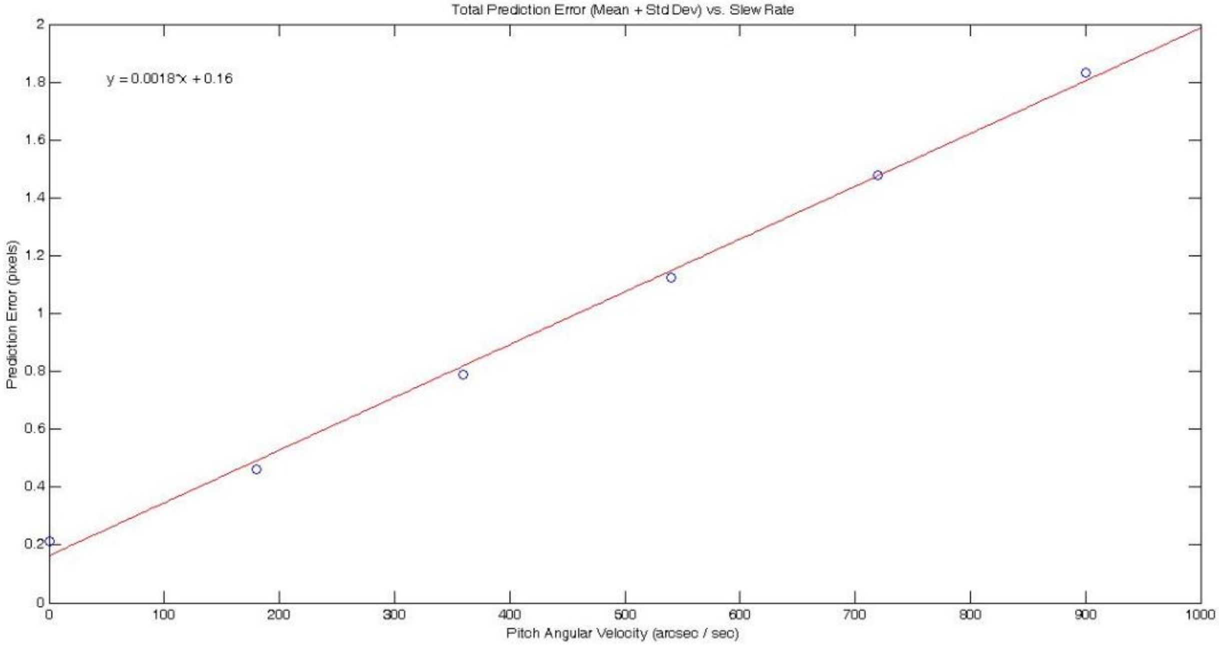


Figure 5-10: Mean plus Standard Deviation Prediction Error vs. Slew Rate with Best Fit Line

Figure 5-10 shows that as angular velocity increases the prediction error mean plus standard deviation increases at a rate of 0.0018 pixels / (arcsec/sec). Recall the air bearing mean minus standard deviation results showed a slope of 0.0019 pixels / (arcsec/sec). There is only about a 5% error between the two results. These numbers indicate that the air bearing best case slope result closely matches the simulation worst case slope result. This observation partially validates the simulation. It makes sense to match the simulation worst case results to the air bearing best case results because the star tracker model is theoretical. The model does not include many of the second and third order optical effects of a camera lens. Also, the hardware optical and imager misalignments can decrease the accuracy of the tracking code, and the testbed star light is not collimated. The star light is collimated in space and is modeled as such in the star tracker model.

5.6. Validation Future Work

In order to fully validate the star tracker model, tests on the flight hardware need to be conducted. The model noise parameters were chosen from a modern CMOS detector spec sheet. Once the flight imager is chosen the noise parameters from the simulation should be updated with the spec sheet values for the chosen imager. Hardware testing should then be used to tune the noise parameters. Dark frames can be compared to simulation dark frames to validate parameters such as dark current. Images of a collimated light source can be used to validate the centroid FWHM and pixel photon response. If the simulation results still do not match hardware data even after the simulation parameters have been tuned, additional second and third order camera effects may need to be modeled. However, this may be difficult and time consuming. In order to match the air bearing data with the simulation data, the noise parameters were simply increased until the results for the zero velocity case were approximately equal. When validating the star tracker model with the flight hardware, rather than increasing the complexity of the model, it may be best to simply increase the noise parameters until the simulation data matches the flight hardware data.

Chapter 6

6. Monte Carlo Analysis

6.1. Motivation and Background

After fully validating the star tracker model, Monte Carlo analysis should be performed. In this thesis Monte Carlo analysis is defined as running a hundred or more simulations with each simulation subject to random detector noise. Data analysis characterizes the fast centroiding algorithm performance. For example, the average and standard deviation centroid error, estimated angular velocity error, and centroid prediction error can be calculated. The analysis can be used to determine if the centroid error is small enough to meet system pointing requirements.

Although MATLAB Simulink is useful for developing the star tracker imager model and software, there are several limitations to using Simulink for Monte Carlo analysis. One limitation to the star tracker simulation is it takes significant computation time to run Monte Carlo simulations even on a modern laptop computer. Another limitation is MATLAB code cannot be used on most embedded systems. The development of space avionics and software systems often requires a processor-in-the-loop (PIL) testbed. The purpose of this type of test is to analyze the algorithm performance on a real-time operating system similar to what would be used in flight. A PIL testbed could be developed for the star tracker simulation. The star tracker simulation could run on a Linux computer passing simulated images to an embedded processor running the fast centroiding algorithm. Before developing a PIL testbed, it is often productive to first create a software-in-the-loop (SWIL) testbed in which the software runs on a real-time operating system written in the programming language used for flight.

MATLAB AutoCoder provides the capability to autocode MATLAB Simulink simulations to C [25]. This tool may provide a solution to the Simulink Monte Carlo issues described above. However, further explanation of this tool is required.

6.2. Matlab AutoCoder

The MATLAB AutoCoder tool offers several advantages. Translating ADCS algorithms from Simulink to C can be tedious, expensive, and can result in human error. Rapidly autocoding to C allows engineers to quickly transfer the code from the MATLAB test environment to the embedded system environment for flight testing. Another advantage is the simulation runs faster in C than in MATLAB allowing for quicker Monte Carlo analysis.

There are some disadvantages of autocode. First, there is no clear process for passing inputs and reading outputs from the autocoded simulation. Second, the code is less efficient and requires more memory storage than coding by hand. Therefore, the autocode may not be useful for programming all of the flight software including the lower level operating system tasks.

Despite the disadvantages, autocode has several important uses. The autocode is convenient for rapidly deploying code for testbed use. It is useful as a tool for model based design. A control systems engineer can rapidly deploy software modules. A qualified software engineer or avionics expert can fit the individual modules into the overall software architecture.

It was determined that the MATLAB AutoCoder should be used to conduct the Monte Carlo analysis for two main reasons. First, the C-code runs faster than the Simulink simulations allowing for faster Monte Carlo analysis. Second, the flight software will most likely be programmed in C. Therefore a simple SWIL testbed can be created by running the simulation in C on a Windows 7 PC. Engineers at Draper Laboratory created a set of tools for producing autocode which can easily read input variables from MATLAB and output simulation results. These tools were used to autocode the star tracker simulation so that a software architecture could be easily created for performing Monte Carlo analysis.

6.3. Draper Autocode Tools

6.3.1. Tunable Parameters

To understand the Draper autocode tools, it is important to be familiar with the concept of “tunable parameters”. The concept of tunable parameters is used in Simulink to indicate to the AutoCoder which parameters should be autocoded as variables and which parameters should be hard coded as values. If a parameter is not specified as tunable, than the autocode will hard code the value in the C-code which was used in the last run of the simulation. For example, if the dark current parameter is set to 300 in the simulation, the autocode will hard code 300 in place for the dark current variable in the C-code. If dark current is specified as tunable, then it will be passed as an input variable to the C-code.

It is important to note that not all variables are tunable. For example, any variables that specify the simulation rate are non-tunable. One standard when writing spacecraft flight code is to always initialize the size of an array to a constant. Dynamically changing an array size during software execution is generally not acceptable for flight code because it alters the software execution speed and memory storage. Therefore, all the variables in the Simulink code were specified as having static size allocation. So, any variable that changes the array size of another variable is also non-tunable.

6.3.2. Monte Carlo Set-Up

The first step in running Draper's autocode tools is to place Draper's specially developed output blocks in the Simulink model. These blocks tell the autocode which variables should be output from the simulation. The second step is to specify which simulation variables are tunable. Next, the MATLAB AutoCoder generates the C-files. Draper's tools automatically move the files into Microsoft Visual Studio which compiles the code and creates and executable file. The executable file can be run from MATLAB.

The executable file is run from MATLAB to produce Monte Carlo simulations. Figure 6-1 below shows a bock diagram of the Monte Carlo set-up in MATLAB.

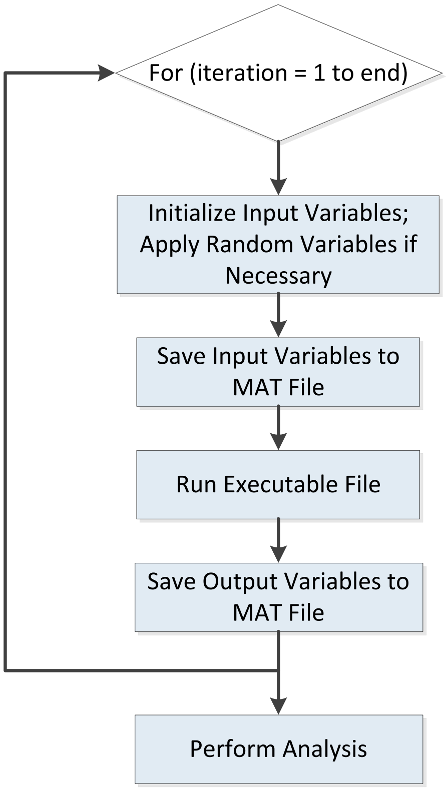

For (iteration = 1 to end)
Initialize Input Variables; Apply Random Variables if Necessary
Save Input Variables to MAT File
Run Executable File
Save Output Variables to MAT File
Perform Analysis

Figure 6-1: Monte Carlo Autocode Set-Up in MATLAB

The Monte Carlo set-up is written as a loop in MATLAB. The loop first initializes the input variables. Often the input variables are initialized with random variables. A nominal input value can be chosen then a Gaussian random variable added to produce noise. Once all the inputs are assigned, they are saved to a MAT file. The simulation executable reads the input variables from the MAT file and runs for a specified simulation time. The executable saves the output variables to the MAT file. This process can be performed for hundreds of iterations. Each time the simulation runs, new random inputs are specified. After the last iteration, analysis can be performed on the results.

6.4. Autocode Validation

The star tracker model was autocoded using Draper's tools and the Monte Carlo framework was written and tested so that future validation of the star tracker model and flight code can be performed. This framework can also be adopted for future research on other spacecraft simulations. In order to validate the framework, the Simulink model and executable file were run with the same inputs. Three different trials were tested varying the spacecraft rate and simulation duration. The outputs were compared to verify that the Simulink model and autocode produce the same results. Table 6-1 below shows the error between the Simulink and autocode output. Error is defined as Simulink output minus autocode output. The table also compares the Simulink and autocode run times.

Table 6-1: Autocode Validation

Autocode Validation						
Trial Number	**1**		**2**		**3**	
Spacecraft Rate (deg / sec) [Roll, Pitch, Yaw]	[0, 0.2, 0]		[0, 0, 0]		[0, 0, 0]	
Simulation Duration (seconds)	2.00		2.00		15.00	
Simulink Run Time (seconds)	255.02		258.17		1860.15	
Autocode Run Time (seconds)	52.55		52.88		382.33	
Variable	**Mean Error**	**Std Dev Error**	**Mean Error**	**Std Dev Error**	**Mean Error**	**Std Dev Error**
State	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Num of Cntrds	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Cntrd Locations (pixels)	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Pred Num Cntrds	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Pred Cntrds Locations (pixels)	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Est Roll (arc sec / sec)	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Est Pitch (arc sec / sec)	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Est Yaw (arc sec / sec)	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

The results show that there is zero error between the Simulink and autocode outputs for all trials. Therefore, the Simulink model and autocode produce the exact same results. The autocode runs approximately 4.8 times faster than the Simulink model. These results demonstrate the advantage of using the MATLAB AutoCoder feature. The autocode produces the same results with shorter processing time allowing for faster Monte Carlo analysis.

6.5. Monte Carlo Analysis and Validation

In order to demonstrate the Monte Carlo framework described above, the autocoded star tracker model executable file was placed in the Monte Carlo framework and run with the same parameters as the air bearing validation. The mean plus standard deviation prediction error was calculated at several different spacecraft slew rates. The simulation run time was set to two seconds to ensure stars did not fall off the detector and twenty runs were conducted for each slew rate. The spacecraft slew rate was declared tunable allowing it to be changed each run.

It is necessary to understand the implications of the random number generators used in the star tracker model. Recall, each noise source in the imager model uses random numbers to simulate the Gaussian noise by using the randn() command. Each noise source essentially has a dedicated random number generator. The star tracker simulation was written in Simulink with a MATLAB S-Function. When random numbers are generated in an S-Function with the randn() command, the randn() output is different for each image. In order to autocode the simulation, the MATLAB S-Function had to be re-written in a CodeGen block. When the randn() command is used in a CodeGen block, the random numbers are initialized to a constant and do not change each time a new image is simulated. So, random numbers had to be passed as inputs to the CodeGen block from the Simulink random number generator blocks with each block initialized to a different seed.   Because the OV imager size is 2560 x 1920 pixels, each seed for each noise source is a 2560 x 1920 matrix. This requires memory for 6 2560 x 1920 matrices. Running the simulation in Simulink resulted in an out of memory error. This error was resolved by reducing the size of the OV imager in the Simulink model to 1500 x 1500 pixels.

The random number generator seeds were autocoded as tunable parameters in order to perform Monte Carlo analysis. The Monte Carlo framework uses random numbers to change the seeds each time the executable file is run. Therefore, the autocode produces different results each time the executable file is run. The OV imager noise parameters were not changed from the Simulink model validation completed in section 5.5. Because the image size is smaller in the autocoded version than the section 5.5 validation, the star positions were fitted proportionally to the new 1500 by 1500 detector size keeping the same relative positions. Figure 6-2 below plots the results with a best fit line.

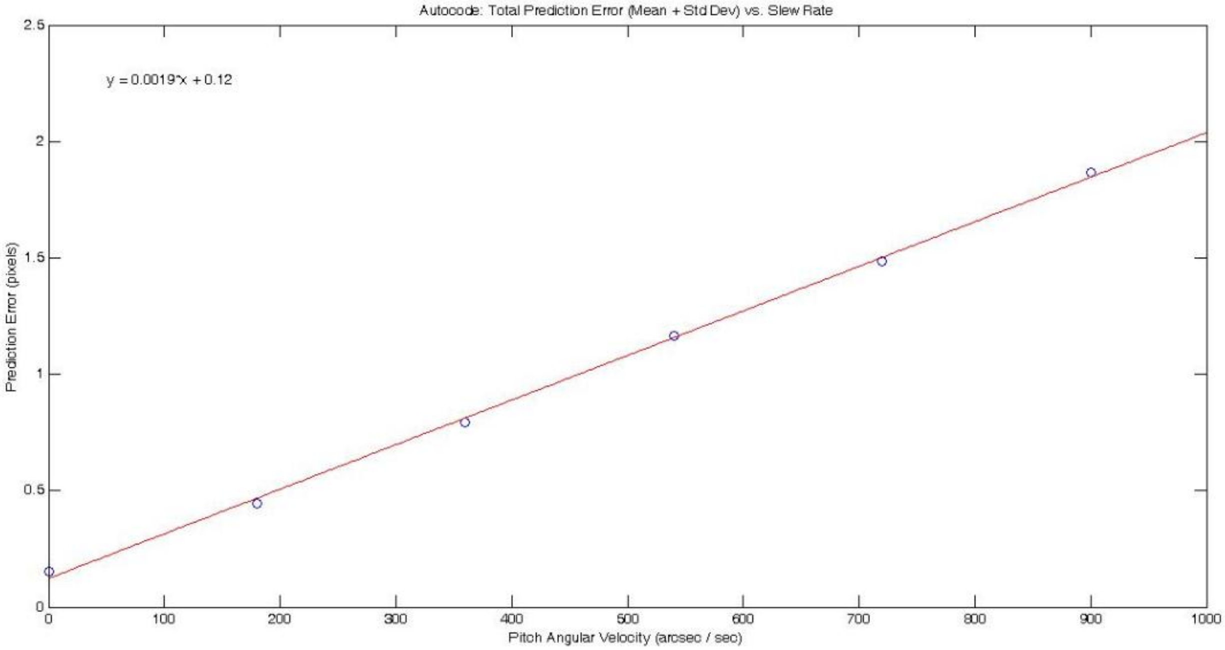


Figure 6-2: Autocode Monte Carlo Results

The prediction error slope increases at a rate of 0.0019 pixels / (arcsec/sec). Recall the Simulink model results show a slope of 0.0018 pixels / (arcsec/sec). There is only a 5 percent difference between the autocode and Simulink Monte Carlo results. Figure 6-3 below shows a direct comparison between the autocode and Simulink Monte Carlo results.

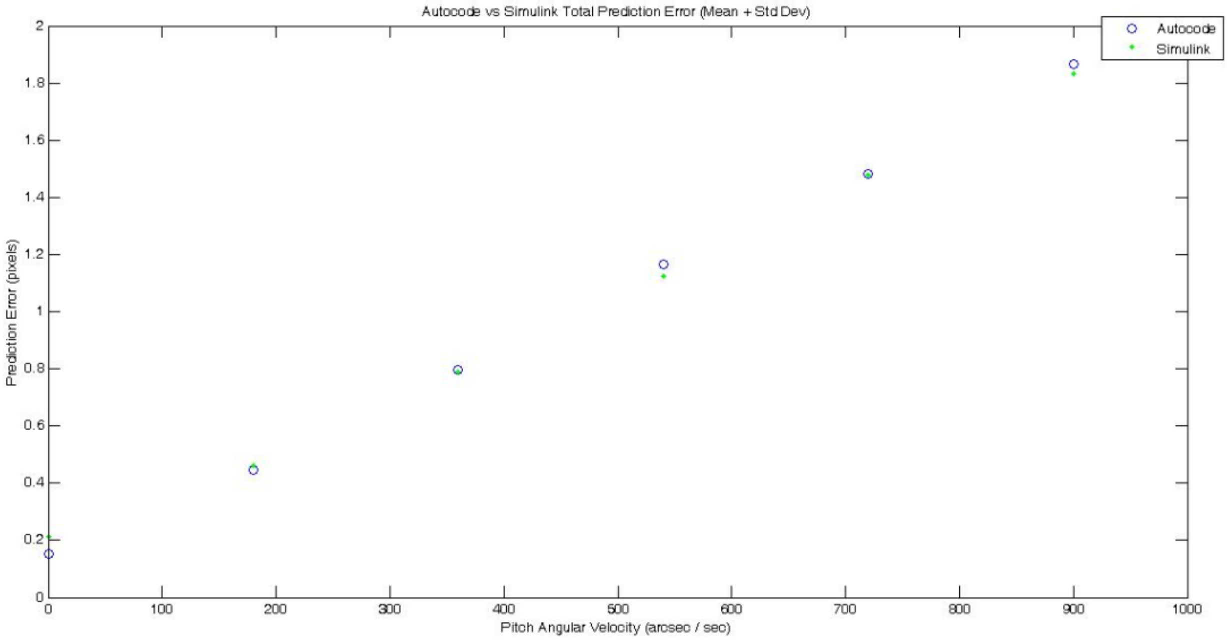


Figure 6-3: Simulink vs. Autocode Monte Carlo Results

The autocode and Simulink Monte Carlo results closely match. These results validate the Monte Carlo framework. Tunable parameters were successfully used to vary the spacecraft rate and random number generator seeds. The Monte Carlo analysis also validates a simple SWIL testbed for the fast centroiding algorithm because the algorithm was executed in C-language. The Monte Carlo framework and autocode tools can easily be applied to additional spacecraft simulations for rapid performance analysis.

Chapter 7

7. Conclusion

7.1. Summary

Complex scientific requirements are levied on small satellites and cubesats in order to reduce the cost of accessing space and completing research objectives. The science mission may necessitate precise pointing on the order of arcseconds. As CCD and CMOS detector technology improves, it becomes more feasible to develop star trackers for small satellites and cubesats. For high precision pointing, the star trackers may need to run at high rate. The objectives of this thesis were to develop image processing algorithms with reduced computation time in order to increase the update rate of the star tracker. Additional objectives were to develop a high fidelity star tracker model to analyze star tracker performance and utilize Draper's MATLAB autocode tools to test a software architecture for rapid Monte Carlo analysis.

A fast centroiding algorithm was derived. The centroid algorithm uses a center of mass calculation to determine the star locations in each image. The tracking code calculates the change in centroid position from the previous to the current frame. The centroid movement is used to estimate star tracker slew rate which provides the capability to estimate the centroid locations in the next image frame. The center of mass calculation is performed near the predicted star locations to speed up the centroid code. A robust star tracker model was developed in MATLAB Simulink. The model incorporates optical and detector noise to produce a simulated star tracker star field image. The fast centroiding algorithm was first analyzed in a simulation combining the star tracker model with the fast centroiding code. The fast centroiding algorithm was then tested with a camera, imager, and embedded avionics mounted on an air bearing platform. The testbed proved that the proposed algorithm can successfully track stars and reduce the centroid processing time. The air bearing data was analyzed to determine the slope of the trend line for increasing prediction error with increasing slew rate. Comparing the air bearing trend line to the simulation trend revealed that the best case air bearing trend approximately matched the worst case simulation trend. This result temporarily validates the star tracker model as a theoretical tool for analyzing star tracker performance. Finally, Monte Carlo simulations were conducted using a software architecture developed with Draper Laboratory MATLAB AutoCoder tools. The star tracker simulation and autocode tools can be used to perform analysis on future space systems.

7.2. Future Work

This thesis lays the groundwork for future research developing star trackers for CubeSats. More hardware testing is required to fully validate the star tracker model. The model noise parameters should be set to the data sheet parameters of the flight imager. The goal of the hardware testing is to compare simulated images to actual images. The noise parameters should be adjusted until the images match. Several different hardware tests can be conducted to validate the simulation. The detector can be set up to take dark frames. The dark frames are used to validate noise parameters such as dark current and fixed pattern noise. The imager should be placed in a spherical light integrator to capture full frame bright images. The bright images can validate saturation and PRNU. A CMOS detector should be set up in a laboratory environment to take images of a collimated stable light source with a point spread function spot size similar to a star. This test can validate the photon count and simulated point spread function. More complex testing may involve shining laser light on each individual pixel to test PRNU. If testing reveals the simulation does not accurately model the hardware, complex second and third order camera effects may need to be added to the simulation. However, this can be difficult, computationally intensive, and time consuming to develop. Another approach may be to just increase the simulated noise parameters such as dark current until the actual and simulated centroid error match. Once the simulation is validated it can be included in a larger spacecraft simulation to estimate pointing performance.

There are several improvements which could be made to the air bearing testbed. The wires leading to the avionics caused the air bearing to oscillate about a stable equilibrium point. A more sophisticated testbed could be developed that places a cubesat prototype on the air bearing complete with camera, imager, embedded processor, and reaction wheels. The reaction wheels can slew the spacecraft from one end of the monitor to the other to test the star tracking performance. The full star catalogue and attitude control software could be included in order to perform end to end pointing performance with star tracker data.

A full test of satellite pointing performance will require upgrades to the star tracker software. A LIS algorithm will have to match centroids to the star catalogue to make an initial attitude estimate. The algorithms developed in this thesis can successfully track a centroid as it slews across the detector. However, new algorithms will have to be developed to add windows to the centroid code as new stars are added to the detector. The star IDs from the catalogue will replace the ID algorithm proposed in this thesis.

References

[1] Smith, Matthew W. et al. "ExoplanetSat: detecting transiting exoplanets using a low-cost CubeSat platform." *Space Telescopes and Instrumentation 2010: Optical, Infrared, and Millimeter Wave.* Ed. Jacobus M. Oschmann et al. San Diego, California, USA: SPIE, 2010. 773127-14.

[2] Smith, Matthew W. et al. "The ExoplanetSat Mission to Detect Transiting Exoplanets with a CubeSat Space Telescope." *Proceedings of the 25th Annual AIAA/USU Conference on Small Satellites,* SSC11-XII-4, August, 2011.

[3] Pong, Christopher M. et al. "One-Arcsecond Line-of-Sight Pointing Control on Exoplanetsat, a Three-Unit CubeSat". *Proceedings of the American Astronautical Society Guidance and Control Conference*, 11-035, January, 2011.

[4] Pong, Christopher M. et al. "Achieving high-precision pointing on ExoplanetSat: initial feasibility analysis", *Proc. SPIE* 7731, 77311 V. 2010.

[5] Kitahara H. et al. "The ETS-VI Satellite." *American Institute of Aeronautics and Astronautics, Inc.* 1993: pg. 1095-1101.

[6] Freidell, James E., Peter Brunt, and Kenneth E. Wilson. "Why laser communication crosslinks can't be ignored." *AIAA International Communications Satellite System Conference*. Washington DC, USA: American Institute of Aeronautics and Astronautics, 25-29 Feb, 1996. A96-21571 04-32 pg. 1397-1409

[7] Arimoto, Yoshinori. et al. "Laser Communication Demonstration Experiment on International Space Station." *American Institute of Aeronautics and Astronautics, Inc.* 1998: pg. 162-169.

[8] Miller, David W., Swati Mohan, and Jason Budinoff. "Assembly of a Large Modular Optical Telescope (ALMOST)." *Space Telescopes and Instrumentation 2008: Optical, Infrared, and Millimeter*. Ed. Jacobus M. Oschmann et al. Marseille, France: SPIE, 2008. 70102H-11.

[9] Liebe, Carl Christian. "Accuracy Performance of Star Trackers—A Tutorial." *IEEE Transactions on Aerospace and Electric Systems*. 38.2 April 2002: pg. 567-599

[10] Jia, Hui. et al. "Systematic error analysis and compensation for high accuracy star centroid estimation of star tracker." *Science China Technological Sciences*. 53.11 Nov. 2010.

[11] Rufino, Giancarlo and Domenico Accardo. "Enhancement of the centroiding algorithm for star tracker measurement refinement." *Elsevier Science Ltd. 2003: pg. 135-147* <http://www.sciencedirect.com>.

[12] Li, Yufeng., Dongmei Li, and Yanbo Liang. "Subpixel centroiding algorithm for EMCCD star tracker." *2008 International Conference on Optical Instruments and Technology: Advanced Sensor Technologies and Applications*. Ed. Anbo Wang et al. Beijing, China: SPIE, 2009 715715-6

[13] Singla, Punett, John L. Crassidis, and John L. Junkins. "Spacecraft Angular Rate Estimation Algorithms for Star Tracker-Based Attitude Determination". AAS 03-191 Web. 16 May 2012.

[14] Crassidis, John L. "Angular Velocity Determination Directly from Star Tracker Measurements." Web. 16 May 2012.

[15] Kolomenkin, Michael. et al. "A Geometric Voting Algorithm for Star Trackers." *IEEE Transactions on Aerospace and Electronic Systems*. 44.2 April 2008: pg. 441-456

[16] Strang, Gilbert. *Computational Science and Engineering*. Wellesley: Wellesley-Cambridge Press, 2007.

[17] Fujimori, Iliana L. "CMOS Passive Pixel Imager Design Techniques." *Massachusetts Institute of Technology*. Thesis. Feb. 2002.

[18] Martinec, Emil. "Noise, Dynamic Range and Bit Depth in Digital SLRs." 22 May 2008. <http://theory.uchicago.edu/~ejm/pix/20d/tests/noise/>. 16 May 2012.

[19] Merline, W.J. and Steve B. Howell. "A Realistic Model for Point-Sources Imaged on Array Detectors: The Model and Initial Results." *Experimental Astronomy*. 6.1-2 1995: pg. 163-210

[20] Janesick, James R. *Scientific Charge-Coupled Devices*. Bellingham: SPIE Press, 2001.

[21] McComas, Brian K. "Toolkit for Remote-Sensing Analysis, Design, Evaluation and Simulation." *Modeling, Simulation, and Calibration of Space-based Systems*. Ed. Pejmun Motaghedi. Bellingham, WA, USA: SPIE, 2004. 5420

[22] Jonathan P. How. Stochastic Estimation and Control. Course Notes, Massachusetts Institute of Technology, Aeronautical and Astronautical Engineering Department, Feb 2011.

[23] "CYIH1SM1000AA-HHCS Detailed Specification-ICD". *Cypress Semiconductor Corporation*. 18 Sept. 2009. <http://www1.futureelectronics.com/doc/CYPRESS/CYIH1SM1000AA-HHC.pdf>. 7 May 2012.

[24] Crowell, Corey Whitcomb. “Development and Analysis of a Small Satellite Attitude Determination and Control System Testbed.” *Massachusetts Institute of Technology*. Thesis. June 2011.

[25] “MATLAB and Simulink.” *The MathWorks, Inc*. <http://www.mathworks.com>. 16 May 2012.