NEW CALIBRATION TECHNIQUE FOR ALL SKY CAMERAS

BY

STEVEN M. BANNISTER, B.S.E.E.

A thesis submitted to the Graduate School
in partial fulfillment of the requirements
for the degree

Master of Engineering

Major Subject: Electrical Engineering

New Mexico State University

Las Cruces New Mexico

December 2012
“A New Calibration Technique for All Sky Cameras,” a thesis prepared by Steven M. Bannister in partial fulfillment of the requirements for the degree, Master of Engineering, has been approved and accepted by the following:

Linda Lacey
Dean of the Graduate School

Laura Boucheron
Chair of the Examining Committee

Date

Committee in charge:

Dr. Laura Boucheron, Chair

Dr. David Voelz

Dr. Nancy Chanover
DEDICATION

I dedicate this work to my mother Rhonda, my father Jody, my brother and sister Daniel and Demi, my grandparents Bernard and Jeanie Rosprim and Garry and Mary Ann Bannister.
ACKNOWLEDGMENTS

I would like to thank my advisor, Laura E. Boucheron, for her guidance, encouragement and patience. I would also like to thank David Voelz for supporting this work and providing the means for its completion. I give special thanks to the people at Sandia National Laboratories for funding the SkySentinel project that cooperated with my research.
February 9, 1987  Born in Page, Arizona

2005-2010  B.S.E.E., New Mexico State University, Las Cruces, New Mexico

2010-2012  Graduate Research Assistant, New Mexico State University, Las Cruces, New Mexico.

FIELD OF STUDY

Major Field: Digital Signal Processing
ABSTRACT

A NEW CALIBRATION TECHNIQUE FOR ALL SKY CAMERAS

BY

STEVEN BANNISTER, B.S.E.E.

MASTER OF ENGINEERING

New Mexico State University

Las Cruces, New Mexico, 2012

Dr. Laura E. Boucheron, Chair

A well documented and stable method for calibrating all-sky cameras is needed in the field of meteor monitoring camera networks. This paper will discuss the development and testing of a new method for calibration that utilizes portions of traditional methods that require prior knowledge about the camera’s distortion characteristics. The set of parameters that describe the camera’s zenith, its position relative to cardinal directions and the lens distortion characteristics are crucial to estimating the trajectory of a meteor event. Traditional methods for determining these parameters have been used for more than thirty years but this paper will discuss potential areas of improvement for these methods. Simulated
calibrations are discussed in order to exhaustively explain the behavior of certain combinations of calibration parameters. The new calibration process can be shown to be more stable in terms of the precision needed for initialization parameters and simulation results as well as the calibration of actual cameras help to show this. The main advantage to this method is that no information about the camera’s lens distortion is needed to converge to parameters that provide an accurate distortion model.
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>xi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xiii</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 SKY SENTINEL CAMERA NETWORK</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Network Functions</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Camera Set Up</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Fisheye Lens Characterization</td>
<td>8</td>
</tr>
<tr>
<td>3 CALIBRATION</td>
<td>10</td>
</tr>
<tr>
<td>3.1 Astronomical Coordinate System</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Horizontal Coordinate System</td>
<td>10</td>
</tr>
<tr>
<td>3.3 Equatorial Coordinate System</td>
<td>11</td>
</tr>
<tr>
<td>3.4 Current Calibration Method</td>
<td>12</td>
</tr>
<tr>
<td>4 CALIBRATION PROCESS</td>
<td>16</td>
</tr>
<tr>
<td>4.1 Set up</td>
<td>18</td>
</tr>
<tr>
<td>4.2 Parameter Determination</td>
<td>19</td>
</tr>
<tr>
<td>4.3 Error Residual</td>
<td>19</td>
</tr>
<tr>
<td>4.4 Integration Frame Characteristics</td>
<td>20</td>
</tr>
<tr>
<td>4.5 Preprocessing Integration Frames</td>
<td>20</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

1. Average Error Residuals: Manual Calibrations .................................. 60
2. Days to Beginning of Month .......................................................... 72
3. Days Since J2000 to Beginning of Each Year .................................. 73
# LIST OF FIGURES

1. Node locations as of 6/29/2012. ........................................ 5
2. Cross-section of camera set up. ........................................ 6
3. Results of rainbow lens characterization. .......................... 8
4. Horizontal coordinate system. ........................................ 11
5. Equatorial coordinate system. ........................................ 12
6. Calibration parameter visual. ......................................... 13
7. Consecutive integration frames. ..................................... 23
8. Differenced integration frames. ..................................... 24
9. Translation of stars within integration frames. .................. 25
10. Star chart generated with fourmilab's yourskey. ............... 27
11. Test Points Used in Simulations ................................. 32
12. Results for 5 training points ..................................... 33
13. Results for 7 training points ..................................... 34
14. Results for 9 training points ..................................... 35
15. Results for 11 training points ................................... 36
16. Results for 13 training points ................................... 37
17. Results for 15 training points ................................... 38
18. Average error residuals vs. number of training points ....... 39
Azimuth residuals reported by Borovicka for variations of his method
Zenith residuals reported by Borovicka for variations of his method
Average error residuals vs. $a_0$ value
Average error residuals vs. $E$ value
Average error residuals vs. $\epsilon$ value
Average error residuals with training points isolated to Quadrant II
Average error residuals with training points isolated to Quadrant IV
Average error residuals when zenith values of training points $< 45^\circ$
Average error residuals when zenith values of training points $> 45^\circ$
Average error residuals when zenith values of training points $> 75^\circ$
Plots of how the lens is modeled for variations in zenith training point values
Average error residuals for node in Parker, AZ
Average error residuals for node in Kingman, AZ
Average error residuals for node in Prescott, AZ
1 INTRODUCTION

Cameras that are capable of viewing the entire night sky in their field of view are referred to as all-sky cameras. In the past astronomers have used traditional photographic cameras with long exposures in an attempt to capture meteors. With the development of technology such as video cameras and motion capture software, we are now able to obtain time-stamped videos of meteors that allow for precise calculations of meteor properties such as trajectories. A growing need for these cameras in the field of meteor astronomy has arisen but it is difficult to translate the \((x,y)\) pixel coordinates into the azimuth angle and zenith distance, \((a,z)\) needed for astronomical calculations.

Traditional methods for calibrating cameras with fisheye lenses that are intended to be used for monitoring the sky can be difficult to implement. One such calibration method was documented by [7] specifically for cameras using photographic plates. This method has been altered on numerous occasions to improve accuracy([2] [3] [14] [11]). It can be difficult to find all the documentation needed to implement one of the more recent versions of the method. This thesis will propose a few alterations to the current calibration process [3] as well as provide a full step-by-step directive for executing the process.

Many all-sky cameras use a fisheye lens that allows for imaging of the entire visible hemisphere. The lenses introduce a distortion that complicates the
recording of astrometric measurements. Methods that are being used for all-sky camera calibration today require extensive knowledge of the wide angle lens’ characteristics in order to achieve an accurate fit. The parameters of the exponential equation used to fit the lens are quite hard to find with iterative convergence methods unless an accurate approximation of these parameters is already known. The proposed alteration to the calibration method will include a model for lens distortion that uses a quadratic equation which can be shown to be a more stable equation to find suitable parameters.

There are many applications for all-sky video cameras but this thesis will focus on utilizing them for meteor monitoring. Meteor events vary greatly in their brightness but can often be captured even with low light cameras. Typically an event of visual magnitude 1.5 or brighter can be seen with relatively cheap digital video cameras. Although, most often, it is the extremely bright events that are of interest to meteor enthusiasts because they are more likely to produce meteorites that can be found with trajectory calculation. Bright meteors often emit secondary particulate matter and information about their mass and disintegration properties can be obtained from spectra analysis [4] [12].

Multiple all-sky cameras in a localized area with similar fields-of-view are referred to as an all-sky camera network. Networks allow for a greater chance of catching individual meteors and provide the possibility of determining meteor trajectories by means of triangulation [7] [1]. Currently, there are a number of
networks in operation and each one has its own goal or purpose. This thesis was written in cooperation with the NMSU All Sky Camera Network (SkySentinel). Sandia National Laboratories has also played a major role in funding the project as well as fielding cameras and developing the software used for event detection. The administration of this network has a mission that is to create a network that can monitor, track, and analyze atmospheric meteor events in order to provide a database for assisting satellite operators in separating natural and man-made events and for instrument calibration tasks.

Amateur meteor watchers might find it sufficient to simply capture an event for viewing purposes. However, if any scientific data, such as meteor trajectory, is meant to be taken from a video then the camera must be calibrated to the stars in the night sky. All-sky calibration is necessary because it allows for a relationship between the camera’s image plane/pixel-location and the astronomical azimuth angle and zenith angle \((a,z)\).

The coordinates \((a,z)\) represent the location of celestial bodies from a given location on the Earth’s surface. A given celestial body will have a different \((a,z)\) value for every location on Earth’s surface and for every instance of time. For this reason there is also a set of astronomical coordinates called Right-Ascension and Declination \((RA,Dec)\) which have constant values for all stars that are visible from Earth. The \((RA,Dec)\) values of stars can be converted to \((a,z)\) for a specific location and time with the knowledge of the location’s latitude and longitude as
well as the current time, day and year. A detailed explanation of this conversion is available in Appendix A.

Once a relationship between \((x,y)\) pixel coordinates and \((a,z)\) has been determined, the operator of a given camera can evaluate the \((a,z)\) of the path of any meteor that is captured on the camera. If multiple calibrated cameras in the same network capture the same event then a meteor trajectory can be estimated using methods based on triangulation[7] [1]. The trajectory of a meteor can be used to determine the impact point or the celestial origin of the meteor.
2 SKY SENTINEL CAMERA NETWORK

New Mexico State University (NMSU) has partnered with Sandia National Laboratories in an attempt to field a network of all-sky cameras across the United States and some foreign countries. The goal of this network is to investigate, improve, and expand a ground-based all-sky camera network necessary for calibration of our nation’s nuclear detection satellites to separate natural and man-made events. The project has been in operation since late 2009. As of June 29, 2012 there were a total of 65 nodes hooked into the network with about 30 of those nodes regularly delivering useful data. Figure 1 shows the locations of most of the nodes that have been fielded. There are also a few nodes that are in locations that do not appear on this map.

Figure 1: Node locations as of 6/29/2012.
2.1 Network Functions

Nodes are operated by volunteers but they report to a central server. The server is meant to catalog events and make them viewable to the public through the network’s website (http://skysentinel.nmsu.edu/allsky). Ideally, the server will also correlate events that are seen by multiple nodes. Correlated events allow for trajectory estimations to be made. The SkySentinel network will also be capable of removing anomalies from event captures and discriminating against false captures. False captures include things such as airplanes, birds, bugs, and lightning. Although currently the number of nodes is around 65, the goal is to field 100+ nodes across the continental United States.

2.2 Camera Set Up

Figure 2: Cross-section of camera set up.
A cross-section of the camera device set up is shown in Figure 2. The PVC housing completely encloses the entire system in order to protect it from the elements. The acrylic dome at the top of the structure provides a transparent enclosure that allows for imaging with the low-lux ccd and fisheye lens imaging system. The heaters and fan work with the thermo-controller to maintain a good working temperature as well as keep moisture from accumulating on the inside of the dome. There are two cables that protrude from the bottom of the enclosure. One of the cables provides power for the camera and temperature control systems. The second cord is a BNC cable that carries the video signal to a computer that contains the video capture software.

Sentinel Video software was developed by Sandia National Laboratories and it controls the video processing for the all-sky cameras. Sentinel has triggering software that determines when to begin recording an event as well as when to terminate the capture. Once an event is recorded it is delivered to the central server at NMSU where it is catalogued. Catalogued events have a composite JPEG file that shows a time lapse of the meteor path, and MP4 video of the event, and text files that contain time and date information. Every camera node also captures an integration frame once every hour during the night. An integration frame is essentially a single image taken with a 34 second exposure in order to image a large number of stars. These integration frames are the key components to performing a calibration they are further discussed in Section 4.4.
2.3 Fisheye Lens Characterization

The fisheye lens that is used in the SkySentinel cameras is a Rainbow fisheye lens (Part no. L163VDC4P). In order to characterize the behavior of the lens an experiment was conducted that determined the relationship between the angle of incidence and the image point. The full experiment is documented in Appendix B but the results can be seen in Figure 3.

![Rainbow Lens Characterization](image.png)

Figure 3: Results of rainbow lens characterization.

Fisheye lenses behave in a manner that causes images to be stretched when in the center of the image and compressed near the edges. The solid line in Figure 3 shows the response of the Rainbow fisheye lens for a range of incidence angles. The dashed line shows a linear mapping for the relationships between incidence angles of $0^\circ$ to $86^\circ$ and radial pixel distances of 0 to 240. The dashed line shows
what a lens response would look like if there were a uniform relationship between incidence angle and radial image location. The Rainbow lens shows an increase in radial distance for changes in incidence angles near $0^\circ$ and smaller radial change for incidence angles nearing $90^\circ$. Although the Rainbow lens does not react extremely different than a linear lens would, it causes enough distortion that the lens will need to be correctly modeled for an accurate calibration.

This calibration method uses a quadratic equation to model the lens characteristics. MATLAB’s \textit{polyfit} function was used to determine the parameters that would most closely model the response of this lens. The values that resulted were:

$$P_1 = 3.24 \times 10^{-6}, P_2 = 0.0054, P_3 = 0.012$$

where,

$$u = P_1 r^2 + P_2 r + P_3$$

and $u$ is the angle of incidence and $r$ is the radial distance from the center of projection (COP) in pixels. It is worth noting that the quadratic term is several orders of magnitude smaller than the the linear term which indicates a small deviation from a linear behavior.
3 CALIBRATION

3.1 Astronomical Coordinate System

There are a variety of coordinate systems that can be used to represent the locations of celestial bodies seen from Earth. The calibration method described in this paper uses a horizontal coordinate system that uses the observer’s local horizon as the fundamental plane. The position of a given star has a different value for every geographic location and time instance when using horizontal coordinates. Equatorial coordinates provide constant specifications for celestial bodies based on geocentric coordinates.

3.2 Horizontal Coordinate System

Horizontal coordinates are given in terms of altitude, sometimes referred to as elevation, and azimuth. Azimuth is the angle around the horizon. An azimuth of 0° usually refers to true North and increases towards East. Altitude is the spherical representation of the angle from the horizon and it can vary from 0° to 90°. Sometimes a zenith angle is used instead of altitude. Zenith is the point directly overhead and the zenith angle increases towards the horizon. For this calibration method we will use azimuth \( a \) and zenith \( z \) as our horizontal coordinates. Figure 4 shows the horizontal coordinate system.
3.3 Equatorial Coordinate System

The equatorial coordinate system is a geocentric coordinate system which means that the center of the Earth is the point of reference. It has a right-handed convention and uses a projection of the Earth’s equator onto a celestial sphere as its fundamental plane. The spherical coordinates of stars are expressed as right ascension and declination. The \((RA & DEC)\) values for stars change very slowly and are only adjusted every fifty years. The most recent representations for the \((RA & DEC)\) values of celestial bodies were recorded in January 2000 and they are still the references used today [8]. This coordinate system allows for observations made on different dates to be compared directly. Conversion from right ascension and declination \((RA & Dec)\) to azimuth and zenith \((a,z)\) is explained in detail in

Figure 4: Horizontal coordinate system \((made available by [6])\).
Appendix A.

Figure 5: Equatorial coordinate system (*made available by [6]*)

3.4 Current Calibration Method

Calibration methods for all-sky cameras have been investigated at least as far back as 1987 [7]. The method developed by Ceplecha has been modified by Borovicka [2] and [3] to improve accuracy. The most recent calibration method requires the determination of the parameters \((A,F,C,V,D,S,P,Q,a_0,E,\epsilon,x_0,y_0)\) in the following equations:

\[
\begin{align*}
  r &= C\sqrt{(x-x_0)^2 + (y-y_0)^2} + A(y-y_0)\cos(F-a_0) - A(x-x_0)\sin(F-a_0) \\
  u &= Vr + S(e^{Dr} - 1) + P(e^{Qr^2} - 1)
\end{align*}
\]
\[ b = a_0 - E + \tan^{-1}\left(\frac{y - y_0}{x - x_0}\right) \]  

(3)

\[
\cos(z) = \cos(u) \cos(\epsilon) - \sin(u) \sin(\epsilon) \cos(b) 
\]  

(4)

\[
\sin(a - E) = \sin(b) \sin(u) / \sin(z) 
\]  

(5)

Figure 6: Calibration parameter visual. *The parameters A,F,C,V,D,S,P and Q are difficult to visualize so their representations are not seen here.*

A total of 13 reduction constants need to be determined. Equation (1) contains the term \( r \) which represents the radial distance of a given point from the center of projection, COP. The reduction constants \( A, F \) and \( C \) are camera constants that define the position of the CCD relative to the camera’s optical axis. The inclination of the plate is described by \( A \) and \( F \) while \( C \) expresses the shift of the plate along the optical axis. Equation (2) represents the azimuth, \( u \), of a given point from the camera’s COP. The value for \( u \) is determined by the
reduction constants $V$, $S$, $D$, $P$ and $Q$. These constants are called the lens constants and they define the distortion that is introduced by the camera’s wide-angle lens. The left side of Equation (3) contains the term $b$ which represents the zenith of a given point from the camera’s COP. The zenith of projection is described by the reduction constants $a_0$, $E$, $x_0$ and $y_0$. These reduction constants determine the origin of the camera’s COP as well as how far the COP lies from true zenith. The constant $a_0$ defines the rotation of the camera’s x-axis from the cardinal direction South. The coordinates $(x_0, y_0)$ represent the position of true zenith. The remaining two constants, $E$ and $\epsilon$ are a spherical representation of the location of the camera’s COP from true zenith. The azimuth angle of the COP is $E$ and the zenith distance is $\epsilon$. The terms $x$ and $y$ are the pixel locations of a given point and $a$ and $z$ are the associated azimuth angle and zenith distance [3].

The process used for determination of these constants is explained in [3]. It is similar to the procedure discussed in this thesis but modifications have been made for the new procedure and a full implementation of the previous method is not possible solely with the instructions provided in this document.

The most recent version of modifications [3] reports residual errors of no more than $0.05^\circ$ for zenith measurements and $0.03^\circ$ for azimuth measurements. This is very good accuracy and there is not a strong demand from meteor watching communities to improve upon these measures. However, when attempting to
implement this method for use with the Sky Sentinel All-Sky video network it became apparent that many of the constants were not easily converged upon with the suggested least-squares method. Particularly, the lens constants $V, S, D, P$ and $Q$ were not effectively obtained without initializing with values determined by an accurate lens characterization. Also, it can be difficult to determine reasonable initialization values for the camera constants $A, F$ and $C$. 
4 CALIBRATION PROCESS

The proposed method for calibration uses similar equations to the existing methods and in some cases uses the exact same equations. The main alteration is that a polynomial is used for the determination of \( u \) instead of the exponential from Equation (2). In [3] the constants \( A, F \) and \( C \) were introduced to eliminate residual errors introduced by lens discontinuities for varying azimuth values. For this implementation we have chosen to exclude those values, but if more precision is required then they could be re-introduced into Equation (6).

A single image, sometimes called an integration frame, taken from the camera is used for calibration. There are eight parameters to find in the equation set that is used for calibration. The equations are as follows:

\[
\begin{align*}
    r &= \sqrt{(x - x_0)^2 + (y - y_0)^2}\quad (6) \\
    u &= P_1 \cdot r^2 + P_2 \cdot r + P_3\quad (7) \\
    b &= a_0 - E + \tan^{-1}\left(\frac{y - y_0}{x - x_0}\right)\quad (8) \\
    \cos(z) &= \cos(u) \cos(\epsilon) - \sin(u) \sin(\epsilon) \cos(b)\quad (9) \\
    \sin(a - E) &= \sin(b) \sin(u) / \sin(z)\quad (10)
\end{align*}
\]

Substituting Equations (6), (7) and (8) into Equations (9) and (10) results in two independent equations with eight unknown parameters \( P_1, P_2, P_3, a_0, E, \epsilon, \ldots \)
\[ x_0, y_0 \].

\[
\cos(z) = \cos(P_1 \star (\sqrt{(x-x_0)^2 + (y-y_0)^2})^2 + P_2 \star (\sqrt{(x-x_0)^2 + (y-y_0)^2})^2 + P_3) \star \\
\cos(\varepsilon) - \sin(P_1 \star (\sqrt{(x-x_0)^2 + (y-y_0)^2})^2 + P_2 \star (\sqrt{(x-x_0)^2 + (y-y_0)^2}) + \\
P_3) \star \sin(\varepsilon) \cos(a_0 - E + \tan^{-1}(\frac{y-y_0}{x-x_0})) \tag{11}
\]

\[
\sin(a - E) = \frac{\sin(P_1 \star (\sqrt{(x-x_0)^2 + (y-y_0)^2})^2 + P_2 \star (\sqrt{(x-x_0)^2 + (y-y_0)^2}) + P_3)}{\sin(z)} \tag{12}
\]

Each \((x,y)\) pair is associated with a single star, or training point. It might be the case that a star covers multiple pixels in which case the centroid pixel is to be used for the \((x,y)\) representation of each star. There are eight unknown parameters and two independent equations, so there must be at least four stars in the training set in order for the unknowns to be found. Each additional training point will allow for a more accurate fit of the unknowns, the trade off between number of training points and accuracy will be discussed in a later section.

Stars within the calibration frame are to be identified by their right ascension and declination (\(RA\ & \ Dec\)) coordinate values. An explanation of equatorial coordinates can be found in Section 3.3. The (\(RA\ & \ Dec\)) of each star in the integration frame should be used to find the azimuth and zenith \((a,z)\) values for
the camera’s location. The process for converting \((RA \& Dec)\) to \((a,z)\) can be found in Appendix A. The \(x\) and \(y\) values along with the \(a\) and \(z\) values for each star are all that is required to calibrate a given node. The full calibration process can be accomplished with a single use of a non-linear solver such as least squares or the Levenberg-Marquardt algorithm [9]. For this study we used MATLAB’s \textit{fsolve} as our equation solver which utilizes the Levenberg-Marquardt algorithm for optimization of the equation set.

4.1 Set up

Perhaps the most important element of this calibration process is to choose an algorithm that will solve the set of equations accurately. This implementation uses the Levenberg-Marquardt algorithm because it is readily available on MATLAB systems that include the \textit{fsolve} function. Calibration methods in the past ([7] [2] [3]) have been documented as using least-squares determinations.

Once a solver has been chosen, initial parameter values need to be set up. When MATLAB’s \textit{fsolve} is used with the Levenberg-Marquardt algorithm, most of the initial conditions are not sensitive to their actual values. Six of the eight parameters can simply be initialized as zero. The two parameters that need to be estimated prior to calibration are \(x_0\) and \(y_0\). These values should initially be set to approximately the center of the camera’s field of view. For example, if the frame size is 640 by 480 pixels then the initial values for \(x_0\) and \(y_0\) should be 320
and 240 respectively.

4.2 Parameter Determination

Now that all the parameters have been given initial values and a solver has been chosen, it is time to determine the actual parameter values. The goal is to find values for the eight unknowns that will make Equations (11) and (12) true. Most likely it will not be possible to exactly fit the equations, in which case the parameters that provide the smallest amount of error should be chosen.

4.3 Error Residual

In order to determine the error associated with a particular set of parameter values, Equations (6 - 10) are used with the original \((x, y)\) pairs to return values for \(a\) and \(z\). The solved \(a\) and \(z\) values are compared to the actual values to determine residual error values.

Equation (10) requires inverse trigonometry to return the \(a\) value. The use of the inverse sine function introduces a quadrant problem that forces all values of \(a\) to fall in quadrant I or quadrant IV. To fix this problem there are some conditional adjustments that must be made.

1. Determine the inverse sine of the right side of Equation (10). We will call this value \(aE\).

2. Solve the equation: 
\[
aE^* = \frac{(\cos(u) - \cos(z) \cos(z)}{\sin(z) \sin(z)}
\]
3. If the value $aE^*$ is negative then the correct value for $a$ is $\pi - aE$.

Occasionally, the values for $a_0$, $E$ or $\epsilon$ will be negative which can cause $a$ values to also be negative. If non-negative $a$ values are desired for an easier visualization then a simple check can be used to return the desired values. If Equations (6 - 10) return a negative $a$ value for a particular $(x,y)$ pair then simply add $2\pi$ to $a$ to return the equivalent non-negative value.

### 4.4 Integration Frame Characteristics

In order to calibrate an all-sky camera, we need an integration frame that we use to determine the $(x,y)$ pixel location of visible stars. An integration frame is an image that is captured through a long exposure in order to image some of the stars that would not be seen in a shorter capture. Ideally, the integration frame will contain at least 15 stars evenly distributed about the camera’s field of view (FOV). It is important to document the exact geographic location, date and time that the integration frame was captured because that information will be used to determine the azimuth and zenith values for the stars present in the frame.

### 4.5 Preprocessing Integration Frames

There are a few preprocessing techniques that need to be performed on the integration frames before they are ready for the calibration process. A single integration frame with the camera setup used by the SkySentinel network will
usually image about four or five stars. This is not enough to do an accurate calibration considering there will be some measurement noise mainly due to the resolution of the camera. A star will usually cover multiple pixels in an image and it can be hard to pinpoint the exact centroid in the \((x,y)\) plane. There is also going to be some atmospheric distortion that will cause some measurement noise. This is why it is important to get somewhere between eleven and fifteen stars for an accurate calibration. We specify at least eleven stars because of the results from Section 5.2.

In order to have the desired number of stars for training, we use multiple integration frames for a single calibration plate. For example, a single node might image the same five stars in each integration frame which are captured about once an hour throughout the night. The five visible stars will move across the camera’s field of view through the night and create more training points. If three consecutive frames are used then there will be fifteen training points available.

Even when a node is capable of imaging several stars it can be difficult to identify those stars within an integration frame. One problem is that over time cameras develop pixels that have been damaged from too much terrestrial cosmic radiation exposure [10]. The damaged pixels will stay "hot" which means that they appear to be receiving light even when they are not. Because a hot pixel covers a very small area in the image, usually a single pixel, they are hard to distinguish from stars in an integration frame. Using multiple integration frames
for a single calibration provides a method for masking out unwanted objects in the images.

The use of multiple integration frames for a single calibration provides a method for masking out unwanted objects such as hot pixels [13]. Subtraction of the pixel values of a frame captured at time \( t_1 \) from a frame captured at time \( t_0 \) results in large values for any non-stationary objects, such as stars, present at \( t_0 \). Large negative values will result from non-stationary objects present a time \( t_1 \). Stationary objects, hot pixels and background will result in small values. A thresholding operation can then be used to eliminates hot pixel and stationary objects that both frames share while revealing non-stationary objects that are only present in the \( t_0 \) frame. Therefore, if the stars present in both frames have changed their position in the image from time \( t_0 \) to \( t_1 \) then we can isolate the stars in a single image. Enhancing the contrast of the resulting frame will boost the visibility of the stars present and we can estimate their centroids. Once the unwanted material in the frame has been removed we can perform a contrast enhancement technique, such as multiplying all pixels by some value, to improve the visibility of non-stationary objects, stars, present at time \( t_0 \). Contrast enhancement for this application was done by multiplying all pixel values by three.

Figure 7 show two integration frames from the same camera. The images were taken from a node located at Sandia National Laboratories in Albuquerque, NM. The image referenced by time \( t = t_0 \) was captured on May 20, 2012 at 4:25am
UTC, and the second image, captured at time $t = t_1$ was captured on the same day at 5:25am UTC.

![Consecutive integration frames](image)

Figure 7: Consecutive integration frames.

It can be seen in Figure 7 that there are a large number of white pixels that are shared between the two images and only a few that are not shared. It is the goal of pixel differencing to isolate those pixels which are not common between the two images. Figure 8 shows the complement of the result of subtracting the image taken at time $t = t_0$ from the image at time $t = t_1$.

The complement of the differenced frames is shown in Figure 8 so that it is easier to see the small image points that represent stars. Also, a circle has been drawn to show the location of the edges of the camera’s FOV.

As noted earlier, image differencing is capable of revealing stars in a cluttered frame because they are essentially the only light source that is translating across the camera’s FOV. With the exception of planets and periodic sources of light
such as stadium lights from a nearby city, the location of stars should be the only change from one integration frame to the next. Figure 9 shows the translation of five stars from consecutive integration frames.

![Differenced Image](image)

**Figure 8: Differenced integration frames.**

The differenced image in Figure 8 appears to contain around 20 stars. However, most of the fainter points in the image are nearly impossible to correctly identify due to the high volume of visible stars in the night sky. In order to avoid misrepresentation of stars, it is recommended that only the very bright and distinguishable stars are used for calibration purposes. The SkySentinel camera setup is, on average, capable of imaging 5-8 distinguishable stars in a single integration frame. This means that in order to meet the minimum requirement of fifteen training points we must obtain three or more differenced frames for an accurate calibration. It is important to note that, even after differencing, not all
distinguishable image points are actually stars. Some remaining artifacts could be planets or light sources that appeared in only one frame such as a strong source of light from a nearby city. Correctly identifying stars is explained in the next section.

![Figure 9: Translation of stars within integration frames.](image)

4.6 Identifying Stars

One of the more difficult problems facing an all-sky camera operator when performing a calibration is to identify the stars in an integration frame. The task requires matching the relative star positions in the frame to stars in a star chart. The exact method for choosing how to determine when a star in a frame corresponds to a specific star in a star chart cannot be easily explained. Instead, the best advice that can be given is to look for specific patterns in the relative
star locations and try to match those stars between the frame and the chart. It is important to note that the camera frame might not be oriented in the same way as your star chart so there might be a rotation present between the two representations. An example of misrepresentation of stars and the reduction in accuracy that it causes can be seen in the results Section 5.6.

Star charts can be obtained in a variety of ways. There are internet sites dedicated to producing plots of visible stars in a specific location. Most star chart generators allow for variations of which stars to plot based on visual magnitude. One such site uses the Yoursky charting tool made available by Fourmilab (http://www.fourmilab.ch/yoursky/). This star chart generator takes geographic coordinates, time and date, and visual magnitude as its inputs and produces a circular FOV star chart. This chart should correspond fairly well to the stars present in a specific integration frame as long as the visual magnitude setting roughly corresponds to the sensitivity of the cameras that captured the frame. Figure (10) shows a star chart generated with Yoursky.

If the inclusion of an outside star chart source is not desired then it is possible to generate star charts with the use of a star catalog. A star catalog is a list of visible stars that usually contains the names, locations and visual magnitudes for each star. For chart generation the star catalog must list stars based on visual magnitude and include the (RA & Dec) values for each star. The catalog does not need to include all the stars visible from Earth but it does need to include...
Figure 10: Star chart generated with fourmilab’s yoursly.

any stars that meet the visual magnitude requirements of the camera being used.

Generating a star chart will require the use of the ($\text{RA} \& \text{Dec}$) to ($a,z$) conversion method. This conversion process is explained in detail in Appendix ???. Once the ($\text{RA} \& \text{Dec}$) values for all stars have been converted to ($a,z$) we must choose which stars are actually going to be seen from our current location. Stars that we can expect to be seen are the stars that have zenith values between $0^\circ$ and $90^\circ$. The ($a,z$) values for all stars that meet the visual magnitude requirements and are visible from the current location can be plotted with the polar convention. The plotted stars should complete a star chart that shows relative positions of all the stars that can be seen in the integration frame.

Once stars have been identified in their respective integration frames, it is
possible to obtain parameters that solve, or come very close to solving, Equations (11) and (12) or (9) and (10).
5 TESTING THE METHOD

Calibration of real integration frames can be cumbersome because each star needs to be located in the frame and identified so that the (RA & Dec) values can be recorded and converted to (a, z). Locating and identifying stars within an integration frame is difficult because the image of a star is extremely small, in comparison to the rest of the frame, and the only way to correctly identify an individual star is to compare its position relative to other stars within the frame. For this reason it is hard to correctly identify the first star in order to begin identifying others. Extensively testing a calibration method would consume a large amount of time if it were done only with real frames because it would require multiple calibrations.

5.1 Simulation Test Set Up

In order to effectively test this method, in a timely matter, a simulated-frame testing procedure was developed. Simulated frames consist of randomly generated (x, y) star positions. The (x, y) values were converted to (a, z) using Equations (11) and (12) with realistic values for the calibration reduction constants. The two pairs of coordinates were used as inputs for the equation solver and the reduction constants, or calibration parameters, were estimated using the Levenberg-Marquardt algorithm. The estimated constants were compared to the
known values to check how well the calibration method worked.

A variety of simulations were run in order to test the method’s sensitivity to different conditions. The conditions that were tested were the number of training points used, the location of the training points, and variations of the reduction constants, or parameters.

5.2 Number of Training Points

This calibration method requires that a total of eight reduction constants be found in order to determine all the necessary node characteristics. The set of five Equations (6 - 10) that are used to determine the constants could actually be combined until only two independent equations remain, Equations (11) and (12). Therefore we get two equations for each training point used, and would need at least four training points to effectively find the eight reduction constants.

Four training points would be sufficient for the equation set if there was no measurement error present in the calibration process. Unfortunately, there will always be some error made in the measurement process which is mostly contributed by the resolution of the camera and atmospheric interference. Because there will be measurement error, it is possible that we will need more than four training points to achieve an accurate calibration. As the number of training points increases, so too should the accuracy of the calibration. A variety of values are used in order to determine how many training points are needed for a sufficiently
accurate calibration.

The testing procedure consisted of a Monte Carlo run of simulated integration frames constructed of randomly placed stars, or training points for each of the various number of training points values. We tested values of 5, 7, 9, 11, 13 and 15 for the number of training points used in the estimation of parameter values. One thousand Monte Carlo runs for each value were simulated to obtain an average result. The reduction constant values used in this test are set to values that can be expected to be seen in a real calibration. It is important to note the parameter values used are not indicative of a particular camera’s calibration parameters but instead are examples of potentially realistic values. The values of the camera constants $P_1$, $P_2$ and $P_3$ are the values that correspond to the Rainbow fisheye lens that is used in the SkySentinel camera set up (discussed in Section 2.3). The full set of values are as follows:

\[
\begin{align*}
a_0 &= 1^\circ \\
x_0 &= 317.959 \\
y_0 &= 240.3599 \\
P_1 &= 3.24 \times 10^{-6} \\
P_2 &= 0.0054 \\
P_3 &= 0.012
\end{align*}
\]
The actual azimuth and zenith, \((a,z)\), values for each training point are determined using the set values for the reduction constants and Equations (11) and (12) or (9) and (10). With the \((x,y)\) and \((a,z)\) values for each training point we estimate the values of the reduction constants. In order to simulate an actual calibration, noise is added to the \((x,y)\) location for each training point before the solver is used to estimate the reduction constants. This noise is meant to recreate the error that could be present when attempting to find the exact \((x,y)\) location in the camera’s frame for each star. The added noise is Gaussian with a mean of zero and a standard deviation of one pixel in each direction \((x,y)\). In order to quantify how accurately the solver identified the reduction constant values, a
test grid of 261 points is constructed that fills the majority of the FOV for the simulated frame. An example of the test grid that is used for testing is shown in Figure 11. The \((a,z)\) value for each test point are found with the actual parameter values and then with the parameters that were estimated in the training stage. The residual errors between the two sets of \((a,z)\) values are recorded and can be seen in Figures 12 through 17.

![Graphs showing average azimuth and zenith residuals](image)

Figure 12: Results for 5 training points
Figure 13: Results for 7 training points
Figure 14: Results for 9 training points
Figure 15: Results for 11 training points
Figure 16: Results for 13 training points
Average Azimuth Residuals (15 training points)

Average Zenith Residuals (15 training points)

(a) Azimuth errors

(b) Zenith errors

(c) Azimuth heat map

(d) Zenith heat map

Figure 17: Results for 15 training points
The "heat maps" in Figures 12 - 17 show the variation of the average residual errors for each of the test points. The colormap uses blue for residual values on the low end of the error range and red for the larger values. The specific error value for a given test point can be deciphered from the colorbar on the right-hand side of each figure.

The azimuth and zenith error plots show the average residual value for each of the 1000 Monte Carlo simulations. To clarify, the errors associated with each of the 261 test points were averaged over the 1000 simulations and then plotted.

From Figure 18 it can be seen that azimuth residuals do not improve very much after the number of training points exceeds eleven. However, the maximum azimuth error residual does not drop below $1^\circ$ until the number of training points reaches fifteen. The average azimuth residual gradually decreases as the number of training points increases, which can be seen in Figure 18(a). The average azimuth
residual for fifteen training points is about $0.3316^\circ$.

When compared to the variations of Borovicka’s methods we can see that calibration method proposed in this thesis produces slightly higher residuals. Figure 19 shows the azimuth residuals for two variations of Borovicka’s method.

![Azimuth residuals reported by Borovicka for variations of his method](image)

(a) Residuals for method without $A$, $F$ and $C$   (b) Residuals for complete method

Figure 19: Azimuth residuals reported by Borovicka for variations of his method

The zenith residuals from Borovicka’s method can be seen in Figure 20.

Borovicka [3] reported that without the reduction constants $A$, $F$ and $C$ (seen in Equation (1)) the absolute value of the azimuth residuals are greater for azimuth angles near $90^\circ$ and $270^\circ$ than angles near $0^\circ$ (or $360^\circ$) and $180^\circ$. This phenomenon proves to be true for this calibration method. Perhaps if equation (1) were used with this method the residual values would be more consistent throughout the range of azimuth angles.

The five or six points that correspond to the highest azimuth residual errors (can be seen in Figures 12(a), 13(a), 14(a), 15(a), 16(a) and 17(a)) are the test points that are nearest to the center of the FOV. This is confirmed in the azimuth
Figure 20: Zenith residuals reported by Borovicka for variations of his method
residual heat maps, Figures 12(c), 13(c), 14(c), 15(c), 16(c) and 17(c), where it can be seen that the points with the largest error are those that are closest to the center of projection. These points are more sensitive to azimuth error because small changes in \((x,y)\) position can result in large azimuth variations when the point lies very near the center of projection.

Figures 12(a), 13(a) and 14(a), as well as Figures 12(c), 13(c) and 14(c), show much larger residual values for small azimuth values than for large azimuth values. It is expected that poor accuracies result from simulations with a small number of training points but the disparity between large and small azimuth values is hard to explain. It is possible that the particular combination of parameter values used in the simulation result in this behavior but it would require an extensive series of simulations to test this hypothesis. The reason for the behavior was not investigated further because it was not present when the number of training points used was eleven or greater. Suffice it say that at least eleven training points should be used to avoid this phenomenon.

Average zenith residuals also decrease as the number of training points increases. The average zenith residual is about \(0.3897^\circ\) when fifteen training points are used. The increase in the residual value for large zenith values is due to the fact that the fisheye lens approximation causes more distortion towards the edges of the FOV.

These residual behaviors correspond to the errors reported by Borovicka prior
to the addition of the second exponential term in Equation (2). Perhaps if a polynomial of a larger order than three (Equation (7)) were used with this calibration method then the increase in residual errors would subside as they did with the introduction of the extra exponential term in the previous method [3].

It is clear that the residuals from Borovicka’s method are smaller than those of the method proposed in this thesis. There are any number of reasons that his method appears to perform better than this one but without intimate knowledge of how the residuals were recorded we can not be sure that Borovicka’s method is preferable to this method. One major difference between the testing of the two approaches is that they used photographic plates while we used fairly inexpensive video cameras. The precision with which their measurements were made are impossible to know. We chose to add Gaussian noise with a standard deviation of one pixel to model measurement noise and this is the main cause of our inaccuracies in these simulated results. When no noise is added and we assume that measurements are perfect we see residuals that do not exceed values of $10^{-6}$°.

5.3 Parameter Variations

The next set of simulated tests were conducted similarly to the first except that now we vary the values of certain reduction constants to see how residual errors were effected. The parameters that are hardest to correctly identify with the equation solving process are $a_0$, $E$ and $\epsilon$. For this reason we decided to
experiment with a variety of values for these parameters. The lens constants $P_1$, $P_2$, $P_3$ are dependent upon the camera lens and will be set to the same values as used in the previous simulations. The values for $x_0$ and $y_0$ were also set to the same values that were previously used. The three parameters that were varied had the following ranges:

$$[\alpha_0 = [1^\circ, 85^\circ], E = [5^\circ, 355^\circ], \epsilon = [0.1^\circ, 15^\circ]]$$

The range for each parameter consisted of 30 linearly spaced values.

Each individual simulation consisted of a single parameter value being varied while the remaining seven were held constant. Eleven training points were used for each simulation because it was observed from the previous simulations that residual values were not improved much by utilizing more than eleven points. One hundred Monte Carlo simulations were run for each parameter variation and each Monte Carlo run had a new, randomly chosen, set of training points. This resulted in a total of 3000 simulated calibrations for each of the three parameters that were tested. The residual errors for each parameter variation were averaged over the entire 100 simulations for each variation of the parameter.

The following figures show the absolute average residuals for the simulations corresponding to the particular parameter that was varied.

The error values for the 261 test points used in this simulation were averaged together and then averaged again over the 100 simulations per parameter value.
Figure 21: Average error residuals vs. $a_0$ value

Figure 22: Average error residuals vs. $E$ value
This resulted in a total of 30 error points for each test, and they are related to the specific parameter value that was used.

For the parameter $a_0$, angles from $0^\circ$ to about $80^\circ$ result in reasonable residual values. This suggests that when estimating the value of $a_0$ it is important to be within the vicinity of the quadrant of the actual value for $a_0$ in order to get a good fit. This corresponds to roughly pointing the top of the camera to the West. Zenith residuals show the same result.

The $E$ azimuth residuals show a phenomenon that is not easily explained. For some reason the residual errors are quite large for values of $E$ that fall between $0^\circ$ and $100^\circ$. We assume that this can be explained somehow, and it is possible that we found a rare combination of parameters that results in this strange result. However, a variety of set parameter values were tested with the $E$ variations and all tests showed this same phenomenon. Because of the extremely high number

Figure 23: Average error residuals vs. $\epsilon$ value
of parameter combinations, it is nearly impossible to do an exhaustive test of this hypothesis. For the purposes of this research we had to accept the problem and continue the process without deciding upon a reasonable explanation.

The azimuth residuals for the rest of the $E$ range were less than $1^\circ$ and these are very acceptable error values. The zenith residuals for the entire $E$ range are below $0.5^\circ$ which is also an acceptable result. However, the zenith residual values fluctuate randomly throughout the range of $E$ values which suggests there is no correlation between zenith residuals and the value of $E$.

The epsilon residuals in Figures 23(a) and 23(b) show different behaviors. Azimuth residuals are largest when the value for $\epsilon$ is below $5^\circ$. This is because small $\epsilon$ values make the estimation of the $E$ parameter harder to determine. If the value for $E$ is inaccurate then the determination of $a$ in Equation 10 will be inaccurate.

Zenith residuals increase as the value for $\epsilon$ increases. This makes sense because the value for epsilon has a direct effect on the value for $z$, which can be seen in Figure 6. The further that the COP lies from true zenith the more likely that the value for epsilon will be misrepresented and the true zenith value will be inaccurate.

When the true value for $\epsilon$ is large the COP is relatively far from true zenith and zenith values will have larger errors. When the value for $\epsilon$ is very small the value for $E$ will be less important in the non-linear solving process and will likely
be slightly misrepresented. When values for $E$ are inaccurate the azimuth errors will increase.

5.4 Training Point Location (Quadrants)

Earlier, we arrived at the conclusion that calibration accuracy depended partly on the number of training points used. Perhaps just as important is the location of the training points. It was hypothesized that if all training points were in a certain region of the FOV then the calibration constant values would be skewed to more accurately represent test points in that particular region. If this were true then it might be possible that test points outside of that region would have larger errors than the points inside the region. Certain regions of a FOV might not contain enough stars if there is an object that obstructs the cameras view or if light pollution from a certain direction causes stars in the area to be difficult to image.

In order to test this hypothesis, we set up a series of tests that restrict the location of training points while maintaining the full coverage of test points. For the first set of tests we limit training regions to the four different polar quadrants. The calibration parameters were set to the same values as in Section 5.2. Eleven training points were used and 1000 Monte Carlo simulations were conducted for each of the locations tested. The following figures show the results for quadrants II and IV.
Figure 24: Average error residuals with training points isolated to Quadrant II

Figure 25: Average error residuals with training points isolated to Quadrant IV
The azimuth and zenith error plots show the error associated with each test point when averaged over the 1000 simulations. The heat maps also show the error associated with each test point when averaged over the 1000 simulations.

Figures 24 and 25 help to prove what was originally surmised. When training points are restricted to certain regions of the camera’s FOV, the errors for points that are outside that region are, on average, larger than those of the points inside the particular region. This result suggests that an accurate calibration can only be achieved when the training points are fairly well spread out through each quadrant.

5.5 Training Point Location (Radial)

We also hypothesized that if all the training points were in a certain radial region we would see different residual errors for test points found inside that region and for points outside the region. One reason that such a situation might arise is because stars near the horizon are difficult to image. The light from stars that are located near the horizon must pass through more of the atmosphere which cause more diffraction. This is just one possible explanation for radial occlusion of stars in an integration frame but there could be any number of reasons for this phenomenon.

This test was conducted in a similar manner as the previous test but instead of limiting training points to particular quadrants we restrict them to have certain
zenith, or radial, values. Once again, 1000 Monte Carlo simulations were run for each scenario and a total of eleven training points were used. The first test consisted of training points with zenith values of $45^\circ$ or less. The second test had training points with zenith values within the range $45^\circ - 90^\circ$.

After conducting the two tests just described it was realized that the results for the test that restricted training points to those with zenith values of $45^\circ$ to $90^\circ$ were not as expected. For this reason, a third test was conducted that restricted training point zenith values to $75^\circ - 90^\circ$.

The following figures represent the average residual errors for each of the described simulated tests.
Figure 26: Average error residuals when zenith values of training points $< 45^\circ$
Figure 27: Average error residuals when zenith values of training points $> 45^\circ$
Figure 28: Average error residuals when zenith values of training points > 75°
The results for the first test show that when training points are restricted to zenith values between $0^\circ$ and $45^\circ$ the residuals for points within that region are smaller than those outside the region. As expected, zenith residuals for test points with large zenith values are greater than those for test points with small zenith values. Figure 26(a) also shows that azimuth residuals experience the same result. That is, the azimuth errors associated with test points near the edge of the FOV are greater than those for points near the center of projection.

As mentioned earlier, the results from the second test are not what was expected. It was hypothesized that the residual values for test points with small zenith values would be greater than those with large zenith values. Figure 27(c) shows that residual values are greatest for test points with zenith values near $60^\circ$ and do not get much smaller for values larger or greater than that.

The reason that zenith residuals for test points with small zenith values are not greater than those for points with large values is that zenith residuals are naturally smaller for points with small zenith values as seen in the results from Section 5.2. The error caused by insufficient representation of training points with small zenith values is combined with the inherently small error for points with small zenith values. Similarly, the expected reduction in residuals for points with large zenith values is combined with the inherently large errors for those points.

The results seen in Figures 28(a) - 28(c) show that when training points are
Rainbow Lens Fit ([0, 45] degrees)  
(a) Zenith values < 45°

Rainbow Lens Fit ([45, 90] degrees)  
(b) Zenith values > 45°

Rainbow Lens Fit ([75, 90] degrees)  
(c) Zenith values > 75°

Figure 29: Plots of how the lens is modeled for variations in zenith training point values
isolated to the extreme outer edge of the FOV the residual values for test points will decrease as their zenith values approach the range of $75^\circ$ - $90^\circ$. Therefore we see the behavior that was originally hypothesized for this scenario.

In order to help explain why the second test did not show the expected results but the third test displayed a more appropriate behavior we have provided Figures 29(a) 29(b) and 29(c).

When zenith values between $0^\circ$ and $45^\circ$ are used for training, the lens model is fit according to Figure 29(a). Likewise when zenith values between $45^\circ$ - $90^\circ$ the lens is modeled as shown in Figure 29(b). Finally, when zenith values between $75^\circ$ - $90^\circ$ are used then the model is fit according to Figure 29(c).

Because zenith residuals are inherently small for small zenith values (see section 5.2) we can expect that large zenith values will likely have greater error than small values. Therefore when we are able to nearly fit the expected lens model for small zenith values (Figure 29(b)) we will not see a big difference between error values for any particular value. This is why residual values for the second test in this experiment were not greater for large zenith values than they were for small zenith values.

If we fit the lens model based solely on zenith points between $75^\circ$ - $90^\circ$ then we will fit the lens model for small zenith values more poorly than if we used zenith points between $45^\circ$ - $90^\circ$. The model fit shown in Figure 29(c) shows a large disparity between the expected model and the model fit for small zenith
values. The inaccurate fitting of small zenith values in this case has a greater effect on residual values than the inherent behavior of small zenith values having small residual errors (Figures 28(a) - 28(c)). This is why the third test of the experiment shows larger residuals for small zenith values and the second test does not.

5.6 Manual Calibration Results

We need to establish how well this method calibrates an actual all sky camera. Simulated calibrations do a good job of exposing the flaws and advantages of this method but it remains to be shown how these translate to real implementations. In order to perform an actual calibration we need to obtain integration frames from active cameras, or nodes.

Currently the SkySentinel network has about 30 nodes that actively communicate with the network. Some nodes are in areas with high light pollution and others have objects that obstruct their view of the sky. Nodes in areas with a lot of ambient light have a harder time imaging dim stars than those that have little to no light pollution. For this paper we chose to attempt to calibrate nodes that had a high amount of visible stars in their integration frames compared to other nodes. We also wanted a node that did not have anything obstructing its field of view. The nodes that fit that criteria were located in Parker, Kingman and Prescott, all of which are towns in Arizona.
The error plots below represent the error values associated with each of the three nodes.

Figure 30: Average error residuals for node in Parker, AZ

Figure 31: Average error residuals for node in Kingman, AZ
Figure 32: Average error residuals for node in Prescott, AZ

Table 1: Average Error Residuals: Manual Calibrations

<table>
<thead>
<tr>
<th>Node Location</th>
<th>Average Azimuth Residual</th>
<th>Average Zenith Residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parker</td>
<td>0.254°</td>
<td>0.165°</td>
</tr>
<tr>
<td>Kingman</td>
<td>0.557°</td>
<td>0.322°</td>
</tr>
<tr>
<td>Prescott</td>
<td>0.418°</td>
<td>0.285°</td>
</tr>
</tbody>
</table>

These error plots show around 30 stars for each integration frame. When frames are effectively processed and analyzed it is often possible to find 10 - 12 stars in an individual frame. Usually about 4 - 5 stars are easily visible but it might be worth the effort to optimize contrast enhancement so that there are a large number of distinguishable stars. The frames used in these calibrations were meticulously analyzed so that even when three frames are used we obtain more than enough training stars.

The azimuth errors for the Parker, AZ node are largest for values near 90° and
270° (Figure 30(a)). However, the largest residual values for the Kingman, AZ and Prescott nodes are near azimuth values of 180° and 360° (Figures 31(a) 32(a)). It is unclear why there is an inconsistency in the results. The commonality between all nodes is that their residual values do not exceed 1.8° and the average residual is near or below 0.5°, both of which are acceptable results for most applications. These results are not as accurate as those seen in [3] but should be adequate for most applications of astronomical measurements.

The zenith residuals for the Kingman, AZ and Prescott, AZ nodes increase, on average, as zenith values increase (Figures 31(b) 32(b)) which is the same result reported by [3]. The node located in Parker, AZ does not necessarily show the same behavior (Figure 30(b)). Maximum zenith residuals for the three nodes do not exceed 1.5° and their average values are near or below 0.3°.

These results suggest that these nodes have been calibrated well enough for most applications. If more accuracy is required for precise trajectory estimations or a similar application then potential improvements to the method are discussed in Section 7.

The simulated test that varied the value of $a_0$ showed that it is vital to initialize $a_0$ with a value that is within 90° of the actual value. This is not difficult to do because the orientation of the camera, with respect to the cardinal directions, can be easily approximated to within 15° or so. However, when manual calibrations are executed we notice that the value for $a_0$ can be accurately estimated even when
no initial estimation is made about the actual value. This means that even when
the positive x-axis of the camera is not pointed roughly towards the South the
value for $a_0$ can be accurately estimated. It is unclear why the results for simulated
calibrations and manual calibrations do not agree. In order to be certain that a
manual calibration has been executed correctly it is a good idea to initialize the
value of $a_0$ as a value that is close to what the expected value will be.
The goal of altering the current all-sky calibration methods was to develop a method that was easier to use and that provided a detailed description of how to implement it. The alterations include replacing the exponential equation used for lens modeling with a quadratic as well as eliminating the scale factors $A$, $F$, and $C$ used in the determination of the radial distance from the center of projection. A series of simulated tests were execute in order to extensively test the method’s ability to accurately calibrate a camera.

The replacement of Equation (2) for modeling of the lens distortion with the quadratic Equation (7) was meant to make the initialization of calibration parameters simpler. In previous methods the values for the parameters present in the lens modeling equation needed to be accurately estimated prior to initialization. The quadratic equation used for lens modeling in this method requires no prior information of the lens and the residual values that result are comparable to previous methods as stated in Section 5.2. However, it remains to be seen if residual values will improve when a larger polynomial than a quadratic is used.

The exclusion of the parameters $A$, $F$ and $C$ from Equation (6) is likely the cause of the fluctuating azimuth residuals seen in the figures in Section 5.2. The estimated values for these parameters are sensitive to their initialized values which is why they were left out of this method. While the residual values that resulted...
from the simulated tests were relatively small, it might be a useful exercise to include these parameters and assess how it affects the residuals.

The simulated tests that were run allow for a fairly exhaustive assessment of this method’s ability to calibrate all-sky cameras. It is shown that when 15 or more training points are used the residual values are below $1^\circ$. Depending upon the specific use of the camera, the residuals associated with this method are probably sufficient. However, if more precision is required then more training points can be used and the residuals should improve. The amount of FOV coverage that training points provide is very important to the accuracy of a calibration. If adequate coverage is not achieved then calibration accuracy will suffer.

It was initially a goal of this research to investigate the residuals associated with different values for the parameters $a_0$, $E$, and $\epsilon$. The results that the parameter variation simulations saw were somewhat inconclusive.

In the case of the parameter $E$, simulations showed that values between $0^\circ$ and $100^\circ$ resulted in large azimuth residuals. Unfortunately time did not allow for an exploration into this phenomenon.

The initialization of the parameter $a_0$ seemed to be an important factor in obtaining an accurate calibration when analyzing simulation results. However, manual calibrations suggest that this not the case. To be certain that an accurate calibration is achieved it is suggested that the initial value for $a_0$ is chosen to be within about $45^\circ$ of the expected value.
The simulation results for the parameter $\epsilon$ show that the residual values increase as the value of $\epsilon$ increases. This suggests that when setting up an all-sky camera it is important to try to use a level surface so as to reduce the value for $\epsilon$ as much as possible.

This new method for calibration seems to be simpler to implement than previous methods and should provide an adequate amount of accuracy. This paper provides a detailed explanation of how the method can be implemented which can be difficult to find for previous methods. There are improvements that can be made to increase accuracy and reduce residual values. For most all-sky camera needs this method is robust enough to provide the accuracy desired.
7 FUTURE WORK

This calibration method has a few shortcomings that were exposed in the results of the tests that were executed for this paper. Residual values could likely be improved by including the scale factors $A$, $F$ and $C$, present in equation 1, but that has yet to be shown. Another potential source for improvement is to use a larger order polynomial for the lens modeling equation. It is possible that the inclusion of these alterations will not affect the accuracy of the method but that will require further research.

The parameter variation tests showed some odd results and time did not allow for a full investigation into the cause of these results. Further simulated tests of parameter variations might help to explain some of the questions that have gone unanswered in terms of various parameter combinations.

The necessity for an accurate initialization of the parameter $a_0$ seems to be unimportant when manually calibrating a camera. However, the simulations that included variations of $a_0$ showed that it was important to provide a reasonable estimation of the parameter. This is another aspect of the calibration method that could be explored to provide a more complete documentation of this method.

One method that can help to determine the accuracy of an all-sky camera calibration is to use multiple calibrated nodes to attempt trajectory determinations of meteor events. If trajectory estimations are accurate then one can assume that
the calibration that was executed is accurate.
APPENDICES
APPENDIX A

CONVERSION FROM RIGHT ASCENSION AND DECLINATION

TO ALTITUDE AND ZENITH
A CONVERSION FROM RIGHT ASCENSION AND DECLINATION TO ALTITUDE AND ZENITH

In order to calculate Azimuth \( (a) \) and Zenith values \( (z) \) from Right-Ascension \( (RA) \) and Declination \( (DEC) \), the following procedure must be followed. The \( (RA \text{ & } DEC) \) value, time of measurement in UTC \( (UT) \), date of measurement, and the latitude and longitude of the measurement location are necessary to calculate the desired \( (a,z) \) values.

A.1 Converting Values to Proper Format

The first step is to convert all input values into a format that is compatible with the rest of the procedure. The values for \( (RA) \) must be in terms of hours. In order to convert \( (RA) \) from hours and minutes to decimal hours use equation (13).

\[
RA_{dec} = 15 \times (RA_{hr} + RA_{min}/60)
\] (13)

Declination \( (DEC) \) values need to be in terms of decimal degrees. To convert \( (DEC) \) values from degrees and minutes to decimal degrees use equation (14).

\[
DEC_{dec} = DEC_{deg} + DEC_{min}/60
\] (14)

Values for UTC time can be converted to decimal hours with equation (15)
\[ UT_{dec} = UT_{hr} + UT_{min}/60 \] (15)

Latitude and Longitude measurements in degrees and minutes are converted to decimal degrees with equations (16) and (17).

\[ Lat_{dec} = Lat_{deg} + Lat_{min}/60 \] (16)

\[ Long_{dec} = Long_{deg} + Long_{min}/60 \] (17)

Latitudes North are counted as positive and latitudes South are negative. Longitudes East are counted as positive and longitudes West are negative.

A.2 Number of Days

The next step is to calculate the number days since epoch J2000. Tables 2 and 3 can be used to calculate the number of days and the fraction of a day from the epoch J2000.

A.3 Local Siderial Time

The number of days is used in the following equation to calculate the Local Siderial Time (\( LST \)).

\[ LST = 100.46 + 0.985647 \times days + Long_{dec} + 15 \times UT_{dec} \] (18)
### Table 2: Days to Beginning of Month

<table>
<thead>
<tr>
<th>Month</th>
<th>Normal Year</th>
<th>Leap Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Feb</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>Mar</td>
<td>59</td>
<td>60</td>
</tr>
<tr>
<td>Apr</td>
<td>90</td>
<td>91</td>
</tr>
<tr>
<td>May</td>
<td>120</td>
<td>121</td>
</tr>
<tr>
<td>Jun</td>
<td>151</td>
<td>152</td>
</tr>
<tr>
<td>Jul</td>
<td>181</td>
<td>182</td>
</tr>
<tr>
<td>Aug</td>
<td>212</td>
<td>213</td>
</tr>
<tr>
<td>Sep</td>
<td>243</td>
<td>244</td>
</tr>
<tr>
<td>Oct</td>
<td>273</td>
<td>274</td>
</tr>
<tr>
<td>Nov</td>
<td>304</td>
<td>305</td>
</tr>
<tr>
<td>Dec</td>
<td>334</td>
<td>335</td>
</tr>
</tbody>
</table>

### A.4 Hour Angle

The third step in the process is to calculate the Hour Angle ($HA$) from the $LST$ and $(RA)$.

$$HA = LST - RA_{dec}$$  \hspace{1cm} (19)

If $HA$ is negative then add 360 to bring it into the range 0 to 360.
Table 3: Days Since J2000 to Beginning of Each Year

<table>
<thead>
<tr>
<th>Year</th>
<th>Days</th>
<th>Year</th>
<th>Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998</td>
<td>-731.5</td>
<td>2010</td>
<td>3651.5</td>
</tr>
<tr>
<td>1999</td>
<td>-366.5</td>
<td>2011</td>
<td>4016.5</td>
</tr>
<tr>
<td>2000</td>
<td>-1.5</td>
<td>2012</td>
<td>4381.5</td>
</tr>
<tr>
<td>2001</td>
<td>364.5</td>
<td>2013</td>
<td>4747.5</td>
</tr>
<tr>
<td>2002</td>
<td>729.5</td>
<td>2014</td>
<td>5112.5</td>
</tr>
<tr>
<td>2003</td>
<td>1094.5</td>
<td>2015</td>
<td>5477.5</td>
</tr>
<tr>
<td>2004</td>
<td>1459.5</td>
<td>2016</td>
<td>5842.5</td>
</tr>
<tr>
<td>2005</td>
<td>1825.5</td>
<td>2017</td>
<td>6208.5</td>
</tr>
<tr>
<td>2006</td>
<td>2190.5</td>
<td>2018</td>
<td>6573.5</td>
</tr>
<tr>
<td>2007</td>
<td>2555.5</td>
<td>2019</td>
<td>6938.5</td>
</tr>
<tr>
<td>2008</td>
<td>2920.5</td>
<td>2020</td>
<td>7303.5</td>
</tr>
<tr>
<td>2009</td>
<td>3286.5</td>
<td>2021</td>
<td>7669.5</td>
</tr>
</tbody>
</table>

A.5 HA and DEC to Altitude and Zenith

The next step is to convert \((HA)\) and \((DEC)\) to \((a,z)\). The following set of equations will complete this process.

\[
z = \sin^{-1}(\sin(DEC_{dec}) \cdot \sin(Lat_{dec}) + \cos(DEC_{dec}) \cdot \cos(Lat_{dec}) \cdot \cos(HA)) \quad (20)
\]

73
\[
a = \cos^{-1}\left(\frac{\sin(DEC_{dec}) - \sin(z) \ast \sin(Lat_{dec})}{\cos(z) \ast \cos(Lat_{dec})}\right)
\]

(21)

If \(\sin(HA)\) is negative then the azimuth value \((a)\) is correct, otherwise \(a = 360 - a\).
APPENDIX B

LENS CHARACTERIZATION EXPERIMENT

In order to obtain polynomial values that best fit the Rainbow lens used in the Skysentinel camera setup, we needed to characterize the lens. This was done by setting up an experiment that allowed us to compare angle of incidence values for a point source to their radial value from center in the image. Angle of incidence values were measured in degrees and radial values were the number of pixels from center that the image location appeared.

B.1 Setup

A sony camera identical the model used in the Skysentinel camera setup with the Rainbow lens was used in this experiment. It was fixed to a device that had a single degree of freedom that allowed it to be rotated in a horizontal direction. The rotational device was capable of measuring the incremental changes made so that it could be rotated in steps of two degrees. The device was fixed to a flat table. On the other end of the table, approximately three meters long, was a point source of light that could be imaged by the camera.

B.2 Measurement Procedure

To begin, the camera was pointed directly towards the light source so that the image of the light source was exactly centered in the camera’s image. This meant that the center of the image of the source had pixel coordinate values (320,240).
The camera was rotated two degrees horizontally to the right and the pixel coordinate location value of the image of the source was recorded. This was repeated until the camera had rotated 90 degrees. The camera was then reset to its initial location and the process was repeated in the direction to the left. The camera was then physically rotated 90° so that the vertical response of the lens was modeled exactly the same way as the horizontal response. There was almost no difference between the horizontal and vertical response so only the horizontal response was reported.

The full process was repeated three times and the pixel coordinate values for each rotation value were averaged in an attempt to reduce measurement errors. The relationship between the angle offset of the camera and the radial distance of the source’s image location allowed us to develop a relationship between angle of incidence and radial image location. The distortion of the lens could now be accurately modeled by a quadratic. The results of this experiment were presented in Section 2.3.
APPENDIX C

MATLAB CODE
C MATLAB CODE

C.1 Manual Calibration

% Manual Calibration Code
% This code will calibrate an all-sky video camera using the method
devolved for the SkySentinel Camera Network that is run by NMSU. It
takes the (x,y) pixel locations of identified stars and compares them to
the Azimuth and Zenith location, for the given camera, of the same set of
stars. Eight parameters are the result of the calibration. The
parameters describe the rotation and offset of the camera as well as the
distortion caused by the camera’s lens. The lens distortion equation is
a quadratic that is meant to fit a fisheye, or wide angle, lens.

NOTE: It is recommended that at least 15 stars are used for calibration
and that they provide full coverage of the FOV.

% INPUTS:
% time - string that contains the date of the capture as well as the exact
% UTC time. Format: ‘yyyy/mm/dd hour:minute:second’
% Lat - Latitude of the camera in decimal degrees. North +, South -
% Long - Longitude of the camera in decimal degrees. East +, West -
% X - vector that consists of the x-coordinates for the stars in the
camera’s frame. This vector is input by entering each value
individually in the form x1 = [], x2 = [], ...
% Y - vector that consists of the y-coordinates for the stars in the
camera’s frame. This vector is input by entering each value
individually in the form y1 = [], y2 = [], ...
% RA - vector that consists of the right ascension values for each star
that was found in the camera’s frame. There must be an RA value for
each corresponding (x,y) pair and they must be listed in the same
order as the x and y coordinates. If multiple frames are used for a calibration then each set of RA values for a given frame must be organized into separate vectors for the RA/Dec -> (Az, Zen) conversion.

DEC - vector that consists of the declination values for each star that was found in the camera’s frame. There must be a DEC value for each corresponding (x,y) pair and they must be listed in the same order as the x and y coordinates. If multiple frames are used for a calibration then each set of DEC values for a given frame must be organized into separate vectors for the RA/Dec -> (Az, Zen) conversion.

OUTPUTS:

a0 - describes angle between the camera’s positive x-axis and South.
a0 - the x-coordinate location of the offset from the center of the camera’s projection to true zenith.
y0 - the y-coordinate location of the offset from the center of the camera’s projection to true zenith.
Q, V, S - The parameters that describe the lens distortion caused by the fisheye lens. They fit the quadratic equation in the form:
Qr^2 + Vr + S

epsl - the zenith distance from the camera’s (x,y) origin (320, 240 for a 640x480 frame) to the camera’s center of projection.
E - the azimuth angle from the camera’s (x,y) origin (320, 240 for a 640x480 frame) to the camera’s center of projection.

NOTE: This code will produce Azimuth values that correspond to South as 0 degrees (or 360 degrees), West as 90 degrees, North as 180 degrees, and East as 270 degrees. Often, the convention is to use North as 0 degrees, East as 90 degrees, South as 180 degrees, and West as 270 degrees. If the former convention is to be used then
% the azimuth outputs should be wrapped accordingly.

clear; clc

% Input frame dates and times

time1 = '2012/05/20 4:25:12';
time2 = '2012/05/20 5:25:00';
time3 = '2012/05/20 6:24:47';

% Input Latitude and Longitude for node in decimal degrees

Lat = 34.9506; % Fact site
Long = -106.45963; % Fact site

% Enter X and Y values for star positions

% Frame one
x1 = 490; y1 = 66; % Capella
x2 = 355; y2 = 359; % Spica
x3 = 294; y3 = 302; % Arcturus
x4 = 137; y4 = 227; % Vega
x5 = 440; y5 = 254; % Regulus

% Frame two
x6 = 399; y6 = 338; % Spica
x7 = 334; y7 = 287; % Arcturus
x8 = 165; y8 = 243; % Vega
x9 = 471; y9 = 223; % Regulus
x10 = 294; y10 = 438; % Antares

% Frame three
x11 = 439; y11 = 313; % Spica
$\text{Organize all points into a single vector for } X \text{ and a single vector for } Y$

$X = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17}]$;

$Y = [y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}, y_{14}, y_{15}, y_{16}, y_{17}]$;

$\text{Plot } (X,Y) \text{ positions for recorded stars}$

\begin{verbatim}
figure()
[xc,yc,zc]=cylinder(250,100);
plot(xc(:,1)+320,yc(:,1)+240);
axis equal
hold on
scatter(320,240,'+');
scatter(X,Y,'x');
hold off
\end{verbatim}

$\text{RA and Dec for stars}$

$\text{Frame one}$

ra1 = 5.2781; dec1 = 45.9833; $\text{Capella}$

ra2 = 13.4199; dec2 = -11.16139; $\text{Spica}$

ra3 = 14.261; dec3 = 19.1825; $\text{Arcturus}$

ra4 = 18.6156; dec4 = 38.7836; $\text{Vega}$

ra5 = 10.1395; dec5 = 11.9672; $\text{Regulus}$

$\text{Frame two}$

ra6 = 13.4199; dec6 = -11.16139; $\text{Spica}$
ra7 = 14.261; dec7 = 19.1825; % Arcturus
ra8 = 18.6156; dec8 = 38.7836; % Vega
ra9 = 10.1395; dec9 = 11.9672; % Regulus
ra10 = 16.4901; dec10 = -26.4319; % Antares

% Frame three
ra11 = 13.4199; dec11 = -11.16139; % Spica
ra12 = 14.261; dec12 = 19.1825; % Arcturus
ra13 = 18.6156; dec13 = 38.7836; % Vega
ra14 = 16.4901; dec14 = -26.4319; % Antares
ra15 = 19.8464; dec15 = 8.8683; % Altair
ra16 = 20.6905; dec16 = 45.2803; % Deneb
ra17 = 10.1395; dec17 = 11.9672; % Regulus

% Organize points from frame one into a vector for RA and a vector for Dec
% These vectors must be manually changed so that the correct stars correspond to their frame/time.
RA1 = [ra1; ra2; ra3; ra4; ra5].*(360/24);
DEC1 = [dec1; dec2; dec3; dec4; dec5];
% Convert RA/Dec to (Az, Alt) for first frame
[Az1, Alt1] = RaDec2AzEl(RA1, DEC1, Lat, Long, time1);
Zen1 = 90 - Alt1; % convert altitude to zenith

% Organize points from frame two into a vector for RA and a vector for Dec
% These vectors must be manually changed so that the correct stars correspond to their frame/time.
RA2 = [ra6; ra7; ra8; ra9; ra10].*(360/24);
DEC2 = [dec6; dec7; dec8; dec9; dec10];
% Convert RA/Dec to (Az, Alt) for second frame
[Az2, Alt2] = RaDec2AzEl(RA2, DEC2, Lat, Long, time2);
Zen2 = 90-Alt2; % convert Altitude to zenith

% Organize points from frame three into a vector for RA and a vector for Dec
% These vectors must be manually changed so that the correct stars
% correspond to their frame/time.
RA3 = [ra11;ra12;ra13;ra14;ra15;ra16;ra17].*(360/24);
DEC3 = [dec11;dec12;dec13;dec14;dec15;dec16;dec17];

% Convert RA/Dec to (Az, Alt) for third frame
[Az3,Alt3] = RaDec2AzEl(RA3,DEC3,Lat,Long,time3);
Zen3 = 90-Alt3; % convert Altitude to zenith

% Combine the coordinates from all frames
Zen = [Zen1;Zen2;Zen3];
Az = [Az1;Az2;Az3];

% Inversion of Azimuth to fit to current (Az, Zen) representation
Az = -Az + 360;

% Wrap negative azimuth values
select = Az < 0;
Az(select) = Az(select)+360;

% Plot actual (Az, Zen) values
figure()
polarmy(Az*(pi/180),Zen,'r+',90,3)

% Convert Azimuth and Zenith from radians to degrees
Az = Az*(pi/180);
Zen = Zen*(pi/180);
% NOTE:
% The Azimuth values found here will use the convention of South as 0
% degrees. If the alternate convention, North as 0 degrees is desired
% then subtract pi from all azimuth values and then add 2*pi degrees to
% any negative azimuth values that result.

data = [X;Y;Az';Zen']; % Data vector contains all (X,Y) and (Az, Zen) pairs

% initialize calibration constants
sa0 = 0*(pi/80); % You might need to adjust this initial parameter if the
% positive x-axis of the frame is not pointed within 45
% degrees of South.
sx0 = 320;
sy0 = 240;
sQ = 0;
sV = 0;
sS = 0;
sepsl = 0;
sE = 0;

% Use solver to find parameter values
initial_conditions = [sa0,sx0,sy0,sQ,sV,sS,sepsl,sE];
[unknowns] = fsolve(@(unknowns)full_solver(unknowns,data),initial_conditions);
a0 = unknowns(1);
x0 = unknowns(2);
y0 = unknowns(3);
Q = unknowns(4);
V = unknowns(5);
S = unknowns(6);
epsl = unknowns(7);
E = unknowns(8);

% Use solved parameters to find (Az, Zen) values
r = sqrt((X-x0).^2+(Y-y0).^2);
u = Q.*r.^2 + V.*r + S;
b = a0 - E + atan2((Y-y0),(X-x0));

% Solved Zenith value
z_solved = acos(cos(u).*cos(epsl) - sin(epsl).*sin(u).*cos(b));
sina_E=(sin(b).*sin(u))./sin(z_solved);
a_E=asin(sina_E);

cosa_E=(cos(u)-cos(epsl).*cos(z_solved))./(sin(epsl).*sin(z_solved));
select=cosa_E<0;
a_E(select)=pi-a_E(select);
a_solved = a_E+E;% Solved Azimuth value

% Fix negative azimuth values by adding 2*pi
select = a_solved < 0;
a_solved(select) = a_solved(select) + 2*pi;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% NOTE:
% The Azimuth values found here will use the convention of South as 0
% degrees. If the alternate convention, North as 0 degrees is desired
% then subtract pi from all azimuth values and then add 2*pi degrees to
% any negative azimuth values that result.
az_error = abs(Az - a_solved')*(180/pi); % Residual between actual Azimuth and solved Azimuth

zen_error = abs(Zen - z_solved')*(180/pi); % Residual between actual zenith and solved zenith

figure(3)
polarmy(a_solved,z_solved*180/pi,'*','90,3)
C.2 Number of Training Points Simulated Calibration

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulated calibration to test number of training points
% This code is set up to run 1000 simulated calibrations. The number of
% training points used can be varied.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;clc

error_val = 0;
PARAMETER_ERRORS = []; PARAMETER_SOLVED = [];
AZIMUTH_ERRORS = []; ZENITH_ERRORS = [];
POINTS_SOLVED = []; Avg_azimuth_err = [];
Avg_zenith_err = [];

[X_test,Y_test] = grid_gen(320,240,230,25); % Test Grid
X_test(125) = []; % Remove center point (comment out if center point is to be
Y_test(125) = []; % included

est_x0 = 320;
est_y0 = 240;

num_points =5; % Change this value to alter the number of training points used

rc=200;

% Run simulations 1000 times to obtain average residual values
for Its = 1:1000
X_train = []; Y_train = [];
X_train_noise = []; Y_train_noise = [];
for k = 1: num_points
    [x y] = cirrdnPJ(est_x0, est_y0, rc);
    x_noise = x + 0.5 * randn(1);
    y_noise = y + 0.5 * randn(1);
    X_train = [X_train, x]; Y_train = [Y_train, y];
    X_train_noise = [X_train_noise, x_noise];
    Y_train_noise = [Y_train_noise, y_noise];
end

% Range of parameter values can be used. Adjust the values below to change
% Parameter values.

a0_range = linspace(1, 90, 9);
E_range = [10:40:330];
epsl_range = linspace(0.5, 20, 9);

% Set initial distortion parameters
actual_a0 = a0_range(1) * (pi / 180); % Change the value in parenthesis to use
actual_E = E_range(5) * (pi / 180); % different parameter values
actual_epsl = epsl_range(2) * (pi / 180);

% These values cannot be changed
actual_x0 = est_x0 + 127 * sin(actual_epsl / pi) * cos(actual_E);
actual_y0 = est_y0 + 127 * sin(actual_epsl / pi) * sin(actual_E);

% Determine necessary values
% Train
r = sqrt((X_train-actual_x0).^2+(Y_train-actual_y0).^2);
actual_V = 0.0054; actual_S = 0.0120; actual_Q = 3.24*10^-6;

u = actual_Q*r.^2 + actual_V*r + actual_S;
b = actual_a0 - actual_E + atan2((Y_train-actual_y0),(X_train-actual_x0));

z_train = acos(cos(u).*cos(actual_epsl) - sin(actual_epsl).*sin(u).*cos(b));
sina_E=(sin(b).*sin(u))./sin(z_train);
a_E=asin(sina_E);

cosa_E=(cos(u)-cos(actual_epsl).*cos(z_train))./(sin(actual_epsl).*sin(z_train));

select=cosa_E<0;
a_E(select)=pi-a_E(select);
a_train = a_E+actual_E;
select = a_train > (2*pi);
a_train(select) = a_train(select)-(2*pi);

% Test:

r = sqrt((X_test-actual_x0).^2+(Y_test-actual_y0).^2);

actual_V = 0.0054; actual_S = 0.0120; actual_Q = 3.24*10^-6;

u = actual_Q*r.^2 + actual_V*r + actual_S;
b = actual_a0 - actual_E + atan2((Y_test-actual_y0),(X_test-actual_x0));

z_test = acos(cos(u).*cos(actual_epsl) - sin(actual_epsl).*sin(u).*cos(b));
sina_E=(sin(b).*sin(u))./sin(z_test);
a_E=asin(sina_E);

cosa_E=(cos(u)-cos(actual_epsl).*cos(z_test))./(sin(actual_epsl).*sin(z_test));

select=cosa_E<0;
a_E(select)=pi-a_E(select);
a_test = a_E+actual_E;
select = a_test > (2*pi);
a_test(select) = a_test(select)-(2*pi);
% This controls whether or not to use noise in training process

data = [X_train_noise;Y_train_noise;a_train;z_train];

% data = [X_train;Y_train;a_train;z_train];

% Begin determination of transformation parameters

sa0 = 0; % init_unk(1);
sx0 = 320; % init_unk(2);
sy0 = 240; % init_unk(3);
sQ = 0;
sS = 0;
sV = 0;
sepsl = 0;
sE = 0;

initial_conditions = [sa0,sx0,sy0,sQ,sS,sV,sepsl,sE];
[unknowns] = fsolve(@(unknowns)full_solver(unknowns,data),initial_conditions);

a0 = unknowns(1);
x0 = unknowns(2);
y0 = unknowns(3);
Q = unknowns(4);
V = unknowns(5);
S = unknowns(6);
sepsl = unknowns(7);
E = unknowns(8);
% Conditions to ensure that all parameter values will be positive

if \( \varepsilon_1 > \pi \)
    \( \varepsilon_1 = -1*(\varepsilon_1 - 2*\pi) \);
    \( E = E + \pi \);
    flag = 1;
elseif \( \varepsilon_1 < 0 \)
    \( \varepsilon_1 = -\varepsilon_1 \);
    \( E = E + \pi \);
    flag = 1;
end

if flag == 0

if \( E > 2*\pi \)
    \( E = E - 2*\pi \);
elseif \( E < 0 \)
    \( E = E + 2*\pi \);
end

end

if \( a_0 > \pi \) && \( a_0 < 2*\pi \)
    \( a_0 = a_0 - \pi \);
elseif \( a_0 < 0 \) && \( a_0 > -\pi \)
    \( a_0 = a_0 + \pi \);
elseif \( a_0 < -\pi \) && \( a_0 > -2*\pi \)
    \( a_0 = a_0 + 2*\pi \);
elseif \( a_0 > 2*\pi \) && \( a_0 < 3*\pi \)
    \( a_0 = a_0 - 2*\pi \);
elseif \( a_0 > 3*\pi \)
    \( a_0 = a_0 - 3*\pi \);
end

if \( V < 0 \)
    \( V = -V \);
if S < 0
    S = -S;
end

% Determine azimuth and zenith values with estimated parameters from the solver
r = sqrt((X_test-x0).^2+(Y_test-y0).^2);
u = Q.*r.^2 + V.*r + S;
b = a0 - E + atan2((Y_test-y0),(X_test-x0));
z_solved = acos(cos(u).*cos(eps1) - sin(eps1).*sin(u).*cos(b));
sina_E=(sin(b).*sin(u))./sin(z_solved);
a_E=asin(sina_E);
cosa_E=(cos(u)-cos(eps1).*cos(z_solved))./(sin(eps1).*sin(z_solved));
select=cosa_E<0;
a_E(select)=pi-a_E(select);
a_solved = a_E+E;
select = a_solved > (2*pi);
a_solved(select) = a_solved(select)-(2*pi);

% Error values for points and parameters
param_errors = [abs(a0-actual_a0), abs(x0-actual_x0), abs(y0-actual_y0),...
    abs(Q-actual_Q), abs(V-actual_V), abs(S-actual_S)...
    ,abs(E-actual_E), abs(eps1-actual_eps1)];
azimuth_errors = [abs(a_test-a_solved)];
a_solved_mod = a_solved; % these three lines prevent wrap around issues
select = azimuth_errors > pi;
a_solved_mod(select) = a_solved_mod(select) + 2*pi;

azimuth_errors = [abs(a_test-a_solved_mod)];
zenith_errors = [abs(z_test-z_solved)];

PARAMETER_SOLVED = [PARAMETER_SOLVED;a0,x0,y0,Q,V,S,eps1];
PARAMETER_ERRORS = [PARAMETER_ERRORS;param_errors];

points = {a_test;z_test};
POINTS_SOLVED = [POINTS_SOLVED,points];
az_errs = [(180/pi).*azimuth_errors];
zen_errs = [(180/pi).*zenith_errors];
AZIMUTH_ERRORS = [AZIMUTH_ERRORS,az_errs];
ZENITH_ERRORS = [ZENITH_ERRORS,zen_errs];
Avg_azimuth_err = [Avg_azimuth_err;mean(azimuth_errors)];
Avg_zenith_err = [Avg_zenith_err;mean(zenith_errors)];

end

% PLOTS

% some labels may need to be changed if number of training points has been changed

% Average azimuth errors

x_az = a_test*(180/pi);
y_AZ = mean(AZIMUTH_ERRORS);
figure();
plot(x_az,y_AZ,'o')
set(gca,'FontSize',16)
Average Azimuth Residuals (Fifteen training points w/ noise)

Average Zenith Residuals (Fifteen training points w/ noise)

Average Zenith Errors

Azimuth Heat Map

Zenith Heat Map
hold on
axis equal
scatter(X_test,Y_test,15,y_Z,'filled')
set(gca,'FontSize',16)
title('Average Zenith Residuals (Fifteen training points w/ noise)');
colorbar
C.3 Quadrant Location Simulated Calibration

Simulated calibration that tests the location of training points with the four polar quadrants. 1000 different simulation are ran to obtain average residual errors.

```matlab
% Clear and clear
error_val = 0;
PARAMETER_ERRORS = [];
PARAMETER_SOLVED = [];
AZIMUTH_ERRORS = [];
ZENITH_ERRORS = [];
POINTS_SOLVED = [];
Avg_azimuth_err = [];
Avg_zenith_err = [];

[X_test,Y_test] = grid_gen(320,240,230,25); % Test grid
X_test(125) = [];
% Remove center point (comment out if center point is to be included)
Y_test(125) = [];

est_x0 = 320;
est_y0 = 240;

num_points = 11; % Number of training points to use
quad = [4]; % Quadrant to place training points in. The value can be
% changed to a vector if multiple quadrants need to be used.
% For example quad = [1,2,3] will place points in the three
```
rc=200;

for Its = 1:1000
    X_train = []; Y_train = [];
    X_train_noise = []; Y_train_noise = [];
    for k = 1:num_points
        [x y]=cirrdnPJ_loc(est_x0, est_y0, rc, quad);
        x_noise = x + 0.5*randn(1);
        y_noise = y + 0.5*randn(1);
        X_train = [X_train, x]; Y_train = [Y_train, y]; % Training vector without noise
        X_train_noise = [X_train_noise, x_noise]; % Training vector with noise
        Y_train_noise = [Y_train_noise, y_noise];
    end

% Range of parameter values can be used. Adjust the values below to change
% Parameter values.

a0_range = linspace(1, 90, 9);
E_range = [10:40:330];
epsl_range = linspace(0.5, 20, 9);

% Set initial distortion parameters
actual_a0 = a0_range(1)*(pi/180); % Change the value in parenthesis to use
actual_E = E_range(5)*(pi/180); % different parameter values
actual_epsl = epsl_range(2)*(pi/180);

% These values cannot be changed
actual_{x0} = est_{x0} + 127 \sin(actual_{\epsilon}/\pi)\cos(actual_{E});
actual_{y0} = est_{y0} + 127 \sin(actual_{\epsilon}/\pi)\sin(actual_{E});

% Determine necessary values

% Train

r = sqrt((X_train-actual_x0).^2+(Y_train-actual_y0).^2);
actual_V = 0.0054; actual_S = 0.0120; actual_Q = 3.24*10^{-6};
u = actual_Q.*r.^2 + actual_V.*r + actual_S;
b = actual_{a0} - actual_{E} + atan2((Y_train-actual_y0),(X_train-actual_x0));
z_train = acos(cos(u).*cos(actual_{\epsilon}) - sin(actual_{\epsilon}).*sin(u).*cos(b));
sina_E = (sin(b).*sin(u))./sin(z_train);
a_E = asin(sina_E);

cosa_E = (cos(u)-cos(actual_{\epsilon})).*cos(z_train))./(sin(actual_{\epsilon}).*sin(z_train));
select = cosa_E < 0;
a_E(select) = pi-a_E(select);
a_train = a_E+actual_{E};
select = a_train > (2*pi);
a_train(select) = a_train(select)-(2*pi);

% Test

r = sqrt((X_test-actual_x0).^2+(Y_test-actual_y0).^2);
actual_V = 0.0054; actual_S = 0.0120; actual_Q = 3.24*10^{-6};
u = actual_Q.*r.^2 + actual_V.*r + actual_S;
b = actual_{a0} - actual_{E} + atan2((Y_test-actual_y0),(X_test-actual_x0));
z_test = acos(cos(u).*cos(actual_{\epsilon}) - sin(actual_{\epsilon}).*sin(u).*cos(b));
sina_E = (sin(b).*sin(u))./sin(z_test);
a_E = asin(sina_E);

cosa_E = (cos(u)-cos(actual_{\epsilon})).*cos(z_test))./(sin(actual_{\epsilon}).*sin(z_test));
select = cosa_E < 0;
a_E(select) = \pi - a_E(select);
a_test = a_E + actual_E;
select = a_test > (2*\pi);
a_test(select) = a_test(select) - (2*\pi);

This controls whether or not to use noise in training process

data = [X_train_noise; Y_train_noise; a_train; z_train];

Begin determination of transformation parameters

sa0 = 0; % init_unk(1);
sx0 = 320; % init_unk(2);
sy0 = 240; % init_unk(3);
sQ = 0;
sS = 0;
sV = 0;
sepsl = 0;
sE = 0;

initial_conditions = [sa0, sx0, sy0, sQ, sS, sV, sepsl, sE];

[unknowns] = fsolve(@(unknowns) full_solver(unknowns, data), initial_conditions);

a0 = unknowns(1);
x0 = unknowns(2);
y0 = unknowns(3);
Q = unknowns(4);
V = unknowns(5);
S = unknowns(6);
epsl = unknowns(7);
E = unknowns(8);

%conditions to ensure that all parameter values will be positive
% if epsl > pi
%   epsl = -1*(epsl - 2*pi);
%   E = E + pi;
%   flag = 1;
% elseif epsl<0
%   epsl = -epsl;
%   E = E+pi;
%   flag = 1;
% end
% if flag == 0
% if E > 2*pi
%   E = E - 2*pi;
% elseif E<0
%   E = E + 2*pi;
% end
% end
% if a0 > pi && a0<2*pi
%   a0 = a0 -pi;
% elseif a0 <0 && a0>=-pi
%   a0 = a0 + pi;
% elseif a0 < -pi && a0 > -2*pi
%   a0 = a0 + 2*pi;
% elseif a0 > 2*pi && a0 < 3*pi
%   a0 = a0-2*pi;
% elseif a0 > 3*pi
%   a0 = a0-3*pi;
% end
% Determine azimuth and zenith values with estimated parameters from the solver

\[ r = \sqrt{(X_{\text{test}} - x_0)^2 + (Y_{\text{test}} - y_0)^2}; \]
\[ u = Q \cdot r^2 + V \cdot r + S; \]
\[ b = a_0 - E \cdot \text{atan2}(Y_{\text{test}} - y_0, X_{\text{test}} - x_0); \]
\[ z_{\text{solved}} = \text{acos}(\cos(u) \cdot \cos(\epsilon) - \sin(\epsilon) \cdot \sin(u) \cdot \cos(b)); \]
\[ \text{sina}_{E} = (\sin(b) \cdot \sin(u)) / \sin(z_{\text{solved}}); \]
\[ a_{E} = \text{asin}(\text{sina}_{E}); \]
\[ \text{cosa}_{E} = (\cos(u) - \cos(\epsilon) \cdot \cos(z_{\text{solved}})) / (\sin(\epsilon) \cdot \sin(z_{\text{solved}})); \]
\[ \text{select} = \text{cosa}_{E} < 0; \]
\[ a_{E}(\text{select}) = \pi - a_{E}(\text{select}); \]
\[ a_{\text{solved}} = a_{E} + E; \]
\[ \text{select} = a_{\text{solved}} > (2\pi); \]
\[ a_{\text{solved}}(\text{select}) = a_{\text{solved}}(\text{select}) - (2\pi); \]
\[ \text{select} = a_{\text{solved}} < 0; \]
\[ a_{\text{solved}}(\text{select}) = a_{\text{solved}}(\text{select}) + 2\pi; \]

% Error values for points and parameters

\[ \text{param}_{\text{errors}} = [\text{abs}(a_0 - \text{actual}_{a_0}), \text{abs}(x_0 - \text{actual}_{x_0}), \text{abs}(y_0 - \text{actual}_{y_0}), \ldots \]
\[ \text{abs}(Q - \text{actual}_{Q}), \text{abs}(V - \text{actual}_{V}), \text{abs}(S - \text{actual}_{S}), \ldots \]
\[ ,\text{abs}(E - \text{actual}_{E}), \text{abs}(\epsilon - \text{actual}_{\epsilon})]; \]
\[ \text{azimuth}_{\text{errors}} = [\text{abs}(a_{\text{test}} - a_{\text{solved}})]; \]
\[ a_{\text{solved}}_{\text{mod}} = a_{\text{solved}}; \] % These three lines prevent wrap around issues
\[ \text{select} = \text{azimuth}_{\text{errors}} > \pi; \]
a_solved_mod(select) = a_solved_mod(select) + 2*pi;

azimuth_errors = [abs(a_test-a_solved_mod)];
zenith_errors = [abs(z_test-z_solved)];

PARAMETER_SOLVED = [PARAMETER_SOLVED;a0,x0,y0,Q,V,S,E,eps1];
PARAMETER_ERRORS = [PARAMETER_ERRORS;param_errors];

points = {a_test;z_test};
POINTS_SOLVED = [POINTS_SOLVED,points];
az_errs = [(180/pi).*azimuth_errors];
zen_errs = [(180/pi).*zenith_errors];
AZIMUTH_ERRORS = [AZIMUTH_ERRORS;az_errs];
ZENITH_ERRORS = [ZENITH_ERRORS;zen_errs];
Avg_azimuth_err = [Avg_azimuth_err;mean(azimuth_errors)];
Avg_zenith_err = [Avg_zenith_err;mean(zenith_errors)];

end

% PLOTS
% Some labels may need to be changed if number of training points has been changed

% Average azimuth errors
y_az = mean(AZIMUTH_ERRORS);
x_az = a_test*(180/pi);
figure()
plot(x_az,y_az,'o')
set(gca,'FontSize',16)
title('Azimuth residuals (Quad IV)');
xlabel('Azimuth value (degrees)');
ylabel('|residual| (degrees)');

%Average zenith errors
y_z = mean(ZENITH_ERRORS);
x_z = z_test*(180/pi);
figure()
plot(x_z,y_z,'o')
set(gca,'FontSize',16)
title('Zenith residuals (Quad IV)');
xlabel('Zenith distance (degrees)');
ylabel('|residual| (degrees)');

%Azimuth heat map
rc=240;
figure()
[x,y,z] = cylinder(rc,200);
plot(x(1,:)+est_x0,y(1,:)+est_y0,'k')
hold on
axis equal
scatter(X_test,Y_test,15,y_az,'filled')
set(gca,'FontSize',16)
title('Average Azimuth Residuals (Quad IV)');
colorbar

%Zenith heat map
rc=240;
figure()
[x,y,z] = cylinder(rc,200);
plot(x(1,:),est_x0,y(1,:),est_y0,'k')
hold on
axis equal
scatter(X_test,Y_test,15,y_z,'filled')
set(gca,'FontSize',16)
title('Average Zenith Residuals (Quad IV)');
colorbar
C.4 Radial Location Simulated Calibration

```
% Simulated calibration that tests the location of training points with the 
% zenith distance. 1000 different simulations are ran to obtain average 
% residual errors.

clear;clc

error_val = 0;
PARAMETER_ERRORS = [];
PARAMETER_SOLVED = [];
AZIMUTH_ERRORS = [];
ZENITH_ERRORS = [];
POINTS_SOLVED = [];
Avg_azimuth_err = [];
Avg_zenith_err = [];

[X_test,Y_test] = grid_gen(320,240,230,25); % Test Grid
X_test(125) = [];% Remove center point (comment out if center point is to be 
Y_test(125) = [];% included

est_x0 = 320;
est_y0 = 240;

num_points =11; % Number of training points
rad = 3; % This controls which region to plot training points.

% A value of one will put all training points in the 0–45 degree
```
% range. A value of two will result in 45–90 degrees. A value of
% three will result in 75–90 degrees.

rc=240;
for its = 1:1000
    X_train = []; Y_train = [];
    X_train_noise = []; Y_train_noise = [];
    for k = 1:num_points
        [x, y] = cirrdnPJ_rad(est_x0, est_y0, rc, rad);
        x_noise = x + 0.5*randn(1);
        y_noise = y + 0.5*randn(1);
        X_train = [X_train, x]; Y_train = [Y_train, y]; % training points w/out noise
        X_train_noise = [X_train_noise, x_noise]; % Training points with noise
        Y_train_noise = [Y_train_noise, y_noise];
    end

% Range of parameter values can be used. Adjust the values below to change
% Parameters values.

a0_range = linspace(1, 90, 9);
E_range = [10:40:330];
epsl_range = linspace(0.5, 20, 9);

% Set initial distortion parameters
actual_a0 = a0_range(1)*(pi/180); % Change the value in the parenthesis
actual_E = E_range(5)*(pi/180); % for different parameter values
actual_epsl = epsl_range(2)*(pi/180);

% These values cannot be changed
actual_x0 = est_x0 + 127*sin(actual_epsl/pi)*cos(actual_E);
\[
\text{actual}_y0 = \text{est}_y0 + 127 \times \sin(\text{actual}_\varepsilon / \pi) \times \sin(\text{actual}_E);
\]

\textbf{Determine necessary values}

\textit{Train}

\[
r = \sqrt{(X_{\text{train}} - \text{actual}_x0)^2 + (Y_{\text{train}} - \text{actual}_y0)^2};
\]

\[
\text{actual}_V = 0.0054; \quad \text{actual}_S = 0.0120; \quad \text{actual}_Q = 3.24 \times 10^{-6};
\]

\[
u = \text{actual}_Q \times r^2 + \text{actual}_V \times r + \text{actual}_S;
\]

\[
b = \text{actual}_a0 - \text{actual}_E + \text{atan2}(Y_{\text{train}} - \text{actual}_y0, X_{\text{train}} - \text{actual}_x0);
\]

\[
z_{\text{train}} = \cos(a) \times \cos(\text{actual}_\varepsilon) - \sin(\text{actual}_\varepsilon) \times \sin(a) \times \cos(b));
\]

\[
s\sin_\varepsilon = (\sin(b) \times \sin(a)) / \sin(z_{\text{train}});\]

\[
a_\varepsilon = \cos(a) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(a) = \cos(u) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b) \times \sin(a) / \sin(z_{\text{train}});\]

\[
a_\varepsilon = \cos(a);\]

\[
\cos(z_{\text{train}});\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b);\]

\[
\sin(a) = \sin(z_{\text{train}});\]

\[
\cos(a) = \cos(u) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b);\]

\[
\sin(a) = \sin(z_{\text{train}});\]

\[
\cos(a) = \cos(u) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b) \times \sin(a) / \sin(z_{\text{train}});\]

\[
a_\varepsilon = \cos(a) \times \cos(z_{\text{train}}) / \sin(z_{\text{train}});\]

\[
\cos(a) = \cos(u) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b);\]

\[
\sin(a) = \sin(z_{\text{train}});\]

\[
\cos(a) = \cos(u) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b);\]

\[
\sin(a) = \sin(z_{\text{train}});\]

\[
\cos(a) = \cos(u) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b);\]

\[
\sin(a) = \sin(z_{\text{train}});\]

\[
\cos(a) = \cos(u) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b);\]

\[
\sin(a) = \sin(z_{\text{train}});\]

\[
\cos(a) = \cos(u) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b);\]

\[
\sin(a) = \sin(z_{\text{train}});\]

\[
\cos(a) = \cos(u) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b);\]

\[
\sin(a) = \sin(z_{\text{train}});\]

\[
\cos(a) = \cos(u) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b);\]

\[
\sin(a) = \sin(z_{\text{train}});\]

\[
\cos(a) = \cos(u) - \cos(\text{actual}_\varepsilon);\]

\[
\cos(z_{\text{train}});\]

\[
\sin(a) = \sin(b);\]

\[
\sin(a) = \sin(z_{\text{train}});\]
a_E(select) = \pi - a_E(select);

a_test = a_E + actual_E;

select = a_test > (2 \cdot \pi);

a_test(select) = a_test(select) - (2 \cdot \pi);

% This controls whether or not to use noise in training process

data = [X_train; Y_train; a_train; z_train];

data = [X_train_noise; Y_train_noise; a_train; z_train];

% Begin determination of transformation parameters

sa0 = 0; \textit{init}_unk(1);
sx0 = 320; \textit{init}_unk(2);
sy0 = 240; \textit{init}_unk(3);
sQ = 0;
sS = 0;
sV = 0;
sepsl = 0;
sE = 0;

initial_conditions = [sa0, sx0, sy0, sQ, sS, sV, sepsl, sE];

[unknowns] = \texttt{fsolve(@(unknowns)fullSolver(unknowns, data), initial_conditions)};

a0 = unknowns(1);
x0 = unknowns(2);
y0 = unknowns(3);
Q = unknowns(4);
V = unknowns(5);
S = unknowns(6);
epsl = unknowns(7);
E = unknowns(8);

keyboard

% conditions to ensure that all parameter values will be positive

if epsl > pi
    epsl = -1*(epsl - 2*pi);
    E = E + pi;
    flag = 1;
elseif eps1<0
    epsl = -epsl;
    E = E+pi;
    flag = 1;
end

if flag == 0
    if E > 2*pi
        E = E - 2*pi;
    elseif E<0
        E = E + 2*pi;
    end
end

if a0 > pi && a0<2*pi
    a0 = a0 -pi;
elseif a0 <0 && a0>-pi
    a0 = a0 +pi;
elseif a0 <-pi && a0 > -2*pi
    a0 = a0 + 2*pi;
elseif a0 > 2*pi && a0 < 3*pi
    a0 = a0-2*pi;
elseif a0 > 3*pi
    a0 = a0-3*pi;
end

% Determine azimuth and zenith values with estimated parameters from the solver

r = sqrt((X_test-x0).^2+(Y_test-y0).^2);
u = Q.*r.^2 + V.*r + S;
b = a0 - E + atan2((Y_test-y0),(X_test-x0));
z_solved = acos(cos(u).*cos(epsl) - sin(epsl).*sin(u).*cos(b));
sina_E=(sin(b).*sin(u))./sin(z_solved);
a_E=asin(sina_E);
cosa_E=(cos(u)-cos(epsl).*cos(z_solved))./(sin(epsl).*sin(z_solved));
select=cosa_E<0;
a_E(select)=pi-a_E(select);
a_solved = a_E+E;
select = a_solved > (2*pi);
a_solved(select) = a_solved(select)-(2*pi);

% Error values for points and parameters

param_errors = [abs(a0-actual_a0), abs(x0-actual_x0), abs(y0-actual_y0),...
               abs(Q-actual_Q), abs(V-actual_V), abs(S-actual_S),...
               abs(E-actual_E), abs(epsl-actual_epsl)];

azimuth_errors = [abs(a_test-a_solved)];
a_solved_mod = a_solved; %these three lines prevent wrap around issues
select = azimuth_errors > pi;
a_solved_mod(select) = a_solved_mod(select) + 2*pi;
azimuth_errors = [abs(a_test-a_solved_mod)];
zenith_errors = [abs(z_test-z_solved)];

PARAMETER_SOLVED = [PARAMETER_SOLVED;a0,x0,y0,Q,V,S,E,eps1];
PARAMETER_ERRORS = [PARAMETER_ERRORS;param_errors];

points = {a_test;z_test};
POINTS_SOLVED = [POINTS_SOLVED,points];
az_errs = [(180/pi).*azimuth_errors];
zen_errs = [(180/pi).*zenith_errors];
AZIMUTH_ERRORS = [AZIMUTH_ERRORS;az_errs];
ZENITH_ERRORS = [ZENITH_ERRORS;zen_errs];

Avg_azimuth_err = [Avg_azimuth_err;mean(azimuth_errors)];
Avg_zenith_err = [Avg_zenith_err;mean(zenith_errors)];

end

% PLOTS
% Some labels may need to be changed if number of training points has been changed

% Average Azimuth errors
y_az = mean(AZIMUTH_ERRORS);
x_az = a_test*(180/pi);
figure()
plot(x_az,y_az,'o')
title('Azimuth residuals (outer-edge circle)');
xlabel('Azimuth value (degrees)');
ylabel('|residual| (degrees)');

% Azimuth heat map
rc=240;
figure()
[x,y,z] = cylinder(rc,200);
plot(x(1,:)+est_x0,y(1,:)+est_y0,'k')
hold on
axis equal
scatter(X_test,Y_test,15,y_az,'filled')
set(gca,'FontSize',16)
title('Average Azimuth Residuals (outer-edge circle)');
colorbar

% Average Zenith errors
y_z = mean(ZENITH_ERRORS);
x_z = z_test*(180/pi);
figure()
plot(x_z,y_z,'o')
title('Zenith residuals (outer-edge circle)');
xlabel('Zenith distance (degrees)');
ylabel('|residual| (degrees)');

% Zenith heat map
rc=240;
figure()
[x,y,z] = cylinder(rc,200);
plot(x(1,:)+est_x0,y(1,:)+est_y0,'k')
hold on
axis equal
scatter(X_test,Y_test,15,y_z,'filled')
```matlab
set(gca,'FontSize',16)
title('Average Zenith Residuals (outer-edge circle)');
colorbar
```
C.5 Parameter Variation Simulated Calibration

```matlab
clear;clc

error_val = 0;
PARAMETER_ERRORS = [];
PARAMETER_SOLVED = [];
AZIMUTH_ERRORS = [];
ZENITH_ERRORS = [];
POINTS_SOLVED = [];
AVG_AZIMUTH_ERROR = [];
AVG_ZENITH_ERROR = [];

[X_test,Y_test] = grid_gen(320,240,230,25); % Test Grid
X_test(125) = [];% Remove center point (comment out if center point is to be
Y_test(125) = [];% included

est_x0 = 320;
est_y0 = 240;

num_points = 15; % Change this value to alter the number of training points used

rc=200;

% Range of parameter values can be used. Adjust the values below to change
```

115
% Parameter values.
a0_range = linspace(1,90,30);
E_range = linspace(5,355,30);
epsl_range = linspace(0.1,5,30);

% Uncomment the line that contains the variable that is to be tested.
% for i = 1:length(a0_range)
% for j = 1:length(E_range)
% for l = 1:length(epsl_range)
Avg_azimuth_err = [];
Avg_zenith_err = [];

% Set initial distortion parameters
actual_a0 = a0_range(1)*((pi/180)); % The value in the parenthesis must be changed
actual_E = E_range(20)*((pi/180)); % to the variable value from line 50-52 in
actual_epsl = epsl_range(1)*((pi/180)); % order for the parameter to vary.

% These values cannot be changed
actual_x0 = est_x0 + 127*sin(actual_epsl/pi)*cos(actual_E);
actual_y0 = est_y0 + 127*sin(actual_epsl/pi)*sin(actual_E);
% actual_x0 = 310;
% actual_y0 = 245;

% Run simulations 100 times to obtain average residual values
for Its = 1:100
    X_train = [];
    Y_train = [];
    X_train_noise = [];
    Y_train_noise = [];
    for k = 1:num_points
\[ x, y = \text{cirrdnPJ}(\text{est}_x0, \text{est}_y0, \text{rc}); \]
\[ x_{\text{noise}} = x + 0.5 \times \text{randn}(1); \]
\[ y_{\text{noise}} = y + 0.5 \times \text{randn}(1); \]
\[ \text{X\_train} = [\text{X\_train}, x]; \text{Y\_train} = [\text{Y\_train}, y]; \]
\[ \% \text{Training vector w/out noise} \]
\[ \text{X\_train\_noise} = [\text{X\_train\_noise}, x_{\text{noise}}]; \]
\[ \% \text{Training vector w/ noise} \]
\[ \text{Y\_train\_noise} = [\text{Y\_train\_noise}, y_{\text{noise}}]; \]
\[ \text{end} \]

\section*{Determine necessary values}

\section*{Train}

\[ r = \sqrt{((\text{X\_train} - \text{actual}_x0)^2 + (\text{Y\_train} - \text{actual}_y0)^2)}; \]
\[ \text{actual}_V = 0.0054; \text{actual}_S = 0.0120; \text{actual}_Q = 3.24 \times 10^{-6}; \]
\[ u = \text{actual}_Q \times r^2 + \text{actual}_V \times r + \text{actual}_S; \]
\[ b = \text{actual}_a0 - \text{actual}_E + \text{atan2}((\text{Y\_train - actual}_y0), (\text{X\_train - actual}_x0)); \]
\[ z_{\text{train}} = \text{acos} (\text{cos}(u) \times \text{cos} (\text{actual}_\epsilon_p) - \text{sin} (\text{actual}_\epsilon_p) \times \text{sin} (u) \times \text{cos} (b)); \]
\[ \text{sina}_E = (\text{sin}(b) \times \text{sin}(u)) / \text{sin}(z_{\text{train}}); \]
\[ \text{a}_E = \text{asin} (\text{sina}_E); \]
\[ \text{cosa}_E = (\text{cos}(u) - \text{cos}(\text{actual}_\epsilon_p) \times \text{cos}(z_{\text{train}})) / (\text{sin}(\text{actual}_\epsilon_p) \times \text{sin}(z_{\text{train}})); \]
\[ \text{select} = \text{cosa}_E < 0; \]
\[ \text{a}_E(\text{select}) = \pi - \text{a}_E(\text{select}); \]
\[ \text{a\_train} = \text{a}_E + \text{actual}_E; \]
\[ \text{select} = \text{a\_train} > (2 \times \pi); \]
\[ \text{a\_train(\text{select})} = \text{a\_train(\text{select})} - (2 \times \pi); \]

\section*{Test}

\[ r = \sqrt{((\text{X\_test} - \text{actual}_x0)^2 + (\text{Y\_test} - \text{actual}_y0)^2)}; \]
\[ \text{actual}_V = 0.0054; \text{actual}_S = 0.0120; \text{actual}_Q = 3.24 \times 10^{-6}; \]
\[ u = \text{actual}_Q \times r^2 + \text{actual}_V \times r + \text{actual}_S; \]
b = actual_a0 - actual_E + atan2((Y_test-actual_y0),(X_test-actual_x0));

z_test = acos(cos(u).*cos(actual_epsl) - sin(actual_epsl).*sin(u).*cos(b));
sina_E=(sin(b).*sin(u))./sin(z_test);
a_E=asin(sina_E);

cosa_E=(cos(u)-cos(actual_epsl).*cos(z_test))./(sin(actual_epsl).*sin(z_test));
select=cosa_E<0;
a_E(select)=pi-a_E(select);
a_test = a_E+actual_E;
select = a_test > (2*pi);
a_test(select) = a_test(select)-(2*pi);

% This controls whether or not to use noise in training process

data = [X_train_noise;Y_train_noise;a_train;z_train];

% data = [X_train;Y_train;a_train;z_train];

% Begin determination of transformation parameters

sa0 = 0; % init_unk(1);
sx0 = 320; % init_unk(2);
sy0 = 240; % init_unk(3);
sQ = 0;
sS = 0;
sV = 0;
sepsl = 0;
sE = 0;

initial_conditions = [sa0,sx0,sy0,sQ,sS,sV,sepsl,sE];
[unknowns] = fsolve(@(unknowns)full_solver(unknowns,data),initial_conditions);
\(a0 = \text{unknowns}(1);\)
\(x0 = \text{unknowns}(2);\)
\(y0 = \text{unknowns}(3);\)
\(Q = \text{unknowns}(4);\)
\(V = \text{unknowns}(5);\)
\(S = \text{unknowns}(6);\)
\(epsl = \text{unknowns}(7);\)
\(E = \text{unknowns}(8);\)

% Determine azimuth and zenith values with estimated parameters from the solver

\[r = \sqrt{(X_{\text{test}} - x0)^2 + (Y_{\text{test}} - y0)^2};\]
\[u = Q \cdot r^2 + V \cdot r + S;\]
\[b = a0 - E + \text{atan2}((Y_{\text{test}} - y0), (X_{\text{test}} - x0));\]
\[z_{\text{solved}} = \cos(u) \cdot \cos(epsl) - \sin(epsl) \cdot \sin(u) \cdot \cos(b));\]
\[\sin(a)_{\text{E}} = (\sin(b) \cdot \sin(u))/\sin(z_{\text{solved}});\]
\[a_{\text{E}} = \sin(\sin(a)_{\text{E}});\]
\[\cos(a)_{\text{E}} = (\cos(u) - \cos(epsl) \cdot \cos(z_{\text{solved}}))/\sin(epsl) \cdot \sin(z_{\text{solved}});\]
\[\text{select} = \cos(a)_{\text{E}} < 0;\]
\[a_{\text{E}}(\text{select}) = \pi - a_{\text{E}}(\text{select});\]
\[a_{\text{solved}} = a_{\text{E}} + E;\]
\[\text{select} = a_{\text{solved}} > (2 \cdot \pi);\]
\[a_{\text{solved}}(\text{select}) = a_{\text{solved}}(\text{select}) - (2 \cdot \pi);\]

% Error values for points and parameters

\[\text{param\_errors} = [\text{abs}(a0 - \text{actual\_a0}), \text{abs}(x0 - \text{actual\_x0}), \text{abs}(y0 - \text{actual\_y0}) , \ldots\]
\[\text{abs}(Q - \text{actual\_Q}), \text{abs}(V - \text{actual\_V}), \text{abs}(S - \text{actual\_S}) \ldots\]
\[\text{abs}(E - \text{actual\_E}), \text{abs}(epsl - \text{actual\_epsl})];\]
azimuth_errors = \[\text{abs}(a_{\text{test}} - a_{\text{solved}})\];

\text{a\_solved\_mod} = a_{\text{solved}}; \text{\textit{\{these three lines prevent wrap around issues\}}}

\text{select} = \text{azimuth\_errors} > \pi;
\text{a\_solved\_mod}(\text{select}) = a_{\text{solved\_mod}}(\text{select}) + 2*\pi;

\text{select} = a_{\text{solved}} < 0;
\text{a\_solved}(\text{select}) = a_{\text{solved}}(\text{select}) + 2*\pi;

azimuth_errors = \[\text{abs}(a_{\text{test}} - a_{\text{solved\_mod}})\];

zenith_errors = \[\text{abs}(z_{\text{test}} - z_{\text{solved}})\];

\text{PARAMETER\_SOLVED} = \text{[PARAMETER\_SOLVED};a_0,x_0,y_0,Q,V,S,E,\text{eps1]};
\text{PARAMETER\_ERRORS} = \text{[PARAMETER\_ERRORS};\text{param\_errors}];

\text{points} = \{a_{\text{test}};z_{\text{test}}\};
\text{POINTS\_SOLVED} = \text{[POINTS\_SOLVED};\text{points]};
\text{az\_errs} = \text{[(180/\pi).\times\text{azimuth\_errors}]};
\text{zen\_errs} = \text{[(180/\pi).\times\text{zenith\_errors}]};

\text{AZIMUTH\_ERRORS} = \text{[AZIMUTH\_ERRORS};\text{az\_errs}];
\text{ZENITH\_ERRORS} = \text{[ZENITH\_ERRORS};\text{zen\_errs}];
\text{Avg\_azimuth\_err} = \text{[Avg\_azimuth\_err};\text{mean(azimuth\_errors)}];
\text{Avg\_zenith\_err} = \text{[Avg\_zenith\_err};\text{mean(zenith\_errors)}];

\text{end}
\text{AVG\_AZIMUTH\_ERROR} = \text{[AVG\_AZIMUTH\_ERROR};\text{mean(Avg\_azimuth\_err)}];
\text{AVG\_ZENITH\_ERROR} = \text{[AVG\_ZENITH\_ERROR};\text{mean(Avg\_zenith\_err)}];
\text{end}
% PLOTS

% some labels may need to be changed if number of training points has been
% changed

% Average azimuth errors

x_az = epsl_range; % Change this according to the variable being varied

y_AZ = AVG_AZIMUTH_ERROR*(180/pi);

figure()

plot(x_az, y_AZ, 'o')

set(gca, 'FontSize', 16)

title('Average Azimuth Residuals vs. epsilon range');

xlabel('epsilon value (degrees)');

ylabel('|residual| (degrees)');

% Average zenith errors

x_z = epsl_range; % Change this according to the variable being changed

y_Z = AVG_ZENITH_ERROR*(180/pi);

figure()

plot(x_z, y_Z, 'o')

set(gca, 'FontSize', 16)

title('Average Zenith Residuals vs. epsilon range');

xlabel('epsilon value (degrees)');

ylabel('|residual| (degrees)');
C.6 Non-Linear Solver

```matlab
function FUN = full_solver(unknowns, data)

a0 = unknowns(1);
x0 = unknowns(2);
y0 = unknowns(3);
Q = unknowns(4);
V = unknowns(5);
S = unknowns(6);
eps1 = unknowns(7);
E = unknowns(8);

X = data(1,:);
Y = data(2,:);
a = data(3,:);
z = data(4,:);

r = [sqrt((X-x0).^2 + (Y-y0).^2)];
u = Q.*r.^2 + V*r + S;
b = a0 - E + atan2((Y-y0),(X-x0));

FUN = [cos(z) - (cos(u)*cos(eps1) - sin(u)*sin(eps1).*cos(b));
        (sin(b).*sin(u))./sin(z) - sin(a-E)];
```
C.7 Right Ascension and Declination to Altitude and Azimuth Converter

```matlab
function [Az El] = RaDec2AzEl(Ra,Dec,lat,lon,time)

% Programed by Darin C. Koblick 01/23/2010

% External Function Call Sequence:

% [Az El] = RaDec2AzEl(0,0,0,-104,'1992/08/20 12:14:00')

% Worked Example: pg. 262 Vallado


% Worked Example: http://www.stargazing.net/kepler/altaz.html

% [Az El] = RaDec2AzEl(344.95,42.71667,52.5,-1.91667,'1997/03/14 19:00:00')

% [311.92258 22.4010] = RaDec2AzEl(344.95,42.71667,52.5,-1.91667,'1997/03/14 19:00:00')

% [Beta,el] = RaDec2AzEl(alpha_t, delta_t, phi, lambda, 'yyyy/mm/dd hh:mm:ss')

% Function Description:

% RaDec2AzEl will take the Right Ascension and Declination in the topocentric reference frame, site latitude and longitude as well as a time in GMT

% and output the Azimuth and Elevation in the local horizon reference frame.

% Inputs:

% Topocentric Right Ascension (Degrees) [N x 1]
% Topocentric Declination Angle (Degrees) [N x 1]
% Lat (Site Latitude in degrees -90:90 \rightarrow S(-) N(+)) [N x 1]
```

123
Lon (Site Longitude in degrees −180:180 W(−) E(+)) [N x 1]

UTC (Coordinated Universal Time YYYY/MM/DD hh:mm:ss) [N x 1]

Outputs: Format:

Local Azimuth Angle (degrees) [N x 1]

Local Elevation Angle (degrees) [N x 1]

External Source References:

Fundamentals of Astrodynamics and Applications
D. Vallado, Second Edition

Example 3–5. Finding Local Siderial Time (pg. 192)
Algorithm 28: AzElToRaDec (pg. 259)

Example 3–5

[yyyy mm dd HH MM SS] = datevec(datenum(time,'yyyy/mm/dd HH:MM:SS'));
JD = juliandate(yyyy,mm,dd,HH,MM,SS);
T_UT1 = (JD-2451545)./36525;
ThetaGMST = 67310.54841 + (876600*3600 + 8640184.812866).*T_UT1 ...
+ 0.093104.*(T_UT1.^2) - (6.2*10^-6).*(T_UT1.^3);
ThetaGMST = mod((mod(ThetaGMST,86400*(ThetaGMST./abs(ThetaGMST)))/240),360);
ThetaLST = ThetaGMST + lon;

Equation 4–11 (Define Siderial Time LHA)

LHA = mod(ThetaLST - Ra,360);

Equation 4–12 (Elevation Deg)

El = asind(sind(lat).*sind(Dec)+cosd(lat).*cosd(Dec).*cosd(LHA));

124
Az = \text{mod}\left(\frac{\text{atan2}\left(-\sin(LHA)\cdot \cos(Dec)/\cos(El)\right)}{\left(\sin(Dec) - \sin(El)\cdot \sin(lat)/(\cos(El)\cdot \cos(lat))\right)/(180/\pi)\cdot 360}\right);

\textbf{function} \quad \text{julian date}(\text{year}, \text{month}, \text{day}, \text{hour}, \text{min}, \text{sec})

\textbf{YearDur} = 365.25;

\textbf{for} i = \text{length(month)}: -1: 1

\quad \textbf{if} (\text{month}(i) \leq 2)

\quad \quad \text{year}(i) = \text{year}(i) - 1;

\quad \quad \text{month}(i) = \text{month}(i) + 12;

\quad \textbf{end}

\textbf{end}

\textbf{A} = \text{floor}(\text{YearDur}\cdot(\text{year}+4716));
\textbf{B} = \text{floor}(30.6001\cdot(\text{month}+1));
\textbf{C} = 2;
\textbf{D} = \text{floor}(\text{year}/100);
\textbf{E} = \text{floor}(\text{floor}(\text{year}/100)\cdot .25);
\textbf{F} = \text{day} - 1524.5;
\textbf{G} = (\text{hour} + (\text{min}/60) + \text{sec}/3600)/24;
\textbf{jd} = A+B+C-D+E+F+G;
C.8 Test Grid Generator

```matlab
function [x,y] = grid_gen(xc,yc,rc,space)

% This function generates (x,y) coordinates for a circular grid of points
% with radius 'rc' and origin ('xc','yc'). The spacing between points is an
% integer value specified by 'space.'

[X,Y] = meshgrid(xc-rc:space:xc+rc, yc-rc:space:yc+rc);

x = []; y = [];
for k = 1:length(X(:,1))
    for j = 1:length(Y(:,1))
        x = [x,X(k,j)];
        y = [y,Y(k,j)];
    end
end

rad = sqrt((xc - x).^2 + (yc - y).^2);

select = rad >= rc;

x(select) = [];
y(select) = [];
```

126
C.9 Polar Plot Generator

function hpol = polarmy(theta,rho,line_style,rmaxx,rtick)

%POLAR Polar coordinate plot.
% POLAR(THETA, RHO) makes a plot using polar coordinates of
% the angle THETA, in radians, versus the radius RHO.
% POLAR(THETA, RHO,S) uses the linestyle specified in string S.
% See PLOT for a description of legal linestyles.
% See also PLOT, LOGLOG, SEMILOGX, SEMILOGY.

% Copyright 1984-2002 The MathWorks, Inc.
% $Revision: 5.22 $ $Date: 2002/04/08 21:44:28 $

if nargin < 1
error('Requires 2 or 3 input arguments.')
elseif nargin == 2
  if isstr(rho)
    line_style = rho;
    rho = theta;
    [mr,nr] = size(rho);
  elseif mr == 1
    theta = 1:nr;
  else
    th = (1:mr)';
    theta = th(:,ones(1,nr));
  end
else
  line_style = 'auto';
end
elseif nargin == 1
  line_style = 'auto';
rho = theta;
[mr,nr] = size(rho);
if mr == 1
theta = 1:nr;
else
th = (1:mr)';
theta = th(:,ones(1,nr));
end
end
if isstr(theta) | isstr(rho)
error('Input arguments must be numeric.');
end
if ~isequal(size(theta),size(rho))
error('THETA and RHO must be the same size.');
end

% get hold state

cax = newplot;
next = lower(get(ca, 'NextPlot'));
hold_state = ishold;

% get x-axis text color so grid is in same color

tc = get(ca, 'xcolor');
ls = get(ca, 'gridlinestyle');

% Hold on to current Text defaults, reset them to the
% Axes' font attributes so tick marks use them.

fAngle = get(ca, 'DefaultTextFontAngle');
fname = get(ca, 'DefaultTextFontName');
fsSize = get(ca, 'DefaultTextFontSize');
fwWeight = get(ca, 'DefaultTextFontWeight');
fUnits = get(cax, 'DefaultTextUnits');
set(cax, 'DefaultTextFontAngle', get(cax, 'FontAngle'), ...
'DefaultTextFontName', get(cax, 'FontName'), ...
'DefaultTextFontSize', get(cax, 'FontSize'), ...
'DefaultTextFontWeight', get(cax, 'FontWeight'), ...
'DefaultTextUnits', 'data')

% only do grids if hold is off
if ~hold_state

% make a radial grid
hold on;
maxrho = max(abs(rho(:)));  
hhh = plot([-maxrho maxrho maxrho -maxrho], [-maxrho maxrho maxrho -maxrho]);
set(gca, 'dataaspectratio', [1 1 1], 'plotboxaspectratio', 'auto')
v = [get(cax, 'xlim') get(cax, 'ylim')];
ticks = sum(get(cax, 'ytick') >= 0);
delete(hhh);

% check radial limits and ticks
rmin = 0; rmax = v(4); rticks = max(ticks-1,2);
if rticks > 5 % see if we can reduce the number
if rem(rticks, 2) == 0
rticks = rticks/2;
elseif rem(rticks, 3) == 0
rticks = rticks/3;
end
end

% ///////////////////////////////////////////////////  
% attention I added these lines  
% rtick and rmax are new function inputs  
rticks = rtick*2;
max = rmax;
min = -rmax;

% define a circle
th = 0:pi/50:2*pi;
xunit = cos(th);
yunit = sin(th);

% now really force points on x/y axes to lie on them exactly
inds = 1:(length(th)-1)/4:length(th);
xunit(inds(2:2:4)) = zeros(2,1);
yunit(inds(1:2:5)) = zeros(3,1);

% plot background if necessary
if ~isstr(get(cax,'color'))
    patch('xdata',xunit*rmax,'ydata',yunit*rmax,...
        'edgecolor',tc,'facecolor',get(gca,'color'),...
        'handlevisibility','off');
end

% draw radial circles

c82 = cos(82*pi/180);
s82 = sin(82*pi/180);
rinc = (rmax-min)/rticks;
for i=(min+rinc):rinc:rmax
    hhh = plot(xunit*i,yunit*i,ls,'color',tc,'linewidth',1,...
        'handlevisibility','off');
    text((i+rinc/20)*c82,(i+rinc/20)*s82,...
        [''' num2str(i)],'verticalalignment','bottom',...
        'handlevisibility','off')
end
set(hhh,'linestyle','-') % Make outer circle solid
% plot spokes
th = (1:6)*2*pi/12;
cst = cos(th); snt = sin(th);
cs = [-cst; cst];
sn = [-snt; snt];
plot(rmax*cs, rmax*sn, ls, 'color', tc, 'linewidth', 1, ...
     'handlevisibility', 'off')

% annotate spokes in degrees
rt = 1.1*rmax;
for i = 1:length(th)
    text(rt*cst(i), rt*snt(i), int2str(i*30), ...
         'horizontalalignment', 'center', ...
         'handlevisibility', 'off');
    if i == length(th)
        loc = int2str(0);
    else
        loc = int2str(180+i*30);
    end
    text(-rt*cst(i), -rt*snt(i), loc,'horizontalalignment', 'center', ...
         'handlevisibility', 'off')
end

% set view to 2-D
view(2);

% set axis limits
axis(rmax*[-1 1 -1.15 1.15]);
end

% Reset defaults.
set(cax, 'DefaultTextFontAngle', fAngle, ... 'DefaultTextFontName', fName, ...
'DefaultTextFontSize', fSize, ...
'DefaultTextFontWeight', fWeight, ...
'DefaultTextUnits', fUnits);

% transform data to Cartesian coordinates.
xx = rho.*cos(theta);
yy = rho.*sin(theta);

% plot data on top of grid
if strcmp(line_style,'auto')
    q = plot(xx,yy);
else
    q = plot(xx,yy,line_style);
end
if nargout > 0
    hpol = q;
end
if ~hold_state
    set(gca,'dataaspectratio',[1 1 1]), axis off; set(cax,'NextPlot',next);
end
set(get(gca,'xlabel'),'visible','on')
set(get(gca,'ylabel'),'visible','on')
C.10  Random Point Generator (Quadrant)

function [x y]=cirrdnPJ_loc(x1,y1,rc,quad)

% This function will randomly generates (x,y) locations for points within a
% radius of rc from the origin (x1,y1). The random point will be within
% the quadrant(s) specified by 'quad.'

if isempty(quad)
    a=2*pi*rand;
elseif length(quad) == 1
    if quad == 1
        a = pi/2*rand;
    elseif quad == 2
        a = pi/2 + pi/2*rand;
    elseif quad == 3
        a = pi + pi/2*rand;
    elseif quad == 4
        a = (3*pi)/2 + pi/2*rand;
    end
elseif length(quad) == 2
    if quad == [1,2]
        a = pi*rand;
    elseif quad == [1,3]
        a = pi*rand;
        if a > pi/2
            a = a*pi/2;
        end
    elseif quad == [1,4]
        a = pi*rand;
        if a > pi/2
            a = a*pi;
        end
    elseif quad == [2,3]
        a = pi*rand;
        if a > pi/2
            a = a*pi/2;
        end
    elseif quad == [2,4]
        a = pi*rand;
        if a > pi/2
            a = a*pi/2;
        end
    elseif quad == [3,4]
        a = pi*rand;
        if a > pi/2
            a = a*pi/2;
        end
    end
end

end
end
elseif quad == [2,3]
    a = pi/2 + rand*pi;
elseif quad == [2,4]
    a = pi/2 + rand*pi;
    if a > pi
        a = a*pi/2;
    end
elseif quad == [3,4]
    a = pi + rand*pi;
end
elseif length(quad) == 3
    if quad == [1,2,3]
        a = (3*pi)/2*rand;
    elseif quad == [1,2,4]
        a = (3*pi)/2*rand;
        if a > pi
            a = a*pi/2;
        end
    elseif quad == [1,3,4]
        a = (3*pi)/2*rand;
        if a > pi/2 && a < pi
            a = a*pi;
        end
    elseif quad == [2,3,4]
        a = pi/2 + (3*pi)/2*rand;
    end
elseif quad == [1,2,3,4]
    a = 2*pi*rand;
end
r=rand;
\[ x = (rc \cdot r) \cdot \cos(a) \cdot x_1; \]
\[ y = (rc \cdot r) \cdot \sin(a) \cdot y_1; \]
\text{end}
C.11 Random Point Generator (Radial)

```matlab
function [x y]=cirrdnPj_rad(x1,y1,rc,rad)

% This function will randomly generates (x, y) locations for points within a
% radius of rc from the origin (x1, y1). The random point will be within
% the radial region specified by 'rad.'

% rad = 1: 0 – 45 degrees
% rad = 2: 45 – 90 degrees
% rad = 3: 75 – 90 degrees

dist = 0.5;
a=2*pi*rand;
if rad == 1
    r=dist*sqrt(rand);
    x=(rc*r)*cos(a)+x1;
    y=(rc*r)*sin(a)+y1;
elseif rad == 2
    a=2*pi*rand;
    r=0.5 + 0.8*dist*sqrt(rand);
    x=(rc*r)*cos(a)+x1;
    y=(rc*r)*sin(a)+y1;
elseif rad ==3
    r = 0.75 +0.15*sqrt(rand);
    x=(rc*r)*cos(a)+x1;
    y=(rc*r)*sin(a)+y1;
end
```

136
C.12 Random Point Generator

```matlab
function [x y]=cirrdnPJ(x1,y1,rc)

% This function will randomly generates (x,y) locations for points within a
% radius of rc from the origin (x1,y1). The random point will be within
% the radius specified by rc.

a=2*pi*rand;
rc=0.8*sqrt(rand);
x=(rc*r)*cos(a)+x1;
y=(rc*r)*sin(a)+y1;
end
```

137
REFERENCES


