Automating Camera Distortion Correction Using LMS Data Fitting

Michael Evan Loutzenheiser

Follow this and additional works at: https://louis.uah.edu/honors-capstones

Recommended Citation
https://louis.uah.edu/honors-capstones/717

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.
Automating Camera Distortion Correction Using LMS Data Fitting

by

Michael Evan Loutzenheiser

An Honors Capstone submitted in partial fulfillment of the requirements for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

4/5/2022

Honors Capstone Director: Dr. Can Ertem

Chief Scientist – Maryland Advanced Development Laboratory

William Wilkerson

William Wilkerson

William Wilkerson

Date: 2022.05.01 11:59:20 -05'00'

Digitally signed by William Wilkerson

Date: 2022.05.01 11:59:34 -05'00'

Digitally signed by William Wilkerson
Honors Thesis Copyright Permission

This form must be signed by the student and submitted with the final manuscript.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Michael Lutzenheiser
Student Name (printed)

____________________________
Student Signature

4/20/22
Date
# Table of Contents

Table of Contents ........................................... 1

Dedication ..................................................... 2

Abstract ....................................................... 3

Chapter 1: Introduction and Background ................. 4
  Background: GUMPS Phase 1 .............................. 4
  Analysis Introduction and Overview ...................... 5

Chapter 2: Project Overview ................................ 12
  Hardware ..................................................... 12
  Software ..................................................... 14

Chapter 3: Error Analysis and Fit Optimization ........ 23
  Approach to Error Analysis ............................... 23
  Choosing an Approximation Method ...................... 23
  Pixel Irregularity .......................................... 25
  Finding the Optimal Fit Function ......................... 29
  More Error Sources ......................................... 30

Conclusion ................................................... 34

Reference List ................................................ 35

Appendix ....................................................... 36
Dedication:

I would like to dedicate this thesis to Dr. Ertem, Roger Pierson, and the rest of the wonderful team at URF. They were extremely welcoming, thoughtful, and patient during my time working in Maryland, and made for a very pleasant summer internship experience. Secondly, I would like to thank my parents, grandparents, and my fiancée Mary for supporting me and encouraging me throughout my undergraduate education.

With love,

Michael Loutzenheiser
Abstract

The purpose for the GUMPS (Ground Based Ultrawideband Multistatic Positioning System) camera calibration was to develop and test the theory that given two targets of known distance apart on a plane, the pixel locations within the image taken of the two targets could be used to effectively estimate the overall distance from the camera to the target plane. To go about affirming this idea, a small scale set up was first tested in an office and then verified with different equipment and scale in an aircraft hangar. Throughout the process, lots of information was to be learned regarding variables that impact the accuracy of the distance approximation, including camera angle assumptions, irregular pixel sizes, and varying LMS coefficient matrices. In summary, two methods were used to approximate distance. The first of which initially approximates camera angle, and the second approximates target separation distance using the small aperture assumption to calculate the angle between the target plane targets and the camera. While both methods are discussed in the report (referred to simply as ‘method 1’ and ‘method 2’ throughout this report, method 2 was chosen to be the most useful for the GUMPS application). After troubleshooting the code and system multiple times, the distance from the camera to the target plane could be consistently estimated with approximately 5% relative error using method 2.
Chapter 1: Introduction and Background

Background: GUMPS Phase 1

The purpose behind the GUMPS research program (Phase 1) is to “conduct a feasibility assessment of designing a ground based ultrawideband multistatic positioning system for Vertical Takeoff and Landing (VTOL) guidance” [1]. To assist Urban Air Mobility (UAM) of unmanned aircraft in GPS degraded urban environments, camera targets are placed on a landing platform. Since the targets are a known distance apart, an aircraft on-board logic could utilize the pixel locations of the targets gathered by the camera sensor to approximate distance to the landing target. An illustration of the concept is shown below in figure (1).

![GUMPS Phase 1 Concept Render](image)

**Figure (1):** GUMPS Phase 1 Concept Render [1]

To begin verifying the feasibility of this concept, an analysis is conducted with a simple IP camera, notably with a fixed lens. While in the final product, infrared or even thermal cameras maybe used, the concept proven by the simple IP camera is relatively transferable to other camera systems.
Analysis Introduction and Overview

For simplicity, the analysis is broken into 3 steps: data collection, LMS fit, and solving for distance.

Data collection

Beginning with data collection, an image is first taken containing target locations. An example image is shown below. For each data ‘point’, 5 values are collected and stored in an excel matrix using a MATLAB GUI (for greater detail on the software and operation of the GUI, see the software section). The vertical and horizontal \((x, y)\) coordinates of two targets are recorded within the pixel matrix domain of the image, along with the real distance (measured in centimeters) between the two targets on the target plane.

Figure (2): Data Included in a ‘point’ (Image Taken at URF Hangar)
Throughout the course of data collection, around 150 data points are collected, all of which vary in target separation distances and locations within the FOV (field of view) of the camera. Including data from varying locations within the FOV, especially the corners and edges of the frame, is especially important for the LMS fit of the data. This diversity in location aids in accounting for pin-cushion distortion of the camera lens (see figure (3)).

![Barrel and Pincushion distortion illustration](image)

**Figure (3):** Barrel and Pincushion distortion illustration [3]

For more information on accounting for pin cushion distortion, see the appropriate section in chapter 3.

LMS Fit and Solving for Distance

Following data collection, the data is organized and placed into two matrices, matrix $A$ and matrix $b$. While the contents of the matrix varied, for the purposes of this example the two matrices will be defined as follows:

$$A = \begin{bmatrix} 1 & d_p & (d_p \times x_a) & (d_p \times y_a) & (d_p \times x_b) & (d_p \times y_b) \end{bmatrix}$$

$$B = [\alpha]$$
Where, \( x_a = x_1 - x_c, x_b = x_2 - x_c, y_a = y_1 - y_c, \) and \( y_b = y_2 - y_c. \) \( x_c \) and \( y_c \) are defined as the center pixel in the image. By mapping all the pixel coordinates to the center image, we assume that the focal axis of the image is reasonably centered. Alpha is defined as the angle between the camera and the two targets on the target plane and is calculated with the equation

\[
\alpha = \arctan\left(\frac{d}{D}\right)
\]

Where \( d \) is target separation distance and \( D \) is the distance from the camera to the target plane at which the calibration data was taken. \( d_p \) is used for ‘pixel distance’ and is defined by the equation:

\[
d_p = \sqrt{\left(x_1 - x_2\right)^2 + \left(y_1 - y_2\right)^2}.
\]

It is worth noting that this equation is only needed for ‘method 1’ calculations, as the purpose of ‘method 2’ is to estimate \( d_p \) instead of calculating it. After populating both matrices with 100 data points, for instance, we yield two matrices with dimensions \( A = [100 \times 4] \) and \( b = [100 \times 1]. \) Using these matrices, we can correlate pixel coordinate data (organized in matrix \( A \)) with angle data (organized in matrix \( b \)) using a simple linear equation in the form \( Ax = b. \) To solve for the \( x \) coefficients in this equation, we refer to the formula below:

\[
x = (A^T A)^{-1} A^T b
\]

With the resulting \( x \) coefficients, we can now take any two pixel locations in an image and their associated real separation distance, populate the \( A \) matrix, and estimate the angle between the camera and target platform. Using this newly estimated angle, we can complete the final step,
solving for distance between the camera and the target platform \( D \). To do this we simply use the following equation:

\[
D = \frac{d}{\tan(\alpha_t)}
\]

where \( d \) is the target separation distance and \( \alpha_t \) is the newly estimated angle. Since these variables can be somewhat confusing, the diagram below is included for reference.

![Diagram showing grid separation \( d \) and overall distance \( D \).](image)

Figure (4): LMS Fit Variables

While the organizing the data in the form \([\text{Coordinate data} \ [x \text{ coefficients}]] = [\text{angles}]\) proves to be a somewhat accurate means of estimating camera to target plane distance \( D \), method 2 can also be used. This method organizes the data in the form \([\text{coordinate data} \ [x \text{ coefficients}]] = [\text{distance}^2]\). For the purposes of this example, the matrices format is described below:

\[
A = [\ (x_a - x_b)^2, (y_a - y_b)^2, \left|\frac{x_a - x_b}{2}\right| (x_a - x_b)^2, \left|\frac{y_a - y_b}{2}\right| (y_a - y_b)^2 \ ]
\]

\[
b = [\ d^2 \ ]
\]

where, \( x_a = x_1 - x_c, x_b = x_2 - x_c, y_a = y_1 - y_c, y_b = y_2 - y_c, \) and \( d = \) target separation distance. After populating both matrices with 100 data points, for instance, we yield two matrices with dimensions \( A = [100 \times 4] \) and \( b = [100 \times 1] \). To associate the pixel location matrix
(A) with the target distance matrix (b), a linear equation is created in the following form $Ax = b$, like the first method. This equation can then be solved for the coefficient matrix $[x]$ using the following formula:

$$[x] = (A^TA)^{-1}A^Tb$$

Due to the [coordinate] $[x] = [d^2]$ format of this new linear equation, given a new set of pixel data we estimate a distance $d$ apart by multiplying the newly populated A matrix by the calculated coefficients in the x matrix. Now, to solve for camera to target plane distance $D$, a very important assumption must be made. Due to the IP Camera’s extremely small sensor and lens aperture, we can assume that for a wide variety of $D$ distances, the angle between the targets and the camera remains constant. This is referred to as the small aperture assumption and is detailed below in figures 5a and 5b. For every height that the drone exists above the target plane, the angle of the three lines representing the drones aperture stay the same. Using this assumption, we can calculate distance $D$ using the following formula:

$$D = d / \tan(\alpha_o)$$

where $\alpha_o$ is the original angle between the camera and target plane. This angle calculated using the following equation:

$$\alpha = \arctan \left( \frac{d}{D} \right)$$

where $d$ is target separation distance and $D$ is the distance from the camera to the target plane at which the calibration data was taken. Both methods of estimating overall distance have their
pros and cons. While the first method may exhibit tolerable error over a wider range of $D$

distances, the second method is much more accurate, albeit over a smaller range.

More on the Small Lens Aperture Assumption

![Illustration of the Small Lens Aperture Assumption](image)

**Figure (5a):** Illustration of the Small Lens Aperture Assumption [1]

Figure 5a illustrates a lose interpretation of the small lens aperture assumption. Namely, when
the camera is within a certain field of view of the target plane (denoted by the yellow area), the
camera angle $\alpha$ stays the same for multiple distances $D$ from the target plane. This is due to the
small aperture of the camera, providing the relatively small sensor with a huge depth of field.
The following figure illustrates the depth of field for multiple different lens apertures:
Figure (5b): Depth of Field Chart [2]
Chapter 2: Project Overview

Hardware

For the both analyses (office and hangar), Reolink RLC-1220A smart PoE IP security cameras were used to collect images and image related data. In the office setup, a 10x10cm dowel grid was used to collect initial calibration data. The grid/camera setup in the office is shown in figure (3). The camera was placed on a swivel mount, so different angles could be tested to simulate different pitch, roll, and yaw angles the camera may experience. 10x10cm dowel grid was far from perfectly spaced, this provided a simple test environment to develop and test initial code before moving to the hangar space.

![Figure (6): GUMPS Camera Initial Test Setup at UTS Office](image)

At the hangar test environment, the dowel grid target plane is replaced with a wooden platform mounted to lockable casters. This would serve as a movable plane where targets could be placed all over for data collection. Above the target plane, mounted in the rafters, the IP
camera was mounted on a circular wooden platform suspended by three ropes. The ropes are fixed to a table on the ground, running through pulleys before being attached to the wooden camera platform. This provides the user with 5 degrees of freedom for simulating pitch, roll, and yaw in addition to vertical and horizontal motion above the target plane. The hangar test environment is shown in figure (7) below:

![Hangar Target Plane](image)

**Figure (7):** Hangar Target Plane

For both test environments, the cameras were linked to UTS’s private network, making access to the cameras easy through PoE (Power over Ethernet) connection. For both environments, control distances from the camera to the target plane were recorded with a simple 30-meter tape measure. For recording distances between targets on the hangar target plane, this tape measure was used in conjunction with peel and stick targets, in addition to printed 5x5cm grids.
Software

Phase 1: Data Collection

To ease the burden of collecting 100+ points of data, each with 5 values a piece, a MATLAB GUI (Graphical User Interface) was developed. The GUI’s primary purpose is to make the selection, input, and storage of data points as easy as possible utilizing MATLAB’s Image Acquisition toolbox. The source code for the GUI is attached in the appendix.

![Image Data Acquisition GUI](image.png)

**Figure (8):** Data Acquisition GUI Main Window
To open the GUI, the user simply types ‘app1’ (or any other file name associated with the source code) into the MATLAB command line. To begin the data collection process, the user selects the ‘Open Pixel Selector’ button, opening a figure incorporating the Image Acquisition Toolbox as shown below.

![Image Acquisition Toolbox Window](image)

**Figure (9):** Pop-up Image Acquisition Toolbox Window

Within the ‘Pixel Selector’ window, the user can use the ‘w’ and ‘s’ keys, respectively, to zoom in and out of the image. This aids the user in being able to select precise pixel locations of targets on the image. After selecting two target locations, a pop-up field is displayed, prompting the user to input the real target distance (otherwise known as $d$) in the input field. The system will not accept zero or negative distances. Also, a blue line with asterisk (*) end-point markers is
displayed on the main window’s coordinate plane. This aids the user in highlighting areas within the camera’s frame and FOV that have been selected and logged in addition to highlighting areas that still need data associated with them. Both the input field and main window coordinate plane are shown in the image below.

Figure (10): Distance Input Field and Main Window Coordinate Axes

Upon selecting ‘Enter’ in the distance input field, the ‘New Data’ table line in the bottom of the main window is appended with the real distance of between the targets. Appending the new line prior to saving the information to the embedded excel sheet immediately allows the user to review the data before submitting, double checking to make sure everything looks valid
before it is much more difficult to erase the newly added data. Upon hitting ‘Save’ next to the ‘New Line’ table, the previously blue line in the main window’s coordinate plane is turned to a red color, signifying that the data has been saved to the embedded excel spreadsheet, an example of which is shown below:

![Image of Excel Spreadsheet](image)

**Figure (11):** Excerpt from the Excel Spreadsheet Database, Managed by the MATLAB GUI

Within the GUI, there are a few other buttons and features that aid in improving the user’s experience in efficiently recording pixel location data. First, there is the ‘Clear New Line’ button, which allows the user to erase the previously selected data stored in the temporary ‘New Line’ table location. Next, the ‘Select Camera Angle’ button allows users to toggle between different
sheets denoted by the labels ‘Angle 1’, ‘Angle 2’, ‘Angle 3’ and so on within the spreadsheet
database. Lastly, the ‘Close and Save Data’ button, located at the bottom of the main window,
provides users with an easy means of exiting the GUI.

Phase 2: LMS fit and Distance Calculation

For both distance approximations outlined in background section, the general flow of
the LMS fit code is very similar. For this example, the final code is used from method 2 is
shown. Since this method automatically accounts for pixel irregularity, it does not include
functions accounting that factor in scaling factors for pixel locations. For more information on
scaling factors and adjustment for pixel irregularity, see the error section. To begin the code,
the user must collect test values to use in estimating the distance from the camera to the target
plane. To do so, a function ‘gettestvals’ is used. The output of the function returns an array of
pixel coordinates, along with the actual target separation distances. This data format is like that
which is found using the GUI mentioned in the previous section. At the core of the gettestvals
function is the use of MATLAB’s Image Acquisition toolbox, the same tool used earlier in the
GUI.

```
7    %% Gather Data
8    n = 1;
9    [xyarray,test_d] = gettestvals(n);
```

**Figure (12):** Declaration of the ‘gettestvals’ function

Following the acquisition of test points to estimate distance from, the calibration data acquired
by the GUI must be taken from the spreadsheet database and organized. Following this, the
calibration distance is declared and stored in the variable name ‘cal_d’, for calibration distance.

Then, to gather information regarding the center pixel coordinates \((x_c, y_c)\) of the image, the ‘iminfo()’ function is used from the Image Acquisition Toolbox. Finally, a print statement summarizes the error analysis output based on the number of data points being tested.

```matlab
11  % Load information and calculate LMS coefficients
12  cal_d = 497; % PixelLogi was made at a distance of 497 cm
13  table = [readmatrix('banana_407.xlsx', 'Sheet', 'Sheet1')];
14  table = [table(:,1) table(:,2) table(:,3) table(:,4) table(:,5)];
15  information = iminfo('http://next.runway.com/cgi-bin/api.cgi?cmd=Snap&channel=U_tac&user=cameras&password=cameras');
16  fprintf('LMS Error Analysis with %d data points\n', length(table(:,1)));
```

**Figure (13):** Gathering and Organizing Calibration Data

Following the print statement, a discrete function is declared. This function is useful in populating the A matrix and can be modified for different forms of the A matrix. Next, the function ‘getcords’ is used to organize the calibration data obtained with the ‘readmatrix()’ function shown above in figure 13. This function, included in the appendix, returns all the data needed for the LMS fit, including alpha. Using this information, an A matrix is populated using the ‘Afind()’ function described above, and the linear equation \(Ax = b\) is solved using MATLAB’s backslash operator. The backslash operator invokes a specialized algorithm that optimizes efficiency and accuracy in solving the rather large system for the coefficient matrix. In the case of the example below, note that the target separation distances are squared, as this is image is indicative of code used in method 2.
Figure (14): Solving the Linear Equation for the Coefficient Matrix

After solving for the x coefficient matrix, a separate function is employed to remove outlier calibration data. The idea behind this function is to take all of the points of data used to calculate the initial coefficient matrix and multiply the points by the coefficients. Since the calibration data was taken at a known distance, any approximations made with outlier calibration data will estimate distances that differ from the known calibration distance. To statistically support the removal of outlier calibration data, Chauvenet’s criterion is implemented. Using Chauvenet’s criterion, outlier data was determined to exist outside a z-score of 2.00. When using this z-score to remove outlier data, 5 points were removed on average from a total of 127 data points. The stated function is included in the figure below, and the function code is included in the appendix.

Figure (15): Removal of Outliers and Solving Again for the Coefficient Matrix

As shown in the figure above, following the remover of outlier data with the ‘removeoutliers’ function, the table of data is updated to a new table, and the process for calculating the coefficient matrix is repeated. Following the recalculation of the coefficient matrix, the next
step is to take the test data and approximate distance. First, the test data is mapped to the center pixel by subtracting the center pixel locations in the image, as shown below:

![Figure (16): Mapping to the Center Pixel](image)

Next, the A matrix is populated with the test data, using the function ‘Afind()’ defined earlier in the script. Then, a short loop is used to multiply each row by the coefficient x matrix element-by-element, and then summing the factors together to approximate distance. For this example, method number 2 is used, and the distance approximated is a target separation distance on the target plane. Then, due to the small aperture assumption, the angle is calculated using data from the calibration distance. Finally, distance to the target (‘D_est’) is approximated using the calculated angle and square root of the approximated distance.

![Figure (17): Estimating Target Separation Distance](image)

To conclude the script, relative error is calculated and multiple ‘fprintf’ statements are used to display the results of the approximation below.
Figure (18): Error Calculation and Summary Print Statements
Chapter 3: Error analysis and Fit optimization

Approach to Error Analysis

Due to the number of factors influencing the error in our system, it is quite difficult to analyze and pinpoint specific error sources or factors that contribute more to the error than other factors. For this reason, error was approached with an open form method. Throughout the process of gathering and approximating distance, different test points were simply taken at random from target locations, used to approximate distance, and compared to the calibration distance using the following relative percent error equation:

\[
\% \text{ error} = \frac{d_{\text{actual}} - d_{\text{measured}}}{d_{\text{measured}}} \times 100
\]

Throughout the error analysis, several factors were identified as the main contributors to error in the calibration. For the purposes of this report, the following sources will be discussed: 1) pixel irregularity, 2) Optimization of the fit function, 3) pitch and yaw of the camera lens, 4) pin-cushion distortion, 5) the small aperture assumption (associated with method 2), and 6) manual error.

Choosing an Approximation Method

As noted in the summary section, ‘method 2’ was chosen over ‘method 1’ as a better and more accurate method for approximating the distance from the target plane to the camera, with method 2 primarily estimating target separation distance based on the calibration data, and method 1 primarily estimating camera angle based on the calibration data. To test the two methods, 15 control data points were taken at a height of 442 cm, and the same 119 calibration
data points were used for each test (5 points were determined to be outliers based on Chauvenet’s criterion). The results of the test are displayed below, where tests 2 and 3 both represent method 2, where test 3 included a fit function with two more terms than test 2. Tests 1 and 4 were conducted using method 1, where test 1 used fit function number 7, and test 4 simply used pixel separation distance as the function.

<table>
<thead>
<tr>
<th>Test #</th>
<th>Method</th>
<th>Average % Error</th>
<th>Max % Error</th>
<th># Data Points</th>
<th># Outliers</th>
<th>Removed % Error</th>
<th>Z-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>11.25493284</td>
<td>24.33684259</td>
<td>119</td>
<td>0</td>
<td>0</td>
<td>2.635418459</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4.816437441</td>
<td>11.13257437</td>
<td>114</td>
<td>5</td>
<td>0.29398326</td>
<td>2.635418459</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3.5385649</td>
<td>9.109046157</td>
<td>113</td>
<td>6</td>
<td>394.6051585</td>
<td>2.635418459</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3.896675421</td>
<td>6.536172871</td>
<td>119</td>
<td>0</td>
<td>0</td>
<td>2.635418459</td>
</tr>
</tbody>
</table>

Looking at the data above, there is much to be learned from this test. First, it can be noticed that between the 4 different tests, both tests using method 2 were much more stable and reliable, while the tests conducted using method 1 had a larger range of mean error. When choosing between the two tests, it is known that method 2 does proves to be much less accurate over a range of distances. Therefore, if method 1 could be as accurate as method 2 at distances very close to the distance at which the calibration data was taken, there would be no need for method 2. However, this is not the case. Even though test 4 performed nearly as well as test 3, the linearity of the fit function provides almost no correction for pin-cushion distortion, something that is vital to the calibration of the camera. For this reason, method 2 was adopted as the conventional method demonstrated in the software overview. Under ideal
circumstances, with more calibration data and a more sophisticated and efficient method of gathering calibration data, both methods could be utilized for the GUMPS project. Method 1 could be used to estimate a distance range at which the camera is currently operating at, and method 2 could be used in conjunction with a customized set of calibration data to provide a more accurate assumption, given that the calibration data would then be well within the distance range where the small aperture assumption applies.

**Pixel Irregularity**

Because the LMS fit is transferring from a real-world measurement domain (in cm) to a ‘virtual’ measurement domain (in pixels), both domains need to be equally scaled for both the x and y axes of measurement. For the real-world measurement, this requires no calibration. A 1-centimeter measurement on the y axis of the target plane is equivalent to a 1-centimeter measurement on the x-axis of the target plane, assuming the measurement tools are calibrated and held constant. Within the virtual pixel domain however, this is not always the case. Instead, some pixels may not be square, but rather rectangular (or even in some cases triangular) in shape. The consequences of this are that a 3-pixel difference measurement on the y axis of the photo may represent a 5-centimeter measurement in the real world, while a 3-pixel difference measurement on the x-axis may represent a 7-centimeter measurement in the real world. In the figure below, we can imagine that the blue lines represent a rectangular pixel domain, while the black lines represent the same 2.4-inch measurement in the real world. Notice that in a rectangular pixel domain, 4 units on the y axis represent the same distance as 6 units on the x axis.
To account for this, two different methods can be employed, a manual method, and an ‘automatic’ method

Approach 1: Manually Accounting for Pixel Irregularity

To manually account for irregular pixels, a user simply needs to take a picture of an object they know is square and, with the help of trigonometry, calculate the difference between the number of pixels in the x-direction and number of pixels in the y-direction represented with the same real-world physical measurement. Then, the user can simply multiply the x or y axis by a factor, effectively squaring up the pixel coordinate axes. This method is extremely useful in the first method of distance approximation, where the distance between target pixels in the image must be calculated to approximate camera angle based on
the calibration data taken at a known distance. To do this efficiently, a function is written titled ‘getdistfact()’ and is included in the appendix. At the heart of the function is the trigonometry formulas shown below.

By taking four-pixel locations and subtracting the two respective x-axis and y-axis locations from one another, the function accounts for any rotation in the image. If this was not the case, the user would select target pixels that would not correspond to a perfect right angle measurement in the pixel domain, potentially leading to further inaccurate factor calculations.

While the function does do a good job of providing the user with an accurate means of accounting for rectangular pixels, there are some drawbacks. First, the function does not however, account for any pitch or roll of the camera lenses. While this does not have as large of a factor in calculating pitch as yaw does, it does present some inaccuracy to the calculation. Second, the pixels in the image are selected manually, and while the zoom tool incorporated into the function aids in this, the perfect pixel locations cannot be verified and selected, thus presenting further inaccuracy.
Approach 2: ‘Automatically’ adjustment using an appropriate coefficient matrix

One additional method for accounting for pixel irregularity is used in the second distance approximation method. In this method, it is important to note that the distance between pixels in the pixel coordinate system does not need to be calculated. Instead, the pixel irregularity is factored out in the creating of the ‘A’ matrix. The matrix used is shown again, here in the form of the ‘Afind()’ function discussed earlier.

```matlab
```

**Figure (22):** ‘Afind’ anonymous MATLAB function, used for populating the A matrix

To understand the matrix slightly better, we can expand the Ax = b linear equation form to include the ‘Afind’ matrix and ‘b’ solution, which in this case is distance squared:

\[
d^2 = x_1 \cdot (x_a - x_b)^2 + x_2 \cdot (y_a - y_b)^2 + x_3 \cdot \left| \frac{x_a + x_b}{2} \cdot (x_a - x_b)^2 \right| + x_4 \cdot \left| \frac{y_a + y_b}{2} \cdot (y_a - y_b)^2 \right|
\]

Using this equation, upon input of the test data in variables \(x_a, x_b, y_a,\) and \(y_b\) are inputted and automatically corrected for any pixel distortion, as the trigonometry is already incorporated into the construction of the ‘A’ matrix in the anonymous ‘Afind()’ function. Within the range of distances accurately estimated by the second approximation method, this method of automatically adjusting for pixel irregularity proved to be extremely accurate. For several tests, the average percent error was under 1%.
Finding the Optimal Fit Function

Early in the research process of the calibration, multiple fit functions were tested when attempting to approximate angle with the LMS fit. Later in the process, method 2 was adopted as the primary method, and the fit function shown above was chosen as the final function. However, there is still much to be learned regarding the choice of an optimum fit function when using angle approximation to approximate camera distance to the target plane. To test multiple fit functions, a standardized set of data was chosen and tested with multiple fit functions, all increasing in the number of terms, and thereby complexity. Below a figure is shown detailing the different fit functions and their respective reference numbers used throughout the test process.

![Figure (23): Test matrices](image)

Where, xa, xb, ya, and yb are all pixel locations, and ‘dist’ is a variable representing the distance between pixel locations in the image. To test the functions above, trials in the office were conducted at a distance of 213 cm from the target plane, and the following results were procured from the test:
Figure (24): Matrices test results

Noticeably, matrices with a pixel distance factor performed much better in the test. While these matrices have been shown to be too inaccurate, it can be noted and learned that functions that best approximate non-linear behavior do the best job in approximation. This hypothesis holds true for the fit function used in method 2, which uses 4 complex terms and does a good job of using a linear equation to represent a non-linear system. For this reason, the fit function previously mentioned in the previous section can be improved by simply adding additional terms to create the function below:

\[ d^2 = x_1 \times (x_a - x_b)^2 + x_2 \times (y_a - y_b)^2 + x_3 \times \left| \frac{x_a + x_b}{2} \times (x_a - x_b)^2 \right| + x_4 \times \left| \frac{y_a + y_b}{2} \times (y_a - y_b)^2 \right| \\
+ x_5 \times \left| \frac{y_a + y_b}{2} \times (x_a - x_b)^2 \right| + x_6 \times \left| \frac{x_a + x_b}{2} \times (y_a - y_b)^2 \right| \]

Undergoing the same testing described previously, this lengthier equation was shown to be more accurate than the previous shorter equation, as the average error calculated with the 15 data points improved from 4.82% to 3.54%.

More Error Sources

Pitch/Yaw of the Camera

Another major contributor to the error of the camera calibration are the pitch and yaw angles at which the camera is taking the photographs. While these are important factors to
acknowledge and address, they are not necessarily factoring that can or should be precisely controlled in the camera calibration, simply due to the design constraints of the project. Because the GUMPS project is designed to be implemented in drone flight, a perfectly level camera plane is nearly impossible, and something that the LMS fit function should be accounting for. So, while the error associated with these qualities is legitimate, it should be regarded as given or accepted error, and mitigated with the robustness of the LMS calibration. It should also be noted that yaw angles do not apply to the camera calibration, as the target pixel locations are all mapped to the center pixel in the LMS calibration.

Pin-Cushion Distortion

While the LMS fit function is designed to accommodate some pin cushion distortion in the camera lens, some error associated with the distortion is very detectable. To test this, 5 control data test points were collected and tested at 5 locations withing the camera frame using approximation methods 1 and 2. All tests were run at a height above the target plane of 442 cm, and calibrated using 245 points of data taken at a height of 442 cm.

![Diagram of test points]
Figure (25): Pin-Cushion Distortion Test Data Location Reference Graphic

The following results were procured from the tests. It should be noted, however, that not many calibration points were taken, contributing to the higher overall error in the pin cushion areas.

![Pin Cushion Distortion Error Test](image)

**Figure (26): Pin-Cushion Distortion Error Test Results**

It can also be noted that method 1 was conducted using a simple, linear, fit function. Because the A matrix only consisted of pixel distance, it was very accurate at the center of the image (location #5). However, it can be clearly seen that this method was less accurate in the areas of the image containing more distortion, especially areas 1 and 3.

Small-Aperture Assumption

As mentioned before in chapter 1, the small aperture lens assumption is used to implement certain mathematical characteristics of the camera system. This of course has some
error associated with the assumption, but it is believed to be negligible, as the camera sensor is extremely small. See chapter 1 for more analysis and figure 5 in the text for an illustration of the small aperture assumption.

Manual Error

Unfortunately, the theoretical nature of the calibration is not preserved throughout the calibration data process. In both the data collection and testing processes, there are discrepancies due to mechanical error. These error sources are categorized into pixel selection errors, measurement errors, and instrument roundoff.

Pixel selection error

Due to the manual nature of the pixel selection tool and ‘ginput’ function. While the zoom tool greatly aids in minimizing this error, there is almost certainly a 1-5 pixel discrepancy between measurements and selections within the imtool window. This may result in a small error in the distance approximation for any given point.

Measurement inaccuracies (roundoff error)

Throughout the testing and data collection process, the same 30-meter tape measure was used to measure distances between the camera and target planes (office and hangar), in addition to some target separation distances on the hangar target plane. This tape measure, while easy to use and quite effective, has no NIST traceability, and is assumed to be relatively accurate so as not to introduce any noticeable error.
Conclusion

Throughout the research process, it is reasonable to believe that LMS fit camera calibration can solve for the distance of a target from a camera plane and estimate the distance between that camera and the plan. While the error associated with the trials run with a simply LMS fit and a camera at the URF hangar have proved to be too inaccurate for proper testing and application, the hypothesis is still proven to be true. To further iterate and correct the process, more research needs to be conducted coning the fit function used for the distance approximation. Overall, the research experience was very helpful and enjoyable, and much was learned regarding camera technology, image sensors, MATLAB GUIs, and the Image Acquisition Toolbox.
Reference List


Appendix

1) MATLAB GUI source code

```matlab
function [xyarray, test_d] = gettestvals(n)
% n is the number of data points you would like to collect
counter = 1;
xyarray = [];
test_d = [];

while counter < n

rgbImage = imread('http://fme.urf.com/cgi-bin/api.cgi?cmd=GetImage&image=image.png');
imshow(rgbImage);
disp('[Select target points and press ENTER]');
number = 0;
xvals = [];
yvals = [];
while number < 2
    [x,y,b] = ginput(1);
    if isempty(b)
        break;
    elseif b == 115
        ax = axis;
        width = ax(2)-ax(1);
        height = ax(4)-ax(3);
        axis([x-width/1.5 x+width/1.5 y-height/1.5 y+height/1.5]);
        zoom(1/2);
    elseif b == 119
        ax = axis;
        width = ax(2)-ax(1);
        height = ax(4)-ax(3);
        axis([x-width/1.5 x+width/1.5 y-height/1.5 y+height/1.5]);
        zoom(2);
    else
        xvals = [xvals x];
        yvals = [yvals y];
        number = number + 1;
    end
end
xyarray = [xyarray; xvals; yvals]
input_d = input('Input the distance between the targets (cm) ');
test_d = [test_d input_d];
counter = counter + 1;
end
```

2) function [x, y, xB, yB, xc, yc, grid_separation] = getcords(information, table, d)

```matlab
pixy = [table(:,1) table(:,2)];
p2xy = [table(:,3) table(:,4)];
grid_separation = table(:,5); % real distance pulled from excel file
columns = information.Height; % in pixels
rows = information.Height;
xc = columns/2;
yc = rows/2;
alpha = atan((grid_separation)/d); \& atan(s/d);
xa = pixy(:,1)-xc;
ya = pixy(:,2)-yc;
xb = pixy(:,3)-xc;
yb = pixy(:,3)-yc;
end
```
function [D_a,new_table,num_outliers,Error_improvement,r] = removeoutliers(table,x,y,d,find,ydist_factor)
% Michael Loutzenheiser
% 6/29/2001
% [new_table,num_outliers,Error_improvement] = removeoutliers(table,x,y,d,find,ydist_factor)
% X is an array of LMS Coefficients
% table is a nx5 array of data points
% n is a marker for which Array to use:
% 0: A = A;
% 1: A = [distance];
% 2: A = [xa ya distance];
% d is the known distance the 'table' was recorded at
% xdist_factor is optional for calibrating the pixel width

clear outliers
clear new_table
clear num_outliers
clear Error_improvement
clear D_a

% Get Data
size_X = size(table);
num_points = size_X(1);
px = [table(:,1) table(:,5)];
py = [table(:,2) table(:,6)];
g = [table(:,1) table(:,6)];
columns = 408; % in pixels
rows = 3072;
xc = columns/2;
yc = rows/2;
xa = px(:,1)-xc; ya = py(:,2)-yc;
xb = py(:,1)-xc; yb = py(:,2)-yc;
if nargin < 6
   xdist_factor = 1;
end
new_table = table;
% distance = abs(sqrt((xdist_factor*(x-a).^2)+(y-b).^2));

% Get A matrix
A = Afind(xa,ya,yb);
% Loop through values to test them against x matrix
i = 1;
D_a = [];
caldist = [];
for i = 1:length(A(:,1))
    D_a = [D_a sum(A(i,:).'*x')];
caldist = [caldist d];
end
Angles = atand(sqrt(D_a)./caldist);
D_a = (grid_seperation'./tand(Angles))';
% Perform chauvenet's analysis on previous data set
N = (1 - 1/(2*og_numpoints)); % number of acceptable std deviations
E_array = (abs((D_a-d)./D_a)');
xbar = mean(E_array);
sx = std(E_array);
z = abs(norminv(N));
T = abs((E_array-xbar)./sx)';
outliers = [];
for i = 1:length(T)
    if T(i) > z
        outliers = [outliers i];
    end
end
D_a(outliers) = [];
new_table(outliers,:) = [];
num_outliers = length(outliers);
disp(outliers)
% disp(E_array)
Error_improvement = abs(mean(E_array) - ...
mean(abs((D_a-d)./D_a))).*(10^-1)
end
function [xfactor] = getdistfactor()
% This is used to get the adist_factor (coefficient multiplied by the x
% pixel values in distance calculation) to account for pixel distortion
% Select pixels in the following form:
% 2 - 3
% 1
% 1

   counter = 1;
   while counter <= 5
      rgbImage = imread('http://fme.urf.com/cgi-bin/api.cgi?cmd=Snapshot&vars=avuser=cameras&password=cameras');
      imshow(rgbImage);
      number = 0; xvals = []; yvals = [];
      while number < 3
         [x,y,b] = ginput(1);
         if isempty(b)
            break;
         elseif b == 15
            ax = axis; width = ax(2)-ax(1); height = ax(4)-ax(3);
            x = x+width/1.5 x+width/1.5 y+height/1.5 y+height/1.5;  
            zoom(1/2);
         elseif b == 119
            ax = axis;
            width = ax(2)-ax(1); height = ax(4)-ax(3);
            x = x+width/1.5 x+width/1.5 y+height/1.5 y+height/1.5;  
            zoom(3/2);
         else
            xvals = [xvals;x];
            yvals = [yvals;y];
            number = number + 1;
         end
      end
      xl(counter) = xvals(1); x2(counter) = xvals(2); x3(counter) = xvals(3);
      yl(counter) = yvals(1); y2(counter) = yvals(2); y3(counter) = yvals(3);
      counter = counter + 1;
      lengthy1 = y2-y3; lengthx1 = x2-x3;
      xfactor1 = lengthx1./lengthy1;
      lengthy2 = y2-y3; lengthx2 = x1-x2;
      xfactor2 = lengthx2./lengthy2;
      xfactor2 = 1./xfactor2;
      xfactor1 = 1./xfactor1;
      avg1 = sum(xfactor1)/length(xfactor1);
      avg2 = sum(xfactor2)/length(xfactor2);
      xfactor = (avg1+avg2)/2; %this is adistfactor.