6-2014

# Localization and System Identification of a Quadcopter UAV

Kenneth Befus
*Western Michigan University*, kenny.m.befus@gmail.com

# LOCALIZATION AND SYSTEM IDENTIFICATION OF A QUADCOPTER UAV

by

Kenneth Befus

A thesis submitted to the Graduate College
in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
Department of Mechanical and Aeronautical Engineering
Western Michigan University
June 2014

Doctoral Committee:

Kapseong Ro, Ph.D., Chair
Koorosh Naghshineh, Ph.D.
James W. Kamman, Ph.D.

# LOCALIZATION AND SYSTEM IDENTIFICATION OF A QUADCOPTER UAV

Kenneth Befus, M.S.E.

Western Michigan University, 2014

The research conducted explores the comparison of several trilateration algorithms as they apply to the localization of a quadcopter micro air vehicle (MAV). A localization system is developed employing a network of combined ultrasonic/radio frequency sensors used to wirelessly provide range (distance) measurements defining the location of the quadcopter in 3-dimensional space. A Monte Carlo simulation is conducted using the extrinsic parameters of the localization system to evaluate the adequacy of each trilateration method as it applies to this specific quadcopter application. The optimal position calculation method is determined.

Furthermore, flight testing is performed in which real range measurement data are collected for the purpose of post-processing and evaluation of the quadcopter's high-level open-loop response to three basic inputs: pitch/roll, thrust, and yaw rate (heading angle). The raw range measurement data allow for the calculation of position data that are then brought into the System Identification Toolbox environment within Matlab. This tool is then used to generate 'best fit' transfer functions for each of the aforementioned dynamic responses.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Kapseong Ro for the general support and direction that he offered as I progressed with the research and writing for my master's thesis. He has been very patient and accommodating over the course of this endeavor.

Secondly, I would also like to thank my parents for being incredibly accommodating and supportive of my thesis work. Their encouragement and provisions played a major role in allowing me to finish this thesis paper while working full-time.

I would like to thank my co-worker, Paul Stoving, for his help regarding the Monte Carlo Simulation that is presented in this thesis paper. His thoughts and comments helped me drive to achieve meaningful, relevant results that heavily contribute to the value of this research.

And finally, I would like to thank Dr. Koorosh Naghshineh for encouraging me to participate in the Accelerated Master's Program at WMU. His direction has enabled me to expedite my pursuit of a master's degree in mechanical engineering.

Kenneth Befus

TABLE OF CONTENTS

Table of Contents - continued

Table of Contents - continued

LIST OF TABLES

LIST OF FIGURES

List of Figures - continued

List of Figures - continued

# CHAPTER I

# INTRODUCTION

## 1.1 Introduction

The use of large unmanned aerial vehicles (UAVs) in today's world is becoming increasingly common, however, the efforts that have been made to utilize this technology on a smaller scale are few. Most existing smaller unmanned aircraft, often called micro air vehicles (MAV), require a human operator to actively navigate the vehicle by way of a remote control. This dependency on a human operator negates the potential benefits that could be observed from the automated flight of an MAV. One primary requirement for the automated flight and navigation of an MAV is an awareness of the vehicles location in space as well as its instantaneous dynamic state. And while current systems employ techniques that locate a unit via GPS, this type of location identification falls short in multiple ways. It lacks the ability to accurately, responsively, and reliably provide a position feedback at all times and/or anywhere. For these reasons, other methods of localization of an MAV are being explored.

One such localization method to be used in the context of MAVs and other automated vehicles is based on range measurements via radio frequency (RF) sensors. Multiple range measurements allow trilateration, which involves the position

calculation of an object based on its distances from several points whose coordinates are known. This thesis explores the use of five trilateration methods for calculation of a quadcopter MAV's position in space. Among these five methods, three are numerical methods, while the remaining two are closed-form approaches to estimating the spatial position of the quadcopter while in flight. Each of the methods is evaluated in terms of accuracy and robustness when comparing their output position coordinates to that of the known position. The trilateration method that most adequately suits the quadcopter application is determined.

An ultrasonic RFID range measuring apparatus comprised of six stationary receivers is configured for use with the quadcopter and used to collect real-time distance measurements utilizing two transmitters located on the quadcopter. This system is integrated with and operated by a computer data acquisition system/program developed for the sole purpose of recording quadcopter data to be used for system identification.

Data collected while performing a series of flight tests provide a basis for the derivation of a system-level model describing the quadcopter's open loop responses to step inputs. Matlab's System Identification Toolbox is used to process this data and generate transfer functions that will serve as inputs to the control system model.

## 1.2 Literature Review

### *1.2.1 Numerical Trilateration Methods*

This study compares multiple trilateration methods that are used to calculate the quadcopter's position in space. Three of these methods employ a numerical approach – meaning that approximate solutions are arrived at by way of numerical calculation. Of the three methods used, two of them utilize error minimization principles.

#### *Simple Position Calculation*

The 'Simple' Method is one that is published by the manufacturer of the ultrasonic RFID sensor system used, Hexamite and is outlined in [1]. This method of position calculation requires that a minimum of three stationary receivers (also referred to as 'beacons') be oriented in a 'square' manner in which one receiver defines the origin of the coordinate system: (X, Y, Z) = (0, 0, 0). A simple calculation is used to approximate the position of the transmitter (also referred to as 'tag') in space with respect to the coordinate system defined by the orientation of the receivers. While computational demands are low for this method, the calculation requires specific orientation of the receivers and is less robust than other methods.

*Linear Least Squares*

Another trilateration method included in this study is the 'Linear Least Squares' approach outlined in [2] by Hereman and Murphy. This approach stems from the concept that the intersection of three spheres, whose radii are defined by the range measurements observed from each of three sensors, defines the location of the helicopter in space. The system of equations derived are linearized, and squared error is minimized to arrive at a solution that is only accurate when range measurements are exact. This method lacks the robustness necessary to produce an acceptable calculated when range measurements are approximate.

*Nonlinear Least Squares*

Also presented in [2] by Hereman and Murphy, the 'Nonlinear Least Squares' trilateration method attempts to accommodate a system exhibiting inherent errors (as all real systems do). An error-minimizing function is used to minimize squared errors iteratively for one set of range values. For the exemplary application presented by Hereman and Murphy, this method proves to be the most reliable for position calculation using error-ridden range values.

### *1.2.2 Closed-Form Trilateration Methods*

The remaining two trilateration methods can be classified as 'closed-form' in which a finite number of mathematical operations are used to arrive at a solution. These methods are preferred over most numerical methods as they minimize the time

needed for computation and are capable of being equivalently or more robust to range value inputs with error.

*Efficient Closed Form Position Estimation*

This approach minimizes computational load by exactly calculating the vector describing the position of an object whose range from 3 beacons is measured. The method that Manolakis outlines in [3] places a large emphasis on the height calculation portion as the application in the context of this document is the height calculation of an aerial vehicle independent from the barometric altimeter. We find that this particular method does not adequately fit the application of a MAV quadcopter for a number of reasons.

*Robot Localization Using Cayley-Menger Determinants*

In [4] a closed form position calculation method is presented that has been derived entirely geometrically – meaning that all calculations describe geometric relationships in a Euclidean space. Thomas and Ros approach the trilateration problem based on the understanding that the 3-receiver/1-transmitter system creates a simplex volume. The trilateration calculation derived from this principle concept is unlike the others presented in this thesis. It proves to be the most generally adequate for the application of the quadcopter MAV.

### *1.2.3 Kalman Filtering Concepts for Localization*

Not unlike most real-time data acquisition systems, data filtering is required for the data collected using the RFID sensor network developed for the purpose of this study. Kalman filtering concepts are adapted and applied similarly to that outlined in [7] by Shareef and Zhu. Kalman filters utilize state-space models in order to make data estimations and incorporate the consideration of known process and data measurement errors. [7] is used as a basis for developing a Kalman filter that is tuned for the quadcopter application presented in this study.

# CHAPTER II

# COMPARISON OF TRILATERATION METHODS

## 2.1 Trilateration Overview

Prior to diving into the specific derivations for the calculations that make up each of the trilateration methods compared in this thesis, it is important to understand the general trilateration concept as it applies to an MAV. For each of the trilateration methods that are evaluated, the receivers are assumed to be stationary. Each of these receivers measures the range (distance, in millimeters) to the target object (transmitter). The coordinates defining the location of each of these receivers are known. Depending on the trilateration method being used, measurements from between 3 and 6 of these receivers are used to determine the position of one transmitter in space. A minimum of 3 measurements are needed in order to determine the location of one transmitter. This requirement is dictated by the concept that the intersection of 3 spheres defines a point in space where the radii are the distances from each receiver to the transmitter and the center of each sphere is the respective location of a receiver. This is visualized in Figure 2.1.1. However, the Linear and Nonlinear Least Squares methods are accepting of as many distance measurements as are available.

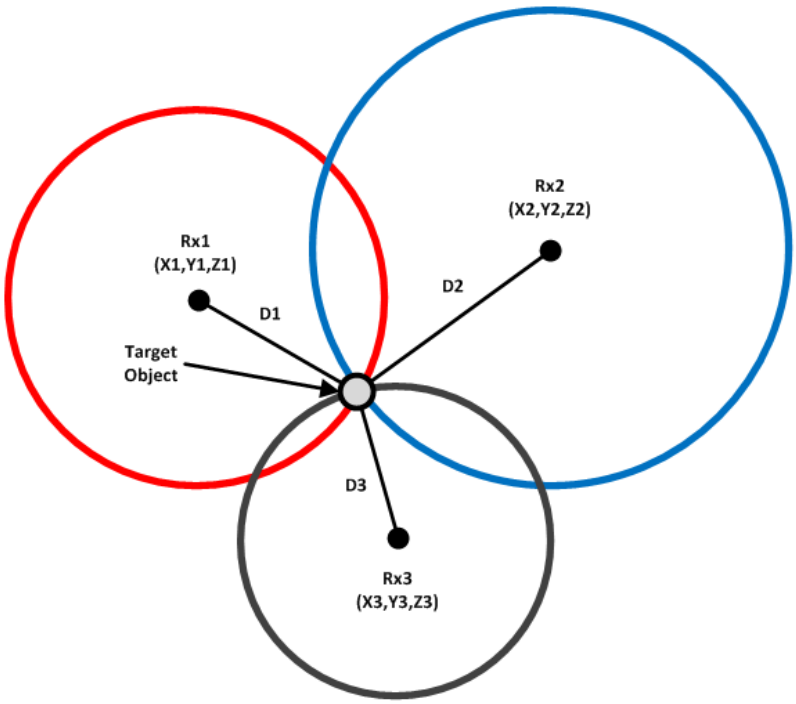



Figure 2.1: Trilateration without noise

The sphere surrounding any one of the receivers is defined mathematically in Equation 2.1:

$$(x - \ _i)^2 + (y - y_i)^2 + (z - z_i)^2 = r_i^2 \qquad (2.1)$$

where $i = 1, 2, \ldots, n$ for $n$ different receivers, $(x_i, y_i, z_i)$ is the location of receiver $i$, and $r_i$ is the range measurement received from receiver $i$ (also defines radius of sphere). Therefore, we have a system of $n$ nonlinear equations for which there is only one solution, $(x,y,z)$, if $n$ is equal to 3 or more and the range measurements from each receiver are exact. The relationships defined by this system of equations serves as a basis for most trilateration calculation methods.

## 2.2 Trilateration Calculations by Method

### *2.2.1 'Simple' Method*

The first of the five methods considered in this paper is referred to as the 'Simple Method' and, as its name implies, is a simplified approach to solving the position estimation problem of some target object in space. This method is presented in [1] and is a basic approach recommended for use with the Hexamite HX19 RFID/USID sensor system (for which more details will be provided in latter portions of this paper). This method imposes some constraints on the spatial configuration of the receivers being used:

1) All receivers are located on the same plane defined by z = 0.
2) The locations of the receivers are such that they form at least 3 vertices of a rectangle.
3) One receiver is located at the origin, (0,0,0).

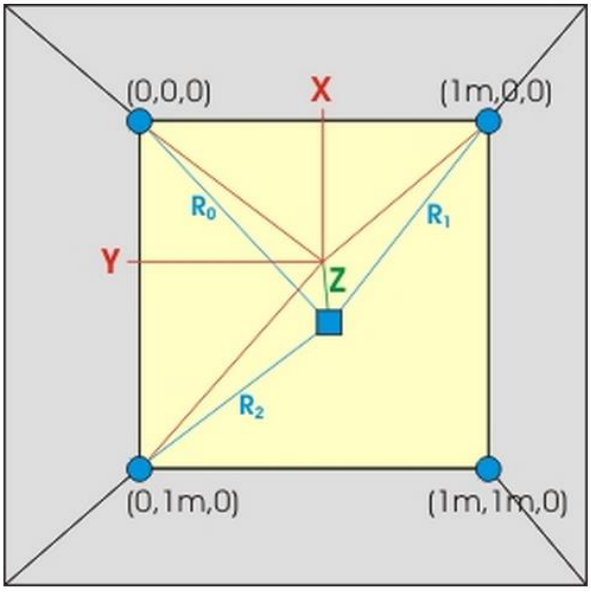



Figure 2.2: X-Y receiver layout for 'Simple' method [1]

Figure 2.2 shows an exemplary receiver layout where all receivers (4, total) are located at the vertices of a 1m x 1m square in the XY plane [1].

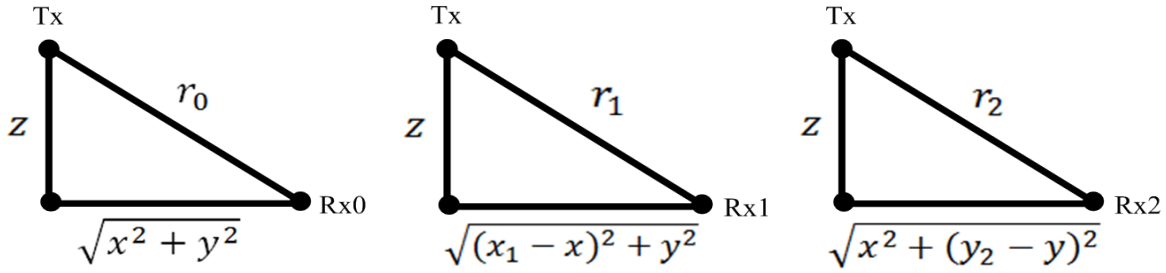



Figure 2.3: Geometric representation for each range measurement, $\boldsymbol{r_i}$

The three geometric relationships illustrated in Figure 2.3 can be manipulated to arrive at the following equations:

$$= ({r_0}^2 - {r_1}^2 + \ {}_1{}^2)/(2x_1) \tag{2.2}$$

$$y = ({r_0}^2 - {r_2}^2 + {y_2}^2)/(2y_2) \tag{2.3}$$

$$z^2 = {r_0}^2 - \ {}^2 + y^2 \tag{2.4}$$

This set of equations serves as a very simple way to calculate the spatial postion of a target object in space. However, this method makes no effort to account for error in the range measurement received from each receiver.

### *2.2.2 Linear Least Squares Method*

This trilateration approach, outlined in [2], suggests a calculation which builds on the concept of intersecting spheres. It begins with a system of equations in which each equation is defined by equation (2.1). For n receivers (whose location coordinates are known), there are $(n-1)$ equations that comprise the system. The number of receivers is not limited for this calculation but must be greater than or equal to four (this results from having three unknown variables and n-1 equations). The number six is used because it is the maximum number of receivers in our system available to provide range measurements. To begin, the system is linearized by adding a $j^{th}$ constraint – adding and subtracting $x_j$, $y_j$, and $z_j$:

$$(x - {}_j + {}_j - {}_i)^2 + (y - y_j + y_j - y_i)^2 + (z - z_j + z_j - z_i)^2 = {r_i}^2$$

After expanding and regrouping terms, this leads to

$$(x - {}_j)(x_i - {}_j) + (y - y_j)(y_i - y_j) + (z - z_j)(z_i - z_j)$$

$$= \frac{1}{2}\left[{r_j}^2 - {r_i}^2 + {d_{ij}}^2\right] = b_{ij}.$$

where

$$d_{ij} = \sqrt{(x_i - {}_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \tag{2.5}$$

is the distance between receivers $Rx_i$ and $Rx_j$ and $b_{ij}$ is a calculated constant. Any of the six available receivers can be used to serve as the linearizing constraint. The first receiver is choosen, $i = 1$. This leaves a system of five equations:

$$(x - {}_1)(x_2 - {}_1) + (y-1)(y_2 - y_1) + (z - z_1)(z_2 - z_1) = b_{21}$$

$$(x - {}_1)(x_3 - {}_1) + (y-1)(y_3 - y_1) + (z - z_1)(z_3 - z_1) = b_{31}$$

$$\vdots$$

$$(x - {}_1)(x_6 - {}_1) + (y-1)(y_6 - y_1) + (z - z_1)(z_6 - z_1) = b_{61}.$$

This can be represented in matrix form:

$$\mathbf{A}\vec{x} = \vec{b}\,, \tag{2.6}$$

where

$$\mathbf{A} = \begin{bmatrix} {}_2 - {}_1 & y_2 - y_1 & z_2 - z_1 \\ \vdots & \vdots & \vdots \\ {}_6 - {}_1 & {}_6 - {}_1 & {}_6 - {}_1 \end{bmatrix},$$

$$^{\rightarrow} = \begin{bmatrix} - {}_1 \\ y - y_1 \\ z - z_1 \end{bmatrix},$$

$$\vec{b} = \begin{bmatrix} b_{21} \\ \vdots \\ b_{61} \end{bmatrix}.$$

A solution can be obtained by solving the linear system presented above; however, the solution is only valuable if all range measurements are exact. As is true with any real-world system, some error is present in the range measure meants collected from each receiver. In order to minimize the effect of this error the squared error is minimized:

$$S = \vec{r}^T\vec{r} = (\vec{b} - \mathbf{A}\vec{x})^T(\vec{b} - \mathbf{A}\vec{x})$$

which leads to the normal equation solved for the estimated position vector, $\vec{x}$:

0 (2.7)

This completes the overview of the Linear Least Squares trilateration method. In [2], Hereman and Murphy comment that while being an improvement upon the linear system calculation (Equation 2.5) this method is still 'unacceptable' for use within the system presented.

### *2.2.3 Nonlinear Least Squares Method*

The third trilateration method is one that is designed to be more robust than the previous two methods in the sense that it can more adequately handle range measurements that are error ridden [2]. It is recursive in nature and requires an interative loop when implemented using analytical software, such as Matlab, to converge on an acceptable solution. Again, the max number of receivers available is six, and this is that number that is used for analysis with this method. However, as few as four receivers could be used, and there is no upper limit to the number of receivers that could be used. Following, is the presentatation of the Nonlinear Least Squares trilateration method provided in [2].

To begin, define the error looking to be minimized as

$$F(x,y,z) = \sum_{i=1}^{6} (\hat{r}_i - r_i)^2 = \sum_{i=1}^{6} f_i(x,y,z)^2,$$

where $\hat{r}_i$ denotes the exact distance from receiver i to the target object and $r_i$ is the measured distance (includes error) and

$$f_i(x,y,z) = \hat{r}_i - r_i = \sqrt{(x - \ _i)^2 + (y - y_i)^2 + (z - z_i)^2} - r_{i.} \qquad (2.8)$$

$f_i$ is the error calculated for one range measurement based on the estimated coordinates of the target object $(\mathrm{x.y.z})$. Therefore, for six receivers, $\mathrm{F(x,y,z)}$ is the sum of six squared errors (each calculated as ${f_i}^2$).

Minimizing $\mathrm{F(x,y,z)}$ requires that the derivative be taken with respect to $\mathrm{x,y,z}$, respectively. The result of this is

$$\frac{\partial \mathrm{F}}{\partial \mathrm{x}} = 2\sum_{\mathrm{i}=1}^{6} f_i \frac{\partial f_\mathrm{i}}{\partial \mathrm{x}},$$

$$\frac{\partial F}{\partial y} = 2\sum_{i=1}^{6} f_i \frac{\partial f_i}{\partial y},$$

$$\frac{\partial F}{\partial z} = 2\sum_{i=1}^{6} f_i \frac{\partial f_i}{\partial z}.$$

Vectors $\vec{\mathrm{f}}$ and $\vec{\mathrm{g}}$ are added as well a the Jacobian matrix, J, introduced:

$$\mathrm{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_6}{\partial x} & \frac{\partial f_6}{\partial y} & \frac{\partial f_6}{\partial z} \end{bmatrix}, \qquad \vec{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_6 \end{bmatrix}, \qquad \vec{\mathrm{g}} = \begin{bmatrix} \frac{\partial \mathrm{F}}{\partial \mathrm{x}} \\ \frac{\partial \mathrm{F}}{\partial \mathrm{y}} \\ \frac{\partial \mathrm{F}}{\partial \mathrm{z}} \end{bmatrix},$$

and

$$\vec{\mathrm{g}} = 2\mathrm{J}^{\mathrm{T}}\,\vec{\mathrm{f}}.$$

Using Newton's Method for finding minima gives

$$\vec{\mathrm{R}}_{\mathrm{new}} = \vec{\mathrm{R}}_{\mathrm{old}} - \left(\mathrm{J}_{\mathrm{old}}{}^{\mathrm{T}}\mathrm{J}_{\mathrm{old}}\right)^{-1}\mathrm{J}_{\mathrm{old}}{}^{\mathrm{T}}\vec{\mathrm{f}}_{\mathrm{old}},$$

where $\vec{R} = (x, y, z)$ and $()_{old}$ and $()_{new}$ denote calculations for previous and current iterations, respectively, and

$$J^T J = \begin{bmatrix} \sum_{i=1}^{6} \frac{(x-x_i)^2}{(f_i-r_i)^2} & \sum_{i=1}^{6} \frac{(y-y_i)(x-x_i)}{(f_i-r_i)^2} & \sum_{i=1}^{6} \frac{(z-z_i)(x-x_i)}{(f_i-r_i)^2} \\ \sum_{i=1}^{6} \frac{(x-x_i)(y-y_i)}{(f_i-r_i)^2} & \sum_{i=1}^{6} \frac{(y-y_i)^2}{(f_i-r_i)^2} & \sum_{i=1}^{6} \frac{(z-z_i)(y-y_i)}{(f_i-r_i)^2} \\ \sum_{i=1}^{6} \frac{(x-x_i)(z-z_i)}{(f_i-r_i)^2} & \sum_{i=1}^{6} \frac{(y-y_i)(z-z_i)}{(f_i-r_i)^2} & \sum_{i=1}^{6} \frac{(z-z_i)^2}{(f_i-r_i)^2} \end{bmatrix}, \tag{2.9}$$

$$J^T \vec{f} = \begin{bmatrix} \sum_{i=1}^{6} \frac{(x-x_i)f_i}{(f_i+r_i)} \\ \sum_{i=1}^{6} \frac{(y-y_i)f_i}{(f_i+r_i)} \\ \sum_{i=1}^{6} \frac{(z-z_i)f_i}{(f_i+r_i)} \end{bmatrix}. \tag{2.10}$$

Ideally, new approximations are iteratively generated for $\vec{R}$ until a solution is reached. However, it is likely that no exact solution is arrived at by way of iterative calculation. For this reason, the difference in magnitude of position vectors is evaluated for each iteration, and if the difference in this magnitude is reasonably small for consecutive iterations (in this case $\left|\vec{R}_{new}\right| - \left|\vec{R}_{old}\right| \leq 1 * 10^{-6}$) then the $\vec{R}_{new}$ position coordinates are used for the position estimation.

### *2.2.4 Efficient Closed Form Position Estimation*

In [3], Manolakis presents a position calculation method that is intended to be 'exact, explicit, and computationally efficient.' By isolating calculated portions that pertain to unchanging known values (such as receiver coordinates and the distance between receivers), the computational magnitude is minimized for each set of range measurement data. Presented next is the high-level summary of calculations as they are applied to the range sensor system on hand. This method is only accepting of three range measurements.

We begin with the familiar equation

$$R_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} \tag{2.11}$$

where $R_i$ now refers to the distance measurement acquired from receiver *i*. Now we define ${S_i}^2$ as

$${S_i}^2 = {x_i}^2 + {y_i}^2 + {z_i}^2.$$

Squaring Equation 2.11 and substituting ${S_i}^2$ gives

$$R_i^2 = S_i^2 - 2x_i x - 2y_i y - 2z_i z + x^2 + y^2 + z^2, \tag{2.12}$$

for *i* = 1,2,3. By subtracting $R_1^2$ from $R_i^2$ we arrive at

$$R_i^2 - R_1^2 = S_2^2 - S_1^2 - 2x_{i1}x - 2y_{i1}y - 2z_{i1}z$$

for *i* = 2,3 and $x_{i1} = x_i - x_1,\ y_{i1} = y_i - y_1,$ and $z_{i1} = z_i - z_1.$

Then the following are defined:

$$\mathbf{W}\mathbf{r}_\mathbf{h} = \boldsymbol{\beta} - \mathbf{d}\mathbf{z},$$

where

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_2^2 \\ \beta_3^2 \end{bmatrix},$$

$$\mathbf{d} = \begin{bmatrix} z_{21} \\ z_{31} \end{bmatrix},$$

$$\mathbf{r_h} = \begin{bmatrix} \\ y \end{bmatrix},$$

$$\mathbf{W} = \begin{bmatrix} {}^{21} & y_{21} \\ x_{31} & y_{31} \end{bmatrix},$$

and

$$\beta_i^2 = (R_1^2 - R_i^2 - S_1^2 + S_i^2)/2.$$

We now define the quadcopter's horizontal position, $\mathbf{r_h}$, as

$$\mathbf{r_h} = \mathbf{W}^{-1}(\boldsymbol{\beta} - \mathbf{d}z), \tag{2.13}$$

$$\mathbf{r_{h1}} = [x_1 \quad y_1]^T, \tag{2.14}$$

and rewrite equation (2.4.4b) as

$$R_1^2 = S_1^2 - 2\mathbf{r_{h1}}^T\mathbf{r_h} - 2z_1 z + \mathbf{r_h}^T\mathbf{r_h} + z^2. \tag{2.15}$$

Equations (2.13) and (2.14) can be substituted into (2.15) to yield the quadratic equation

$$az^2 + bz + c = 0 \tag{2.16}$$

where

$$a = 1 + \mathbf{d}^T\mathbf{W}^{-T}\mathbf{W}^{-1}\mathbf{d},$$

$$b = 2\mathbf{r}_{\mathbf{h1}}^T\mathbf{W}^{-1}\mathbf{d} - 2z_1 - 2\mathbf{d}^T\mathbf{W}^{-T}\mathbf{W}^{-1}\boldsymbol{\beta},$$

$$c = S_1^2 - R_1^2 + \boldsymbol{\beta}^T\mathbf{W}^{-T}\mathbf{W}^{-1}\boldsymbol{\beta} - 2\mathbf{r}_{\mathbf{h1}}^T\mathbf{W}^{-1}\boldsymbol{\beta}.$$

This allows us to solve for z:

$$z = \frac{-b \pm (b^2 - 4ac)^{\frac{1}{2}}}{2a}, \tag{2.17}$$

and, by plugging z back into equation (2.13), x and y can be calculated:

$$x = r_h(1), \quad y = r_h(2).$$

It should be noted that W, d, and $r_{h1}$ are calculated from the stations' locations coordinates only (one time calculation). Only β is dependent on the range measurements from each receiver. This aids substantially in reducing computational complexity [3].

### *2.2.5 Trilateration Using Cayley-Menger Determinants*

The fifth, and final, method for estimating the position of the quadcopter discussed in this paper takes an altogether different approach. It is derived entirely geometrically and allows the locations of three receivers and that of the target object to define the vertices of a simplex volume in a Euclidean space. This method, outlined in [4] by Thomas and Ros, avoids employing algebraic manipulation of the governing system of equations. Instead, all mathematical manipulations have geometric implications making it a somewhat more tangible trilateration approach. Cayley-Menger determinants are utilized to define the relationship between the volume of the simplex and the three range measurements.

Figure 2.4 shows the locations as the three receivers as $p_1, p_2,$ and $p_3$ while the unknown location of the quadcopter is represented as $p_4$; $l_1, l_2,$ and $l_3$ are the range measurements corresponding to receivers one, two, and three, respectively.

The derivation in its entirety is not shown here but can be found in [4]. The primary calculations implemented in the Matlab program are provided.

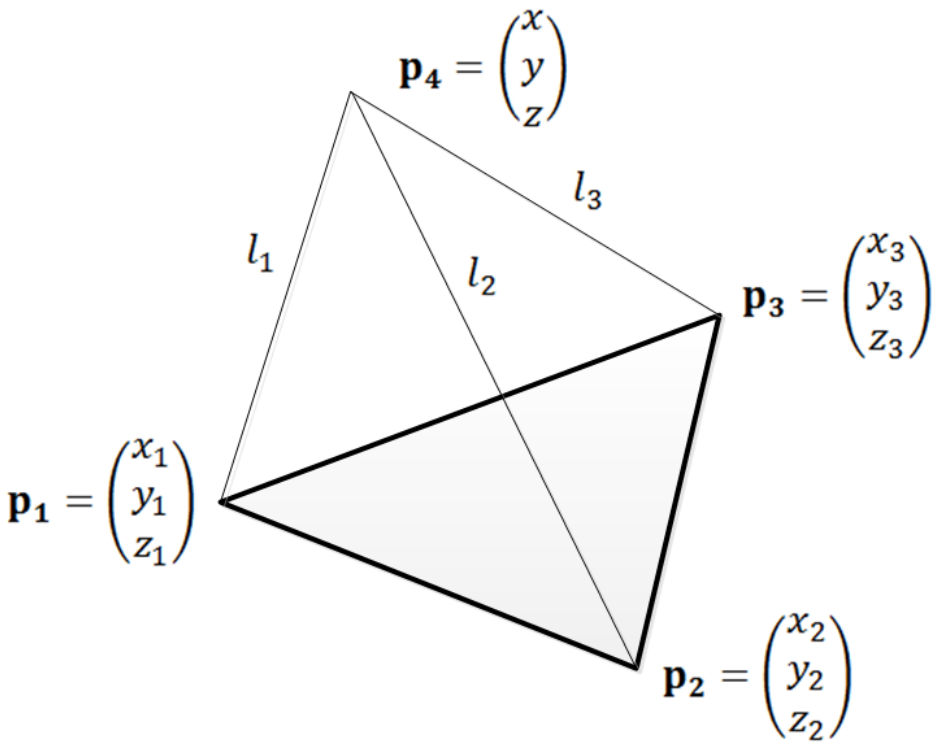



Figure 2.4: Simplex defined by receiver and quadcopter locations

To begin, once again we have a system of equations describing the intersection of three spheres:

$$\left.\begin{aligned} (x - {}_1)^2 + (y - y_1)^2 + (z - z_1)^2 &= l_1^2 \\ (x - {}_2)^2 + (y - y_2)^2 + (z - z_2)^2 &= l_2^2 \\ (x - {}_3)^2 + (y - y_3)^2 + (z - z_3)^2 &= l_3^2 \end{aligned}\right\}. \tag{2.18}$$

The number of points that make of the simplex volume in 3D space is $(\mathrm{n} = 4)$. The Cayley-Menger bideterminant of two sequences of n points $[\mathrm{p}_1, \ldots, \mathrm{p}_\mathrm{n}]$ and $[\mathrm{q}_1, \ldots, \mathrm{q}_\mathrm{n}]$ is

$$\mathrm{D}(\mathrm{p}_1, \ldots, \mathrm{p}_\mathrm{n};\ \mathrm{q}_1, \ldots, \mathrm{q}_\mathrm{n})$$

$$= 2\left(\frac{-1}{2}\right)^n \begin{vmatrix} 0 & 1 & 1 & \cdots & 1 \\ 1 & D(p_1,q_1) & D(p_1,q_2) & \cdots & D(p_1,q_n) \\ 1 & D(p_2,q_1) & D(p_2,q_2) & \cdots & D(p_2,q_n) \\ \vdots & \vdots & \vdots & \ddots & D(p_3,q_n) \\ 1 & D(p_n,q_1) & D(p_n,q_2) & \cdots & D(p_n,q_n) \end{vmatrix} \tag{2.19}$$

where $D(p_i, q_j)$ represents the squared distance between points $p_i$ and $q_j$. For cases when the two sequences of points are the same, $D(p_1, \ldots, p_n;\ p_1, \ldots, p_n)$ is the Cayley-Menger determinant and is defined for $(n = 3)$ as

$$D(p_1, p_2, p_3;\ p_1, p_2, p_3) = D(p_1, p_2, p_3)$$
$$= \|(p_2 - p_1) \times (p_3 - p_1)\|^2 \tag{2.19}$$

Let us define vectors $\vec{v}_1$ and $\vec{v}_2$ as

$\vec{v}_1 = (p_2 - p_1)$ and $\vec{v}_2 = (p_3 - p_1)$.

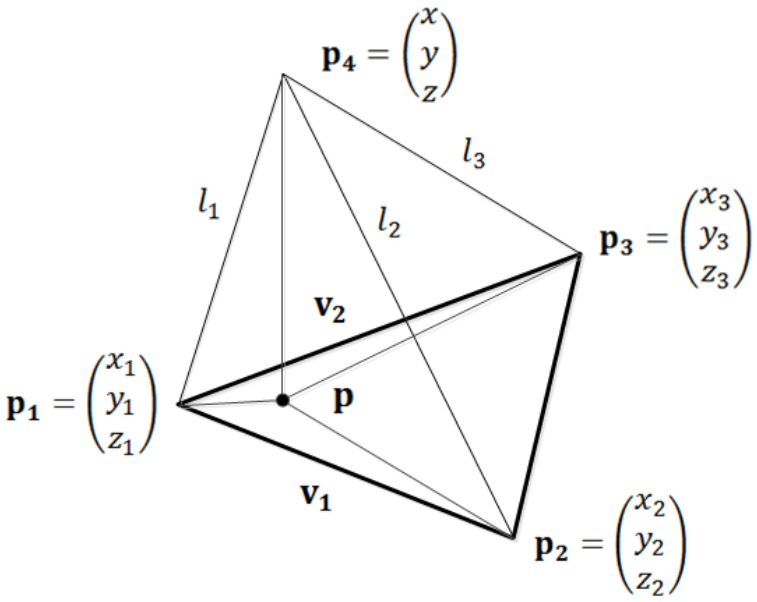



Figure 2.5: Simplex volume with added vectors $\vec{\mathbf{v}}_\mathbf{1}$ and $\vec{\mathbf{v}}_\mathbf{2}$ and point $\mathbf{p}$

Additionally, point p is defined as the orthogonal projection of $p_4$ onto the base of the tetrahedron (defined by points $p_1$, $p_2$, and $p_3$). These are shown in Figure 2.5. Also,

$$p = p_1 + k_1\vec{v}_1 + k_2\vec{v}_2 \tag{2.20}$$

where

$$k_1 = \frac{D(p_1, p_2, p_3;\ p_1, p_3, p_4)}{D(p_1, p_2, p_3)}$$

and

$$k_2 = \frac{D(p_1, p_2, p_3;\ p_1, p_2, p_4)}{D(p_1, p_2, p_3)}$$

Furthermore, the height (z-component) can be included by

$$p_4 = p \pm k_3(\vec{v}_1 \times \vec{v}_2) \tag{2.21}$$

where

$$k_3 = \frac{+\sqrt{D(p_1, p_2, p_3, p_4)}}{D(p_1, p_2, p_3)}$$

Using the definition for the Cayley-Menger bideterminant (or determinant of sequence if points are the same - i.e.: $D(p_1, p_2, p_3) = D(p_1, p_2, p_3; p_1, p_2, p_3)$), $k_1, k_2$, and $k_3$ are calculated as

$$k_1 = \frac{2\left(\frac{-1}{2}\right)^3 \begin{vmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & D(p_1,p_3) & l_1^2 \\ 1 & D(p_2,p_1) & 0 & l_2^2 \\ 1 & D(p_3,p_1) & 0 & l_3^2 \end{vmatrix}}{\|(p_2-p_1)\times(p_3-p_1)\|^2} \tag{2.22}$$

$$k_2 = \frac{2\left(\frac{-1}{2}\right)^3 \begin{vmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & D(p_1,p_2) & l_1^2 \\ 1 & D(p_2,p_1) & 0 & l_2^2 \\ 1 & D(p_3,p_1) & 0 & l_3^2 \end{vmatrix}}{\|(p_2-p_1)\times(p_3-p_1)\|^2} \tag{2.23}$$

$$k_3 = \frac{2\left(\frac{-1}{2}\right)^4 \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & D(p_1,p_2) & D(p_1,p_3) & l_1^2 \\ 1 & D(p_2,p_1) & 0 & D(p_2,p_3) & l_2^2 \\ 1 & D(p_3,p_1) & D(p_3,p_2) & 0 & l_3^2 \\ 1 & l_1^2 & l_2^2 & l_3^2 & 0 \end{vmatrix}}{\|(p_2-p_1)\times(p_3-p_1)\|^2} \tag{2.24}$$

This allows for the computation of the coordinates describing the location of point $p_4$ as

$$p_4 = p + k_3(\vec{v}_1 \times \vec{v}_2) = (x, y, z) \tag{2.25}$$

This method of trilateration is comparable to the Efficient Closed Form Position Estimation method in its computational efficiency but proves to be more robust in other ways. This is investigated later in this study.

## 2.3 Monte Carlo Simulation and Error Analysis

Any of the aforementioned trilateration methods provide equally acceptable solutions if no error exists within the quadcopter position estimation system. However, no real-world system operates in an error-free manner. While some general error analyses have been presented pertaining to the trilateration methods examined in [2], [3], and [4], the purpose of this portion of the study is to evaluate the effectivenness of each method as it applies to the quadcopter application. In this section, four of the five different trilateration methods outlined in the previous section are evaluated by way of conducting a Monte Carlo Simulation.

For this simulation system parameters are set to generally reflect the characteristics and parameters of the ultrasonic RFID sensor system that is used with the quadcopter. While the details pertaining to this system will be explained in greater depth later in this paper, it is important to note that this simulation is generally representative of the test system used for this study as the intention of this comparison is to determine which of these methods best suits the quadcopter application.

### *2.3.1 Simulation Overview*

Monte Carlo simulations are often used to understand how noise-inducing factors of a system affect the output of the system. Often systems will have many potential sources of variation. The effect of these sources, individually and altogether, need to be fully understood in order to know the system will always perform the way it is intended. This is known as designing for robustness – a major primary concept in Design for Six Sigma practices. For the purpose of our quadcopter localization system, a Monte Carlo simulation is conducted in order to observe the effects of error in the range measurements made by each receiver on the position estimation calculated using each trilateration method. Although there are certainly other sources of error in the quadcopter system, the error in range measurements from the receivers tend to be dominant and are the most capable of impacting the proper functioning of the system. For this reason, only range measurement error is simulated as it affects the calculated position.

Beginning with one point, $\hat{\mathbf{p}} = (\mathrm{x}, \mathrm{y}, \mathrm{z})$, in space whose coordinates are randomly generated and known, the exact distance between $\hat{\mathbf{p}}$ and each receiver is calculated as

$$\hat{R}_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} \tag{2.26}$$

and

$$R_i = \hat{R}_i + \Delta R_i \tag{2.27}$$

where $_i, y_i$ and $z_i$ are the location coordinates of receiver $i$ and $\Delta R_i$ is the error in the range measurement for $i = 1,2, \ldots, n$ for $n$ receivers. In this simulation, the measurement error is assumed to be normally distributed with a mean of zero. This is typical when using the Monte Carlo method. Approximate values for $_i, y_i$ and $z_i$ are shown in Table 2.1 and mimic the coordinates of the real system.

Table 2.1: Approximate coordinates for all receivers

| Receiver #, $i$ | $_i$, (mm) | $y_i$, (mm) | $z_i$, (mm) |
|---|---|---|---|
| 1 | -325 | 158.572 | 22 |
| 2 | 325 | 158.572 | 22 |
| 3 | 0 | -375.278 | 22 |
| 4 | 0 | 750.552 | 0 |
| 5 | -650 | -375.278 | 0 |
| 6 | 650 | -375.278 | 0 |

$\Delta R_i$ is defined as a random variable whose distribution is normal and is characterized by a standard deviation of $\sigma_i$. The variation observed in range measurements using the Hexamite HX19R receivers is relatively small (on the order of 1-2 mm), but a conservative value is used here as error can increase over prolonged use. For this simulation $\sigma_i = 5$mm is used for all receivers ($i = 1,2, \ldots,6$). If some receivers were known to exhibit more or less variation in range measurements than

the others then $\sigma_i$ could be valuated according to the individual receiver's performance characteristics.

Now that $R_i$ is available for all receivers, each trilateration method is used to again calculate the position based on the range measurements that include added error. This is done with all methods except the 'Simple' method. The reason for this is that it is largely redundant as the Linear Least Square method is derived in a very similar manner to that of the 'Simple' method. Thus,

$$\mathbf{p}_{LS} = f_{LS}(R_1, R_2, R_3, R_4, R_5, R_6),$$

$$\mathbf{p}_{NLS} = f_{NLS}(R_1, R_2, R_3, R_4, R_5, R_6),$$

$$\mathbf{p}_{CFP} = f_{CFP}(R_4, R_5, R_6),$$

$$\mathbf{p}_{CMD} = f_{CMD}(R_4, R_5, R_6).$$

where $\mathbf{p}_{LS}, \mathbf{p}_{NLS}, \mathbf{p}_{CFP}$, and $\mathbf{p}_{CMD}$ are the calculated positions using the Linear Least Squares, Nonlinear Least Squares, Efficient Closed Form, and Cayley-Menger Determinant methods, respectively. The reason for using range measurements $R_4, R_5$, and $R_6$ for the latter two methods is that these are the outermost receiver locations given the triangular configuration shown in Figure 2.6. Using these receivers helps to minimize sensitivity to error by spanning a larger area.

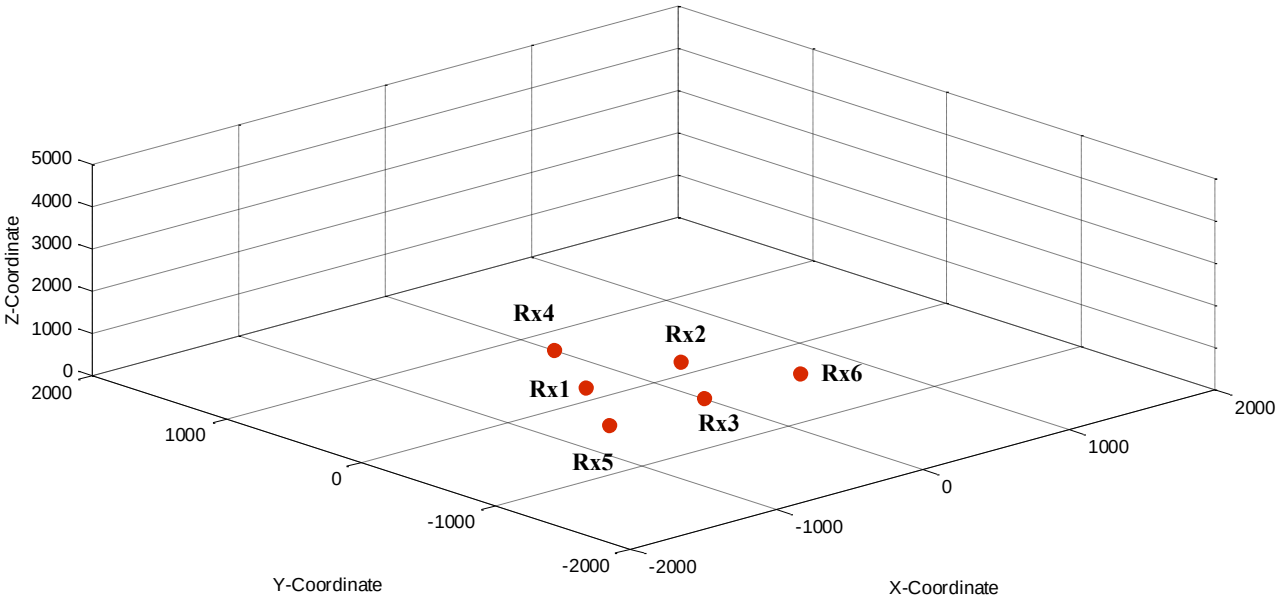



Figure 2.6: Receiver configuration for simulation

In order to find the error associated with each position estimation, the difference must be taken:

$$\vec{e}_{LS} = \mathbf{p}_{LS} - \hat{\mathbf{p}}$$

$$\vec{e}_{NLS} = \mathbf{p}_{NLS} - \hat{\mathbf{p}}$$

$$\vec{e}_{CFP} = \mathbf{p}_{CFP} - \hat{\mathbf{p}}$$

$$\vec{e}_{CMD} = \mathbf{p}_{CMD} - \hat{\mathbf{p}}$$

Knowing the amount of error in in the position estimation for one set of error-ridden range measurements is not all that helpful, and it is not an effective way of comparing trilateration methods. Performing a Monte Carlo simulation involves generating sufficiently many points such that the mean error is zero for all points within any localized volume of the 3D space being considered.

Instead of simulating for a volume, points are randomly generated in an X-Y plane where $z$ is constant. An example of this is shown in Figure 2.8. These points could also be generated in space, however, it would require that many more points be used for the simulation. It is found to be more appropriate to perform the simulation at multiple values of $z$ and compare those results individually. These points are the exact known locations of the quadcopter in space ($\hat{\mathbf{p}}$). In order to analyze the results of the simulation in a meaningful way, it is necessary to lump all these data (actual positions and estimated positions) into 2D bins, called pixels, such that each 2D segment of data can be statistically analyzed. This concept is visualized in Figure 2.7. A bin size of 1000mm is used for this simulation (pixel = 1000mm x 1000mm).

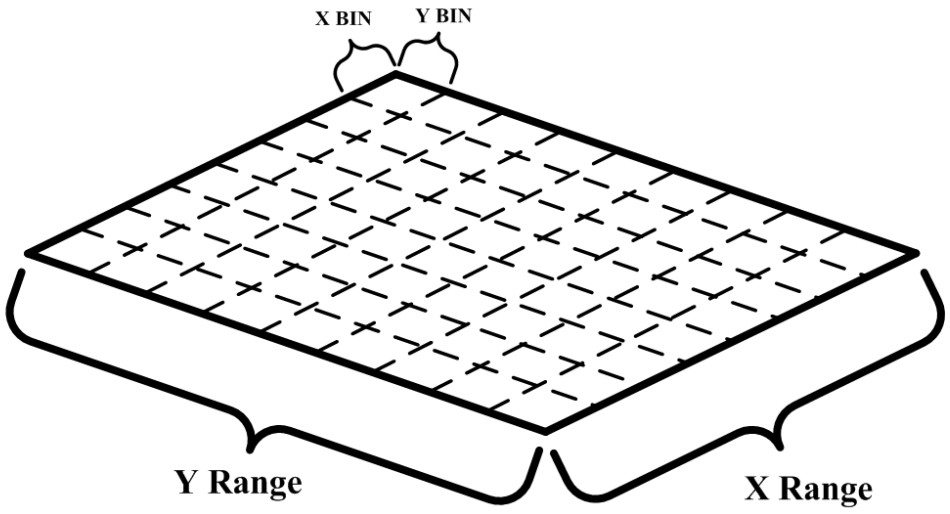



Figure 2.7: 2D Bins on X-Y plane at Z = constant

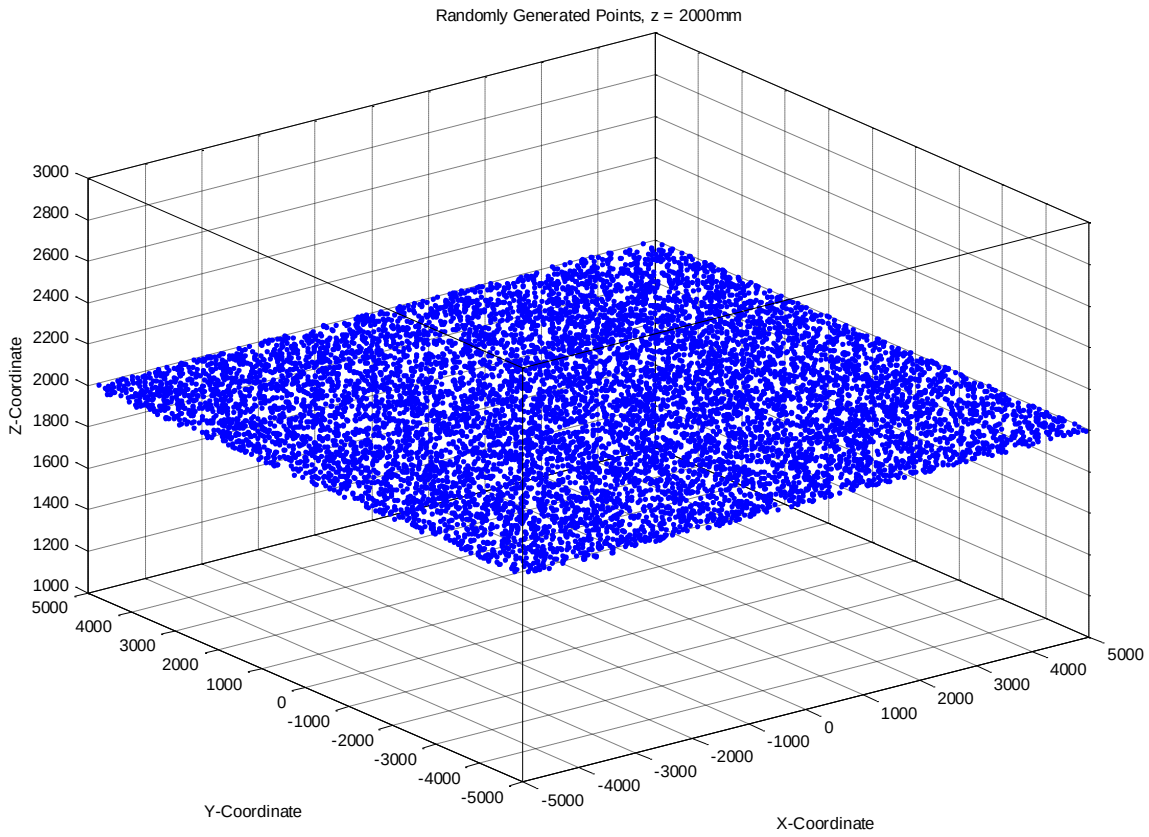



Figure 2.8: 10,000 points randomly generated in XY plane where Z = 2000mm

The assortment of histogram plots shown in Figure 2.9 serve as an example for how added-error values are normally distributed (with mean of zero) for each receiver.

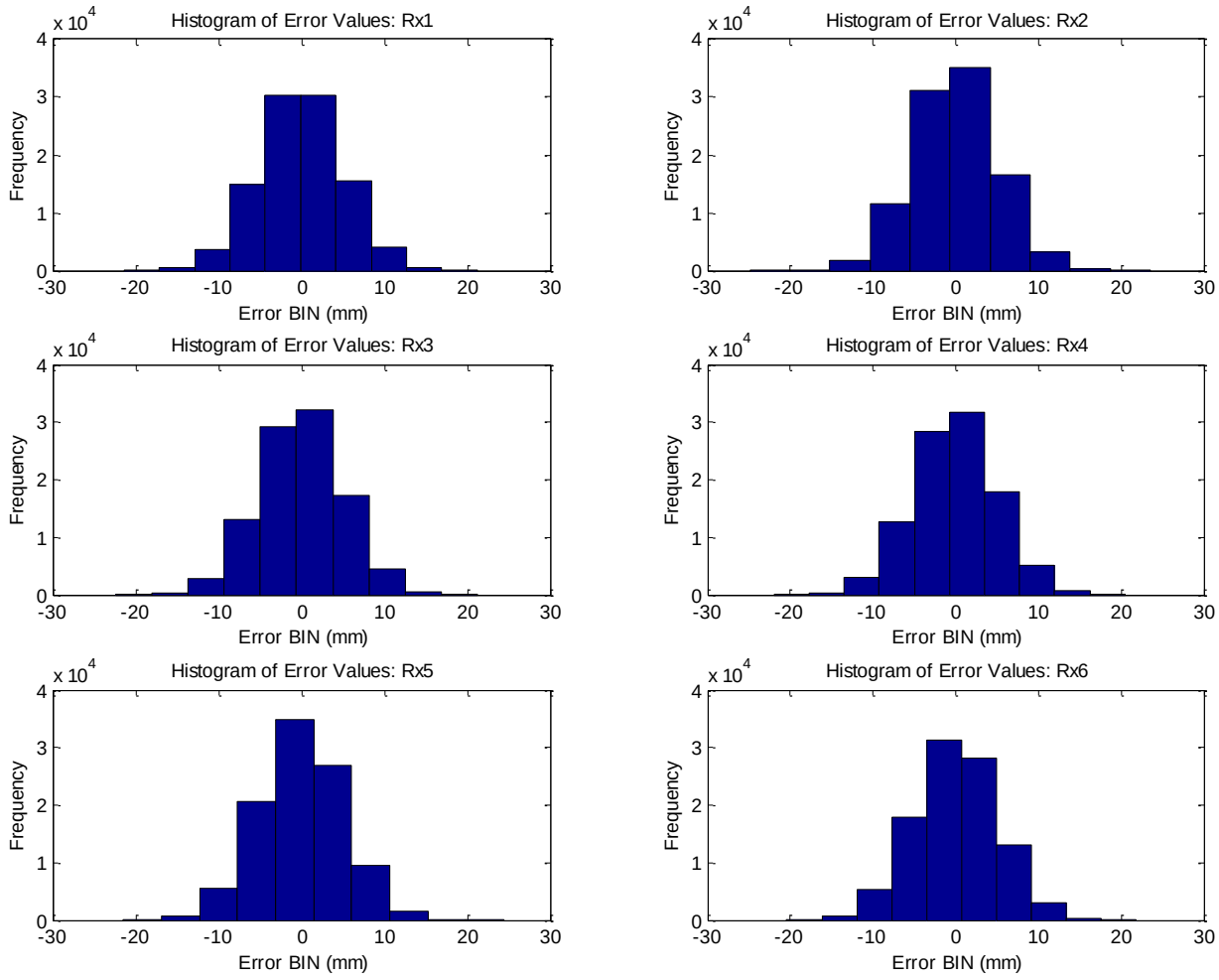



Figure 2.9: Distribution of measurement errors for each receiver

Once all data points have been sorted into the aformentioned pixels (2D bins), specific metrics can be obtained for each pixel as well as for the whole plane of points. The metrics that are used to evauluate the average error in each directions $(x, y, z)$ are defined as

$$\mathbf{E}_{x,avg}(i,j) = \left(\frac{\sum_{k=1}^{n} e_{x,k}}{n}\right)_{i,j},$$

$$\mathbf{E}_{y,avg}(i,j) = \left(\frac{\sum_{k=1}^{n} e_{y,k}}{n}\right)_{i,j}, \tag{2.28}$$

$$\mathbf{E}_{z,avg}(i,j) = \left(\frac{\sum_{k=1}^{n} e_{z,k}}{n}\right)_{i,j}.$$

The same can be done for mean squared error in each direction:

$$\mathbf{E}_{x,MSE}(i,j) = \left(\frac{\sum_{k=1}^{n} {e_{x,k}}^2}{n}\right)_{i,j},$$

$$\mathbf{E}_{y,MSE}(i,j) = \left(\frac{\sum_{k=1}^{n} {e_{y,k}}^2}{n}\right)_{i,j}, \tag{2.29}$$

$$\mathbf{E}_{z,MSE}(i,j) = \left(\frac{\sum_{k=1}^{n} {e_{z,k}}^2}{n}\right)_{i,j}.$$

Total MSE for a pixel is

$$\mathbf{E}_{MSE}(i,j) = \left(\frac{\sum_{k=1}^{n} {e_{x,k}}^2}{n}\right)_{i,j} + \left(\frac{\sum_{k=1}^{n} {e_{y,k}}^2}{n}\right)_{i,j} + \left(\frac{\sum_{k=1}^{n} {e_{z,k}}^2}{n}\right)_{i,j}$$

$$= \mathbf{E}_{x,MSE}(i,j) + \mathbf{E}_{y,MSE}(i,j) + \mathbf{E}_{z,MSE}(i,j). \tag{2.30}$$

where $i$ and $j$ specify the indices of the the pixel of interest within the XY plane of pixels, $n$ is the number of points contained by that pixel, and $e_{x,k}, e_{y,k}$, and $e_{z,k}$ are the $x, y$, and $z$ components of the error vector $\vec{e}_k$. Furthermore, these metrics can be used to obtain similar metrics for the whole set of points plotted in the XY plane defined.

This Monte Carlo simulation is performed for the parameters and respective values summarized in Table 2.2. The Matlab program developed to perform this simulation can be found in Appendix B.

Table 2.2: Summary of simulation parameters

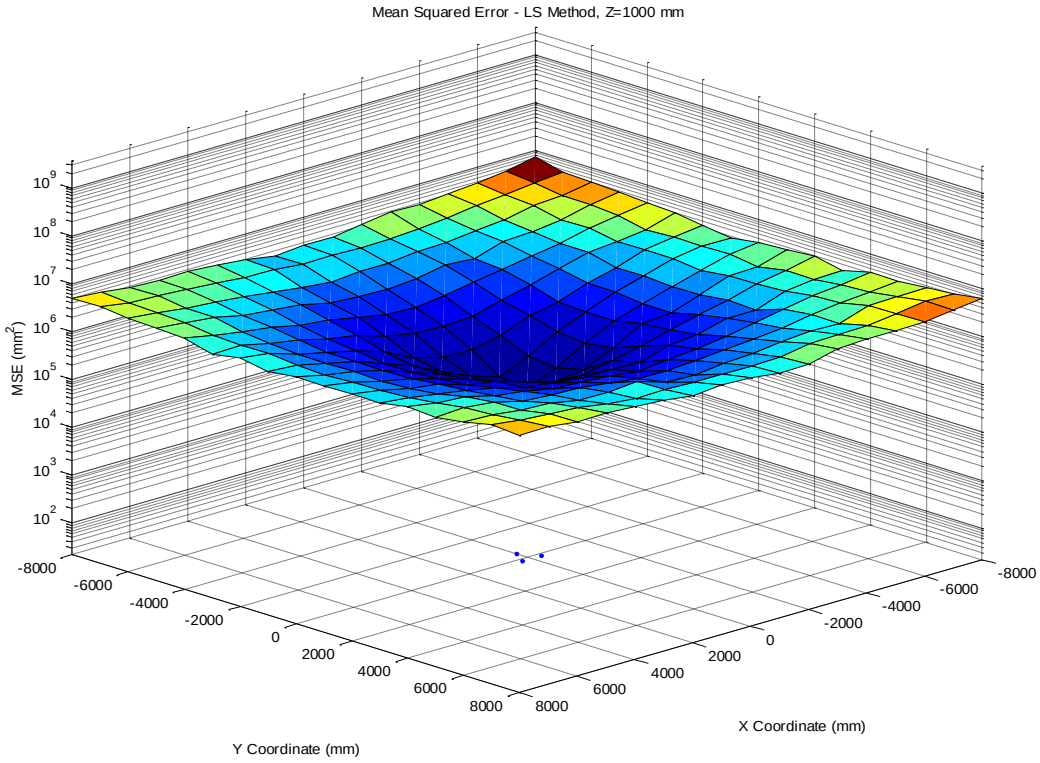
| Parameters for Monte Carlo Simulation | |
|---|---|
| X-Span: | -8000 to +8000 mm |
| Y-Span: | -8000 to +8000 mm |
| Z-Value (height): | 1000, 2000, 3000, 4000, 5000, and 7000 mm |
| Bin Size: | 1000 mm |
| Number of Points: | 100,000 |
| Std Dev of Rx Error: | 5 mm |

### *2.3.2 Simulation Results*

All parameters are held constant for these simulations except for the height parameter. This parameter dictates how many simulations must be performed. In this case, six simulations are run, and the same metrics are used to analyze the results of each simulation. Surface, contour, and quiver plots are shown for the simulations when Z = 1000mm, 3000mm, and 5000mm. By looking at Figures 2.10 – 2.24 it can be seen that the four trilateration methods simulated perform very differently.

There are multiple eligible approaches evaluating the performance of each of these trilateration methods individually. The interest of this study is to establish which method will be the most appropriate and effective for the application of the quadcopter UAV. Several general performance metrics are commonly used when

evaluating trilateration methods such as those simulated in this study. Among these metrics are accuracy, precision, complexity, and robustness [5]. The two that are used for analyzing the simulation results are accuracy and precision. Accuracy within this context is defined as location error (where smaller location error equals a higher accuracy). This metric translates as being a measure of systematic location bias induced by the position calculation method used. Average error in each direction (*x, y,* and *z*) is used to quantify this for each method (equation 2.3.1g). We would expect the average error to be near zero in each localized space in the event that there is no biasing. Precision is considered to measure the variation in results. In the case of the position calculation, we can consider precision to be the variation in distance error between the estimated position and the actual position for a constant actual position. The mathematical metric used for this analysis is mean squared error (MSE). The MSE can be calculated in each direction (*x, y,* and *z*) or as a total squared distance – both are used here.



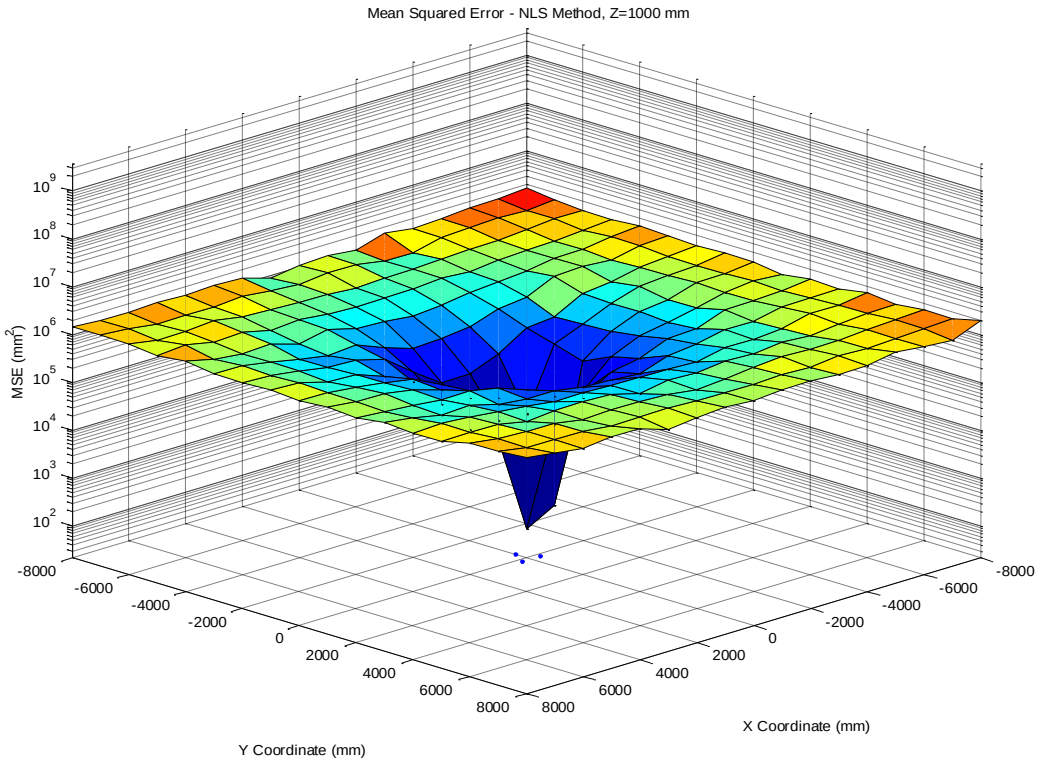
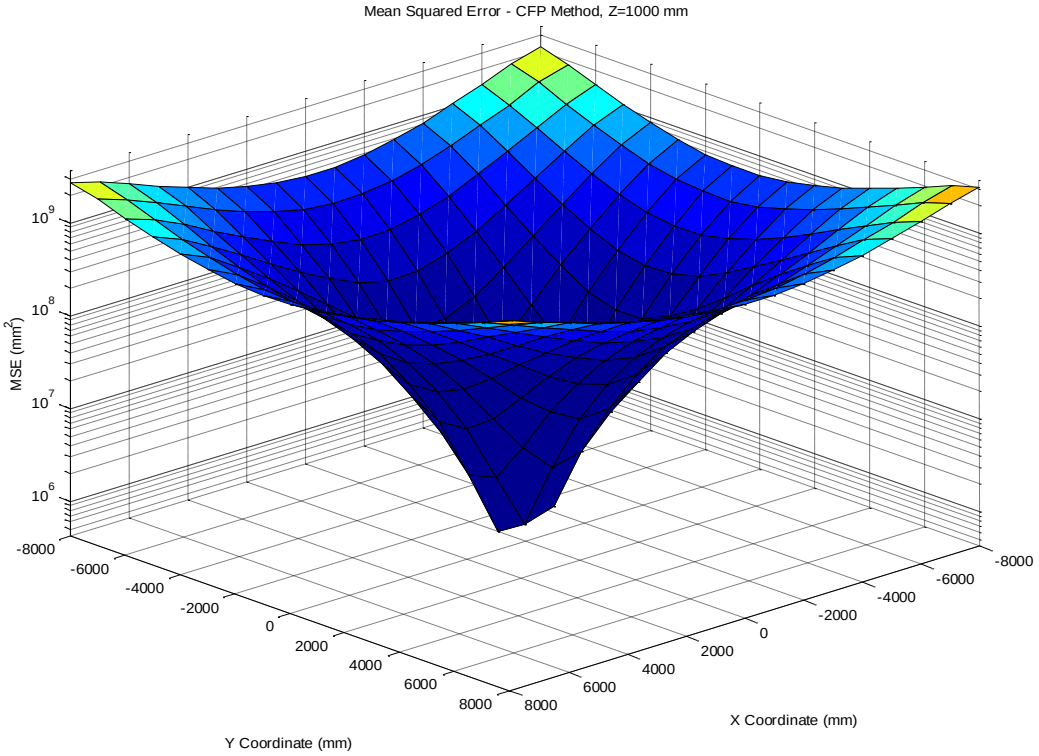




Figure 2.10: MSE, Z = 1000mm



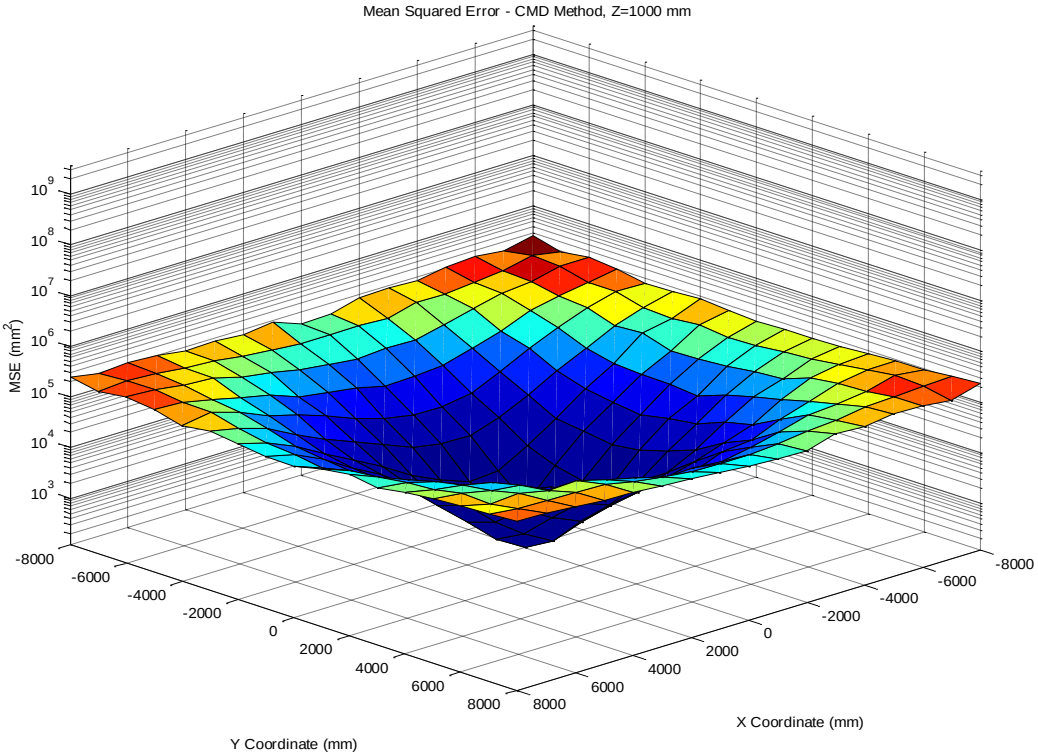




Figure 2.11: MSE, Z = 1000mm

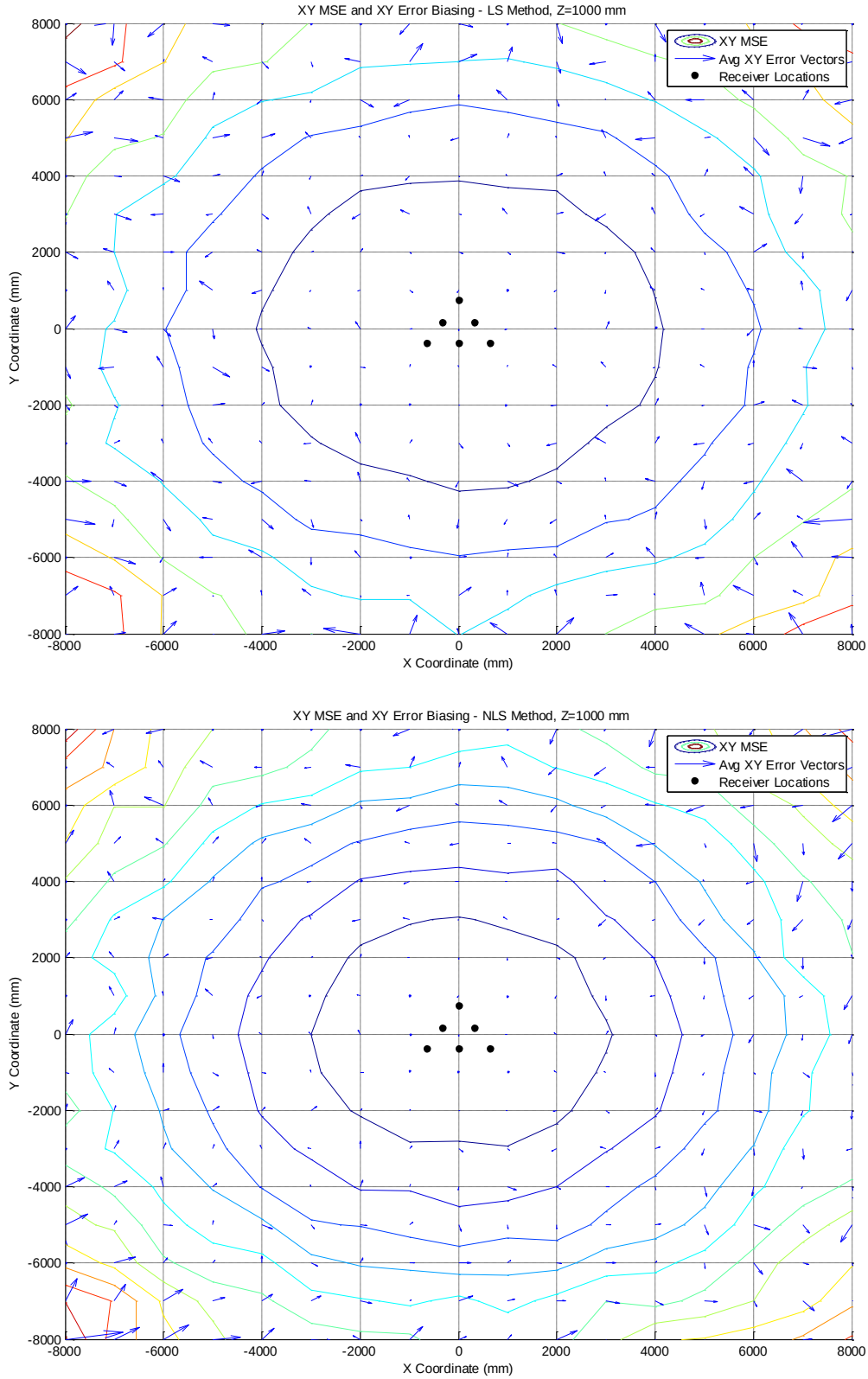



Figure 2.12: XY Error, Z = 1000mm

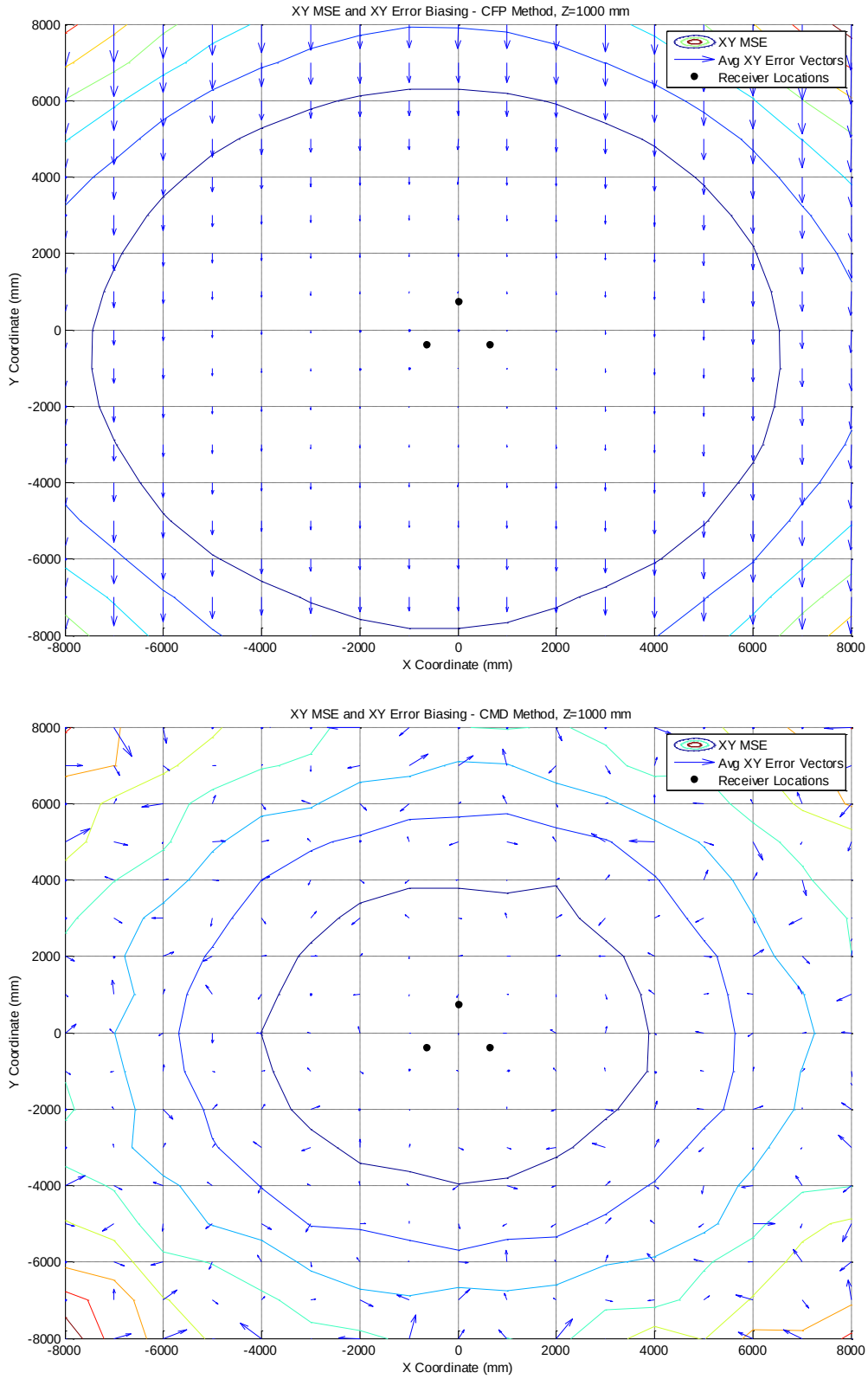



Figure 2.13: XY, Error, Z = 1000mm

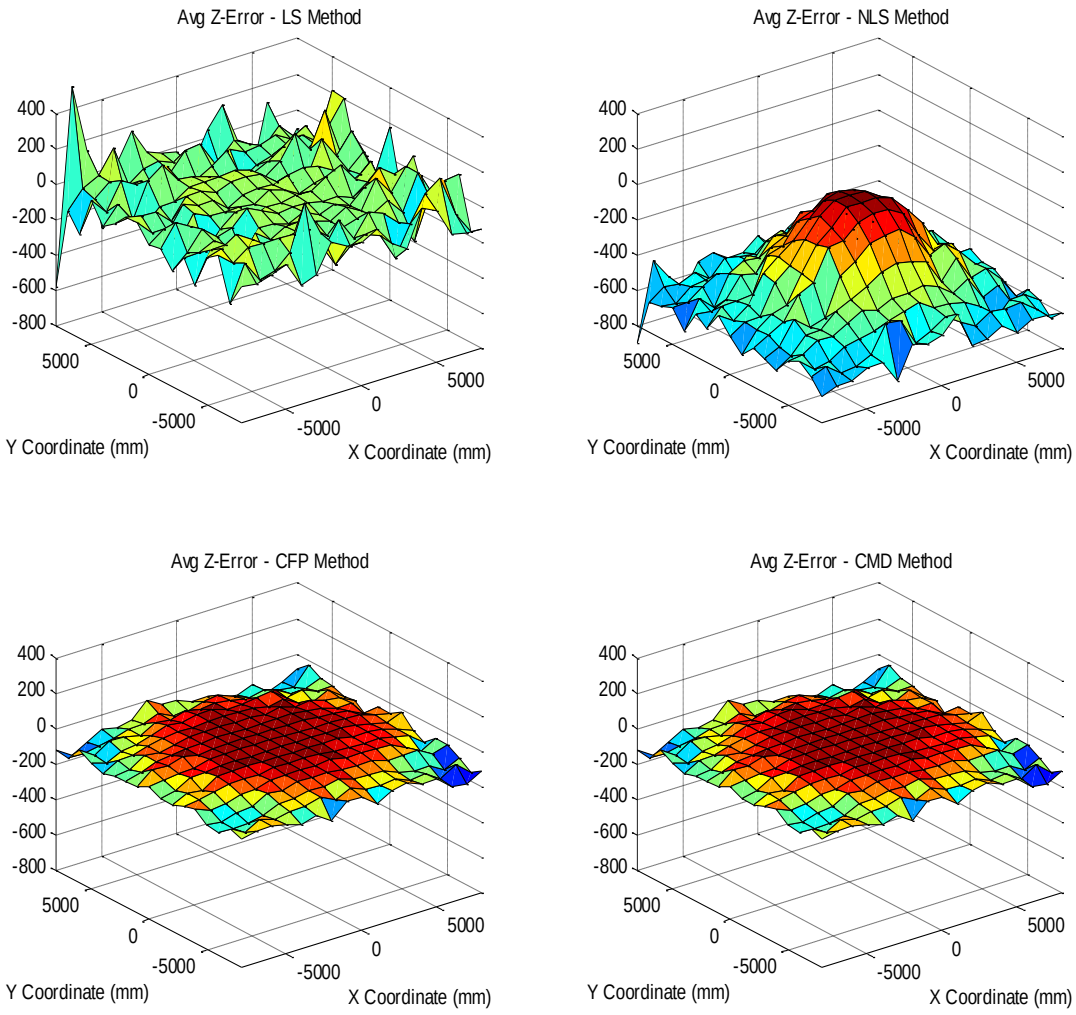
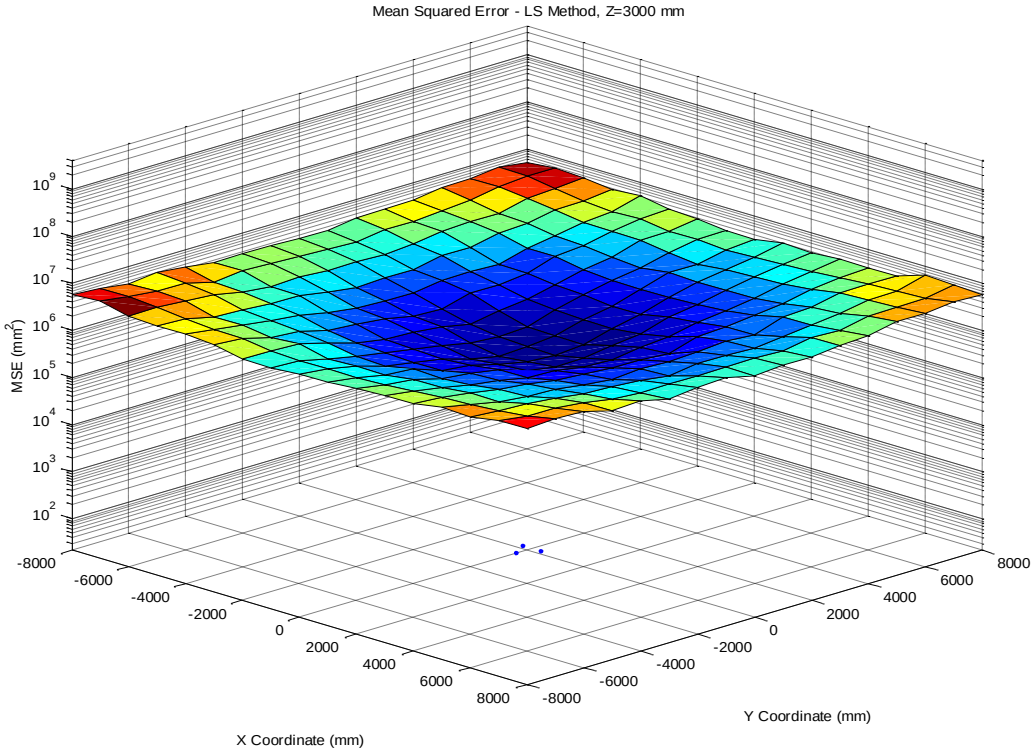



Figure 2.14: Z Avg Error, Z = 1000mm



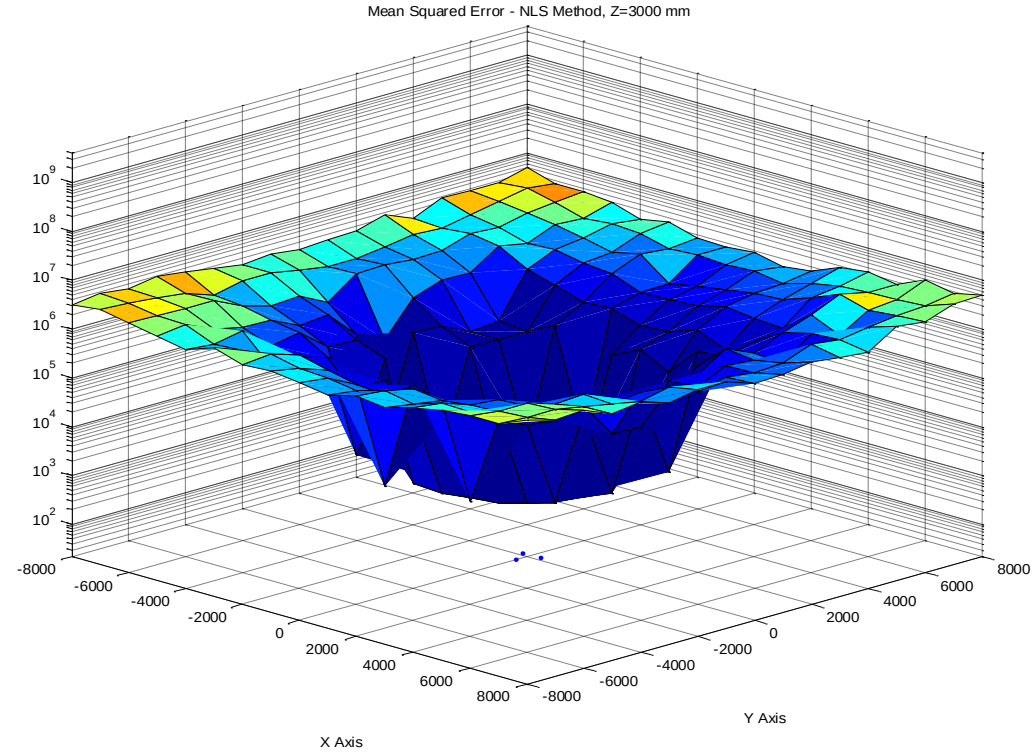




Figure 2.15: MSE, Z = 3000mm

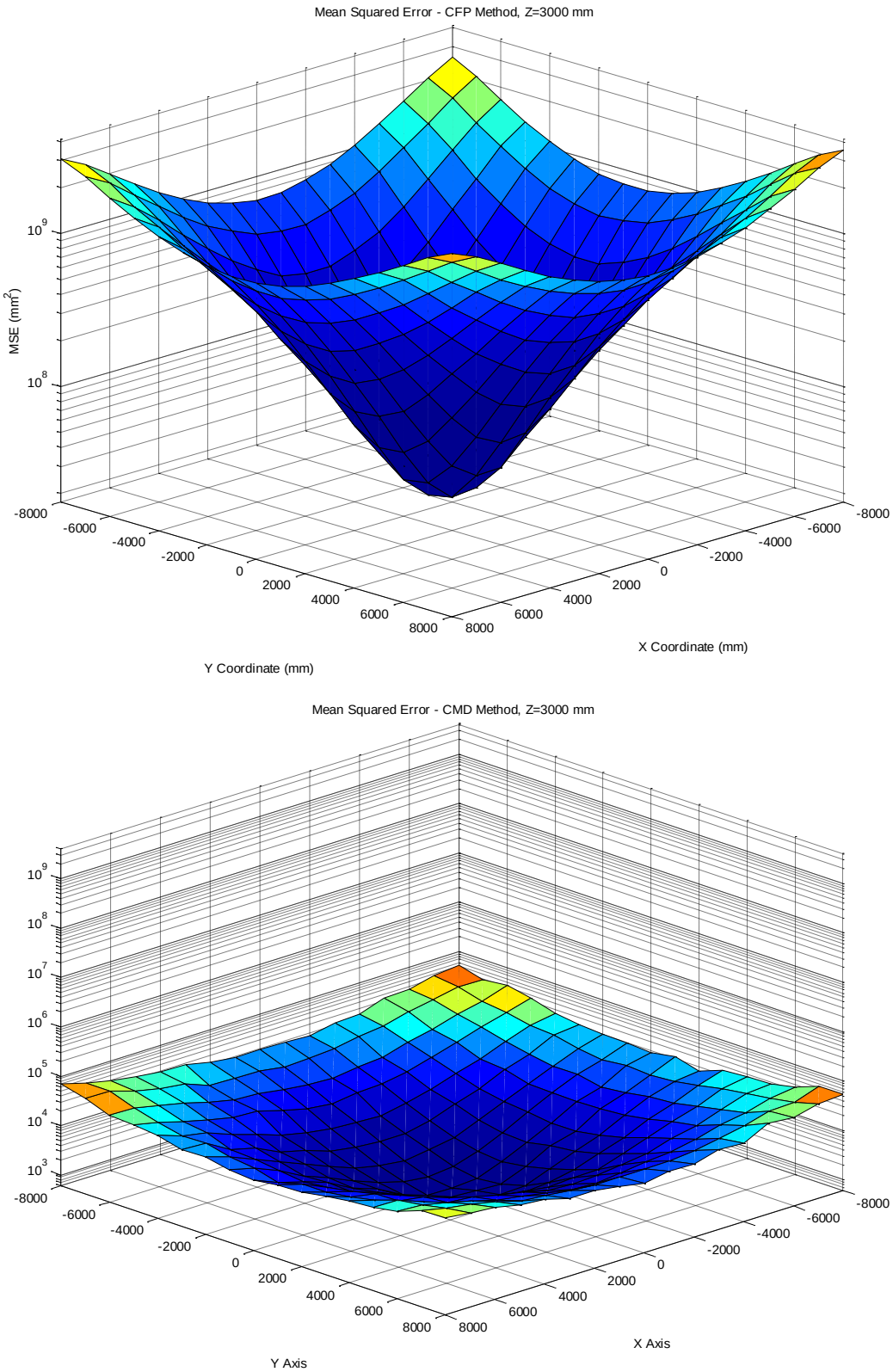



Figure 2.16: MSE, Z = 3000mm

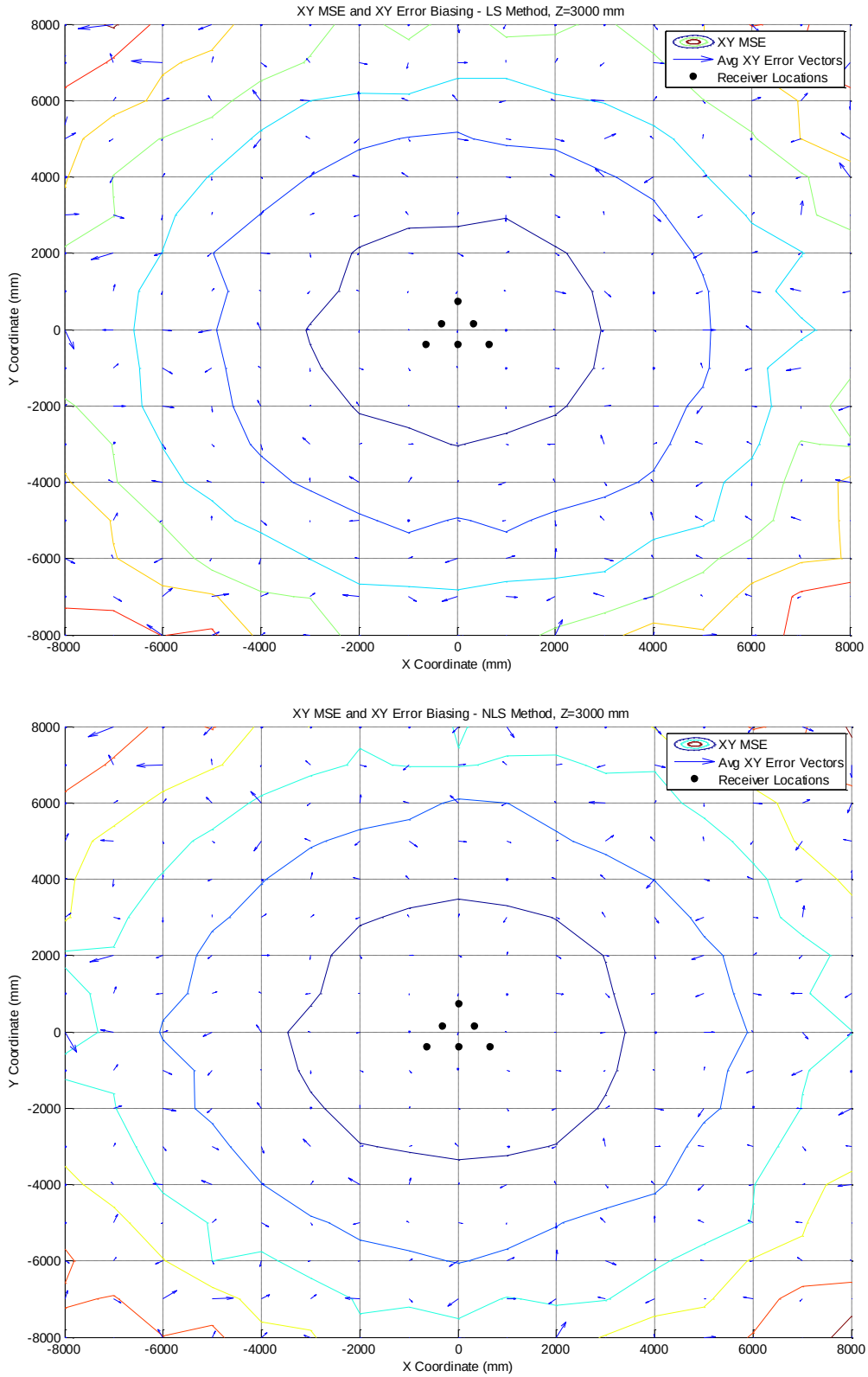



Figure 2.17: XY Error, Z = 3000mm

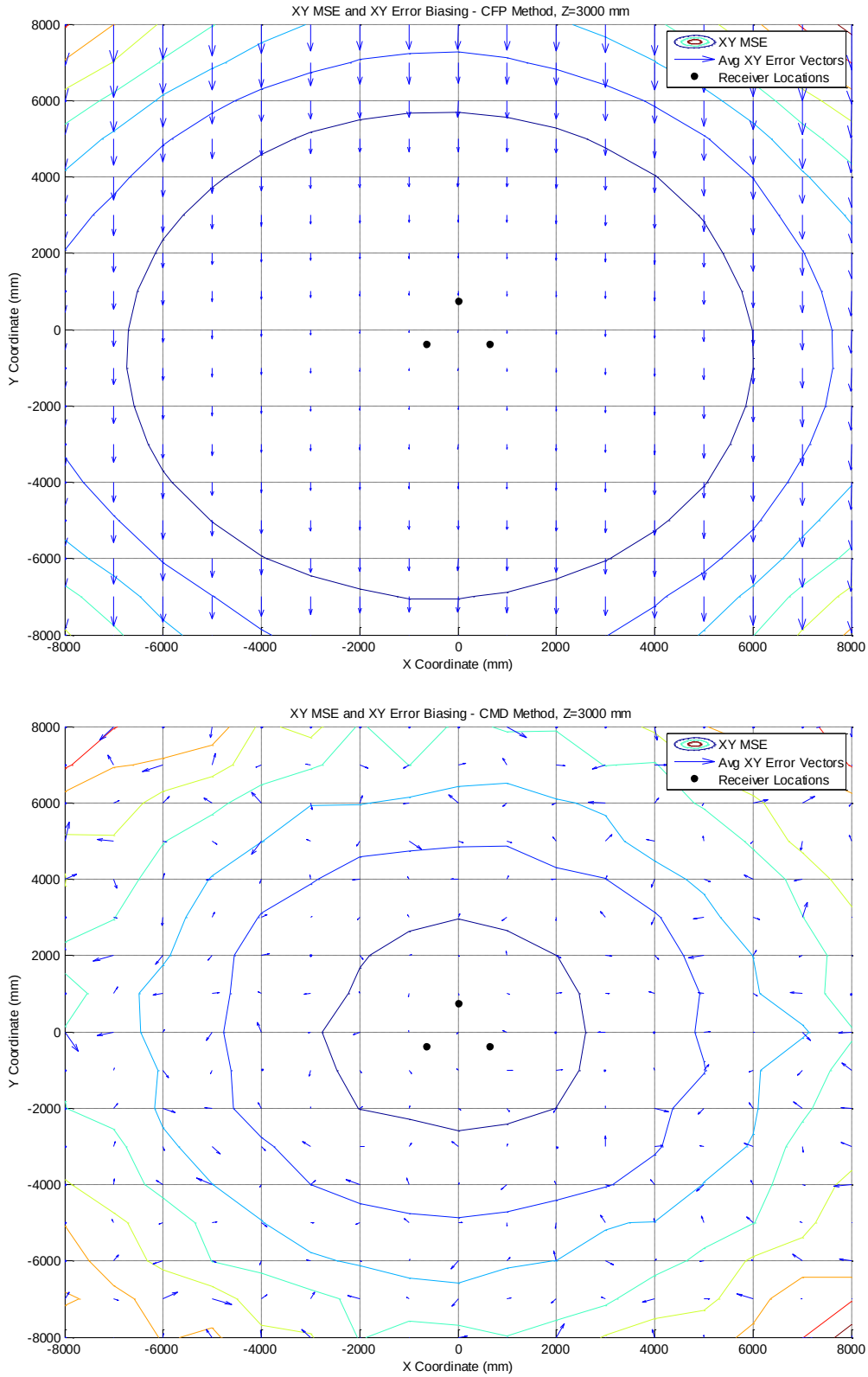



Figure 2.18: XY Error, Z = 3000mm

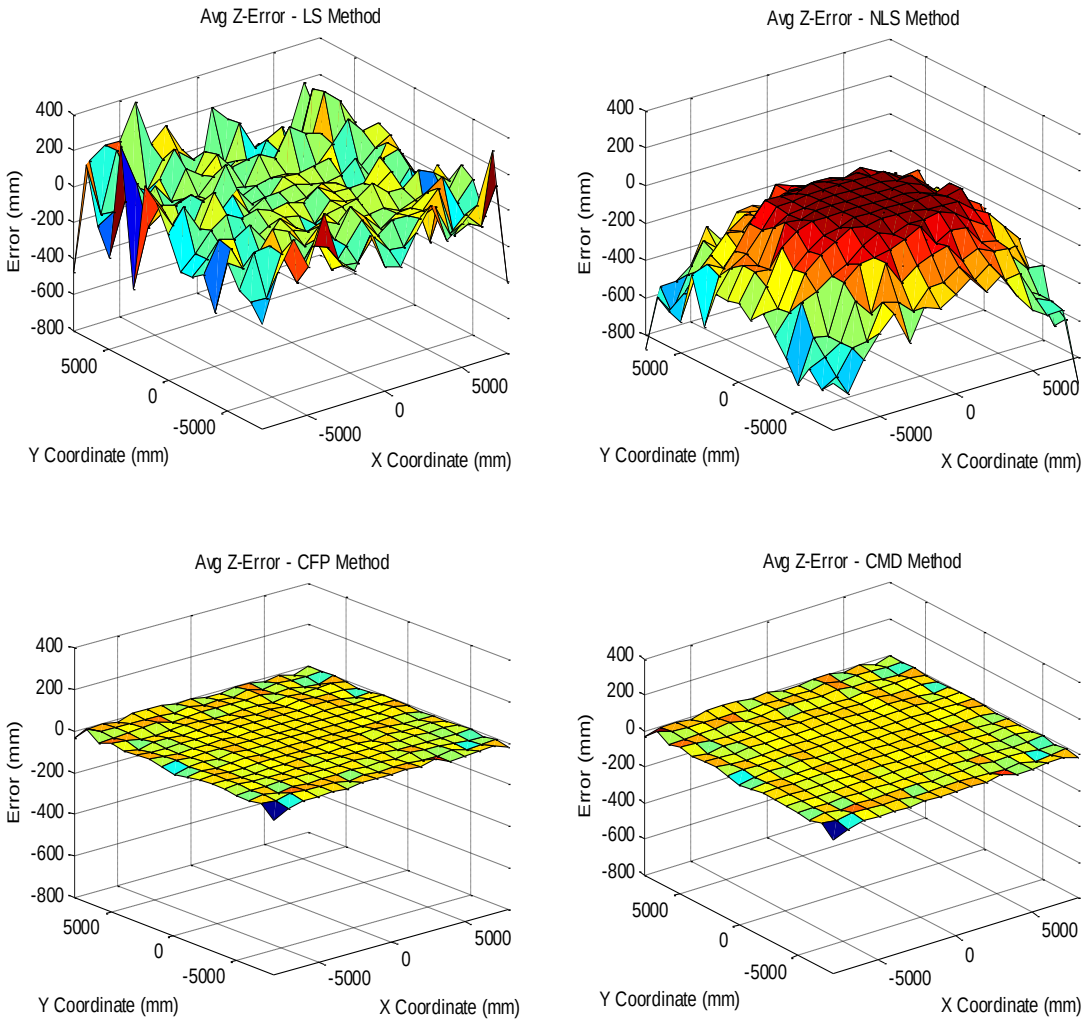



Figure 2.19: Z Avg Error, Z = 3000mm

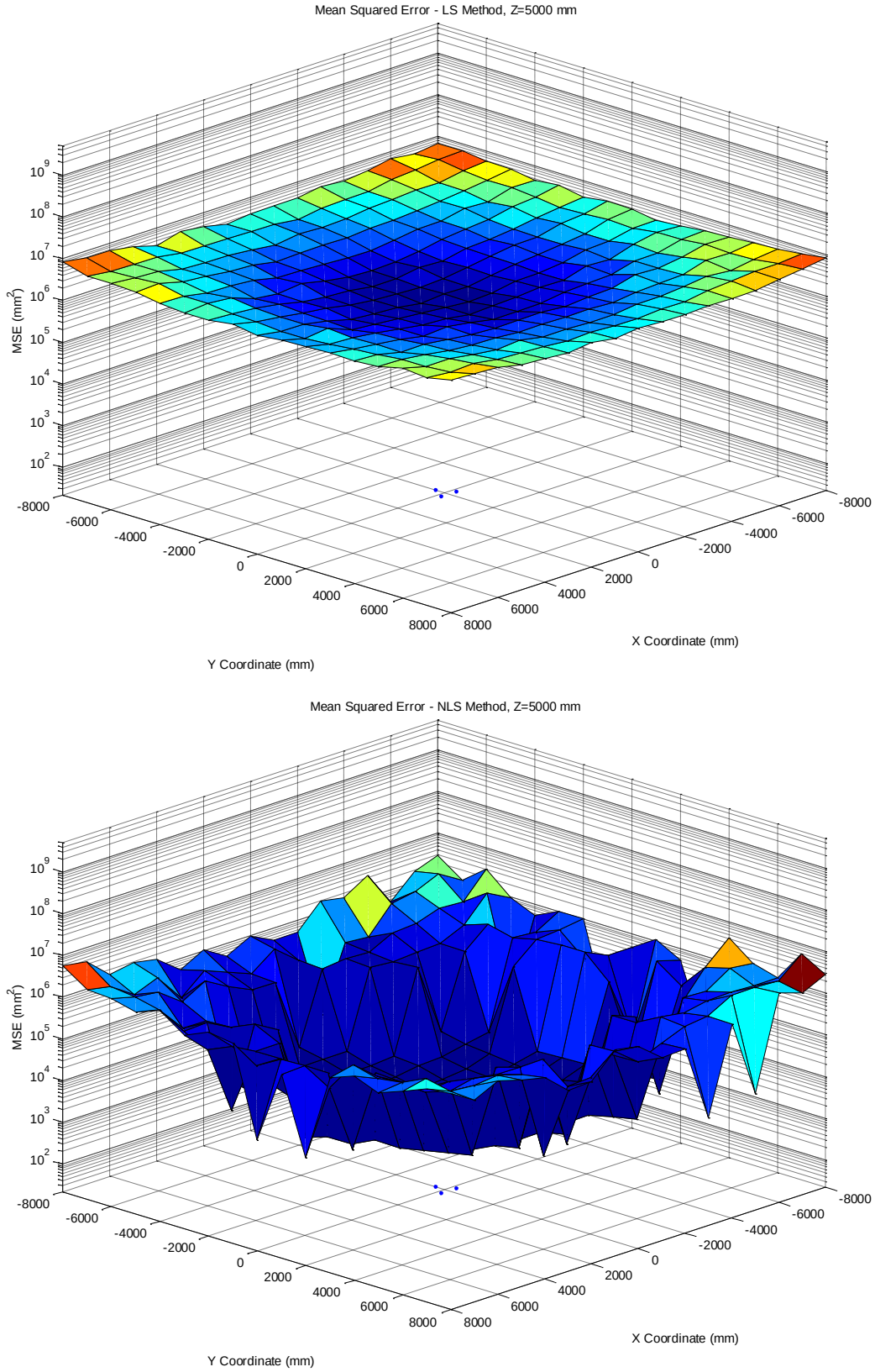
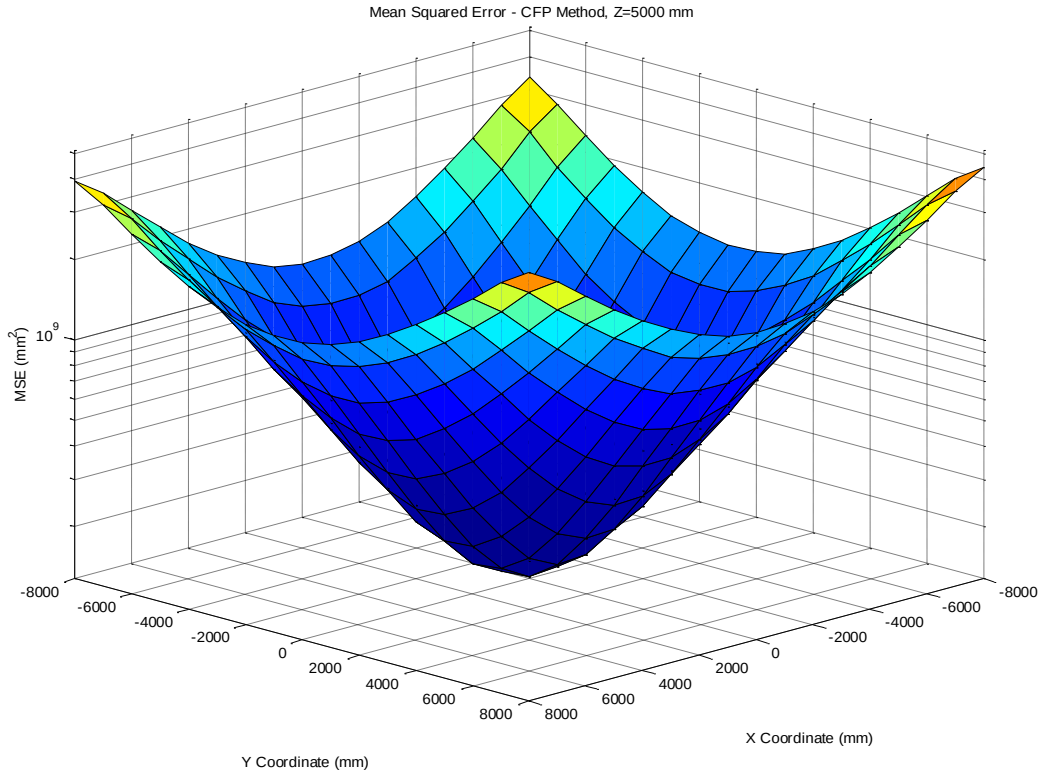



Figure 2.20: MSE, Z = 5000mm



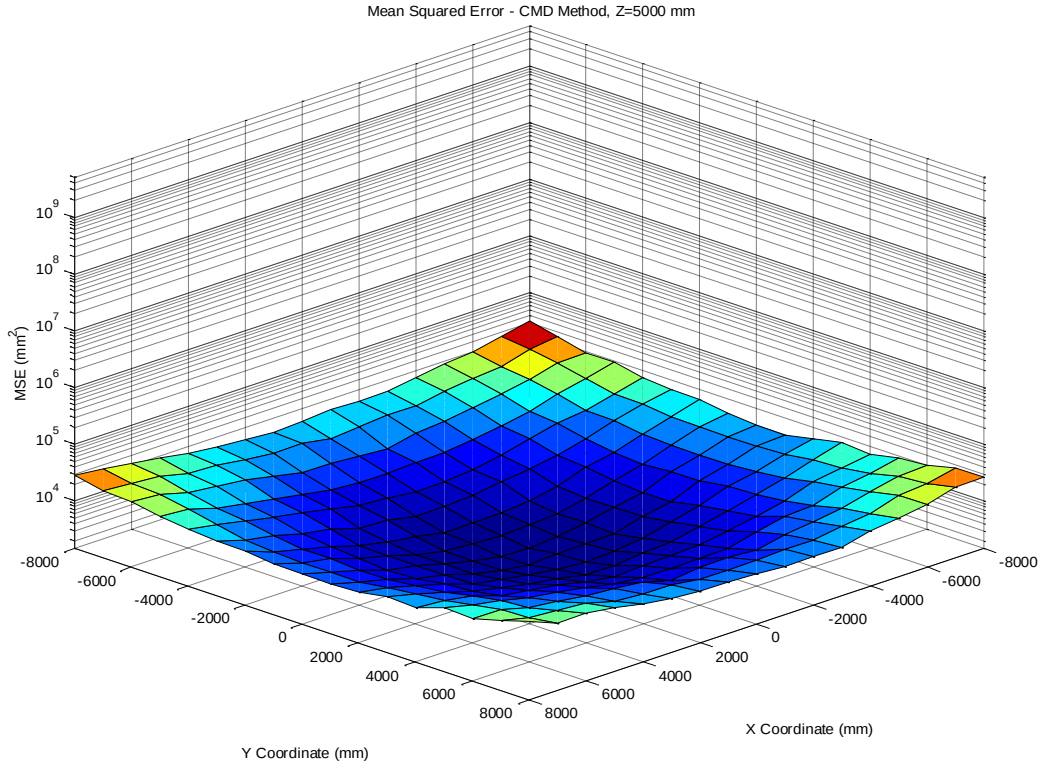




Figure 2.21: MSE, Z = 5000mm

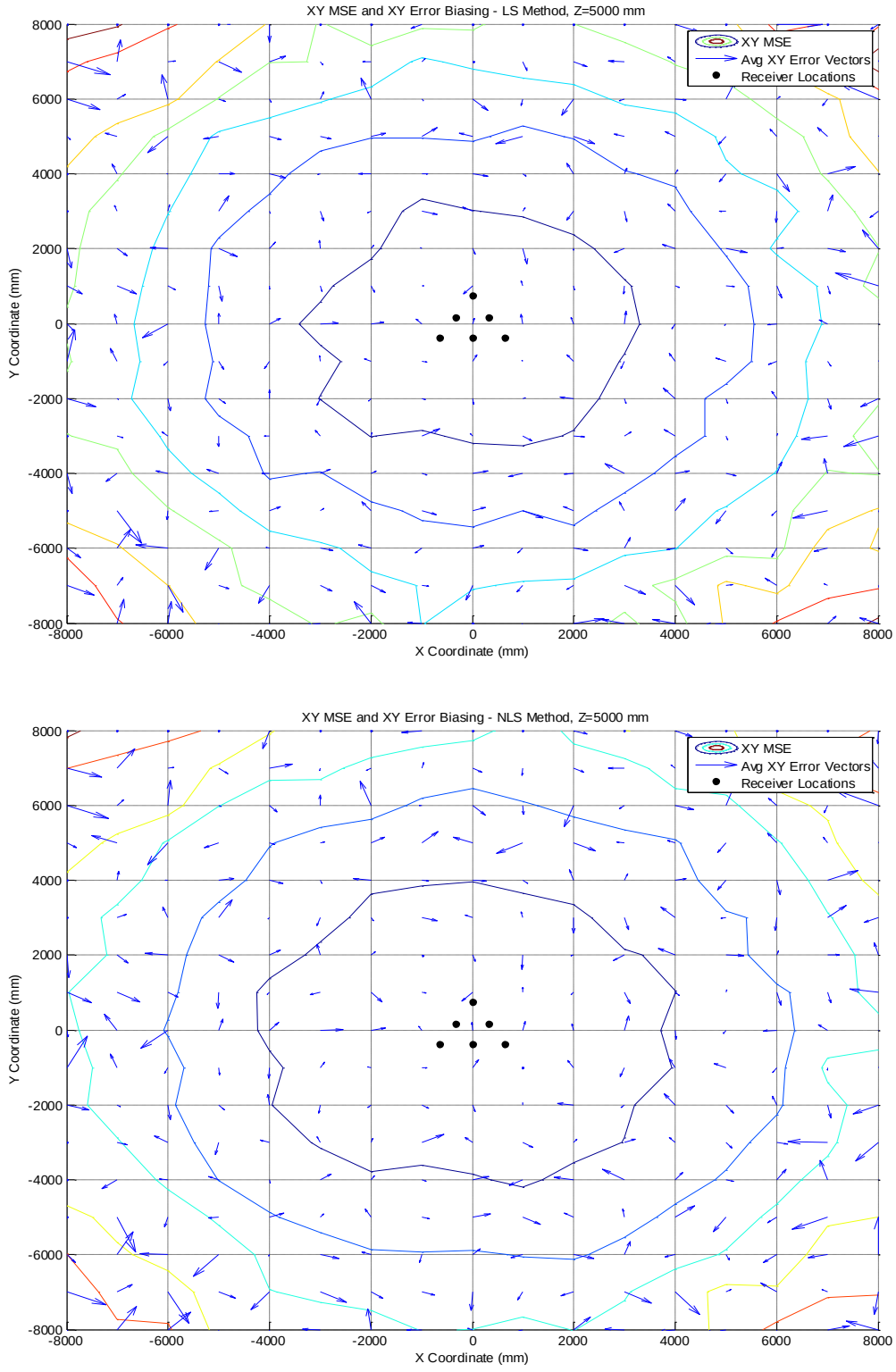



Figure 2.22: XY Error, Z = 5000mm

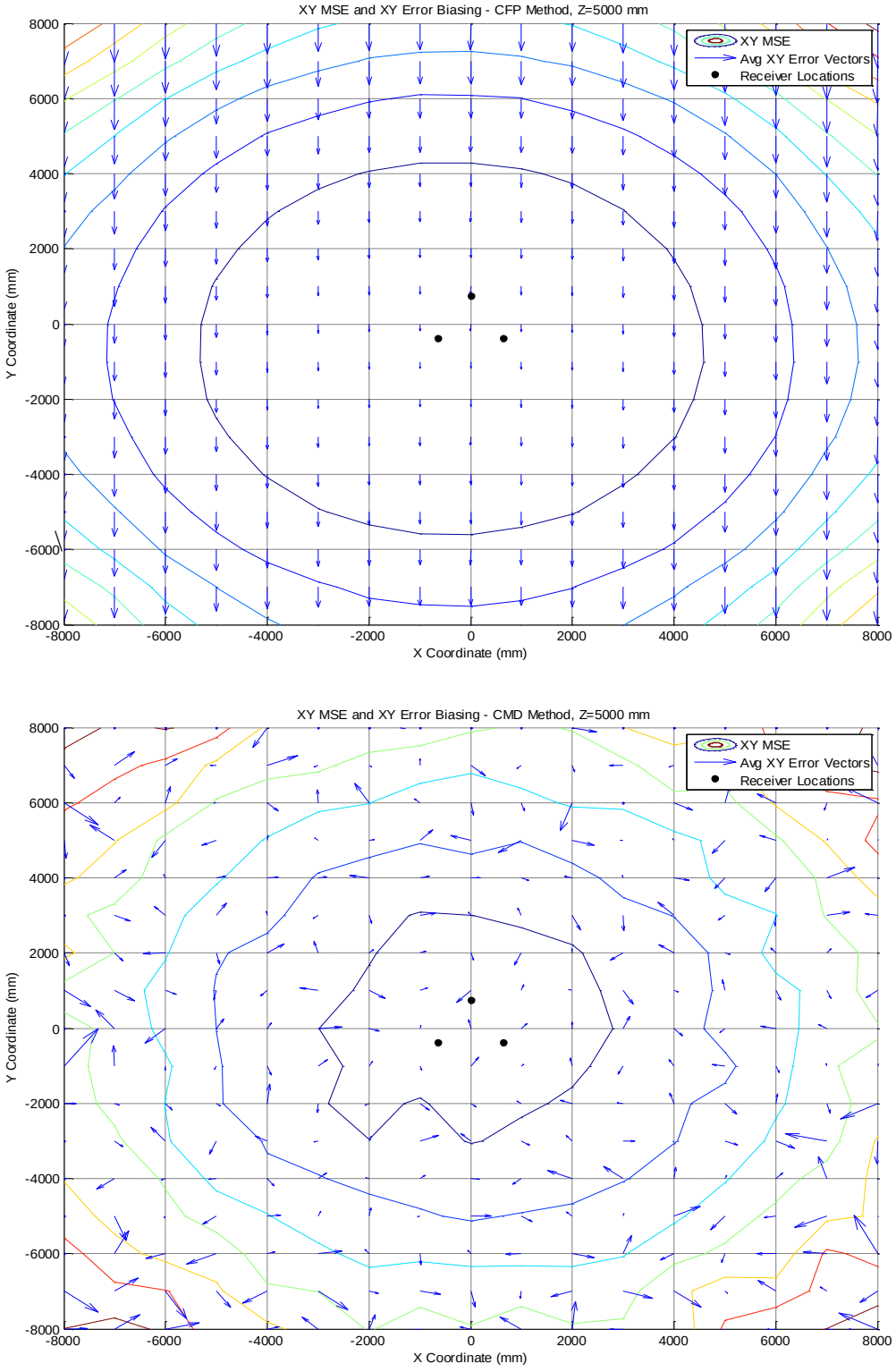



Figure 2.23: XY Error, Z = 5000mm

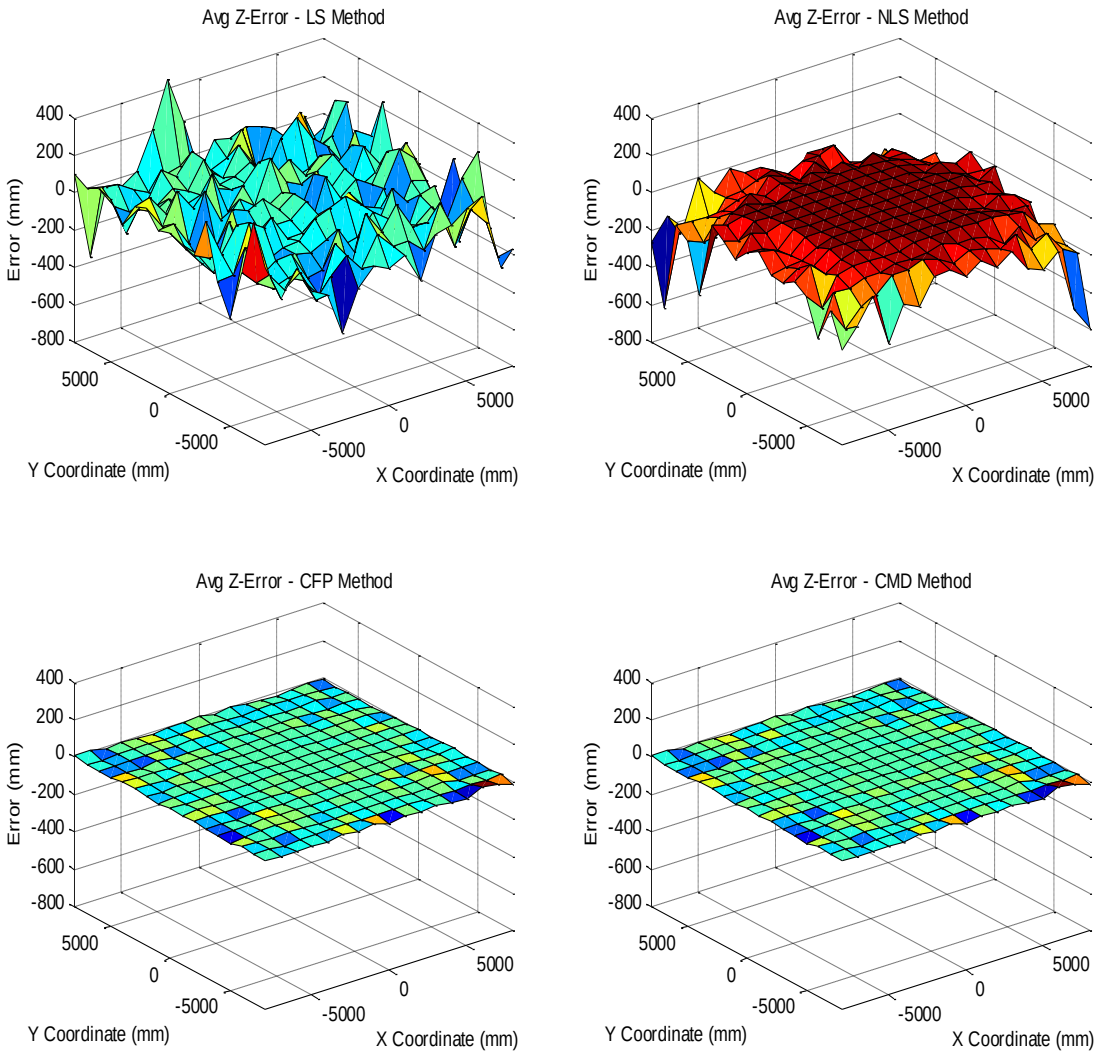



Figure 2.24: Z Avg Error, Z = 5000mm

While not providing an exhaustive visual comparison of the four trilateration methods, the plots shown at the three heights (Z = 1000, 3000, and 5000mm) allow for sufficient visualization of each method's performance within the 3D space simulated. At Z = 1000mm, it can be seen that for all methods, precision decreases dramatically as the quadcopter moves away from the origin. This can be explained as approaching a singularity, in which the quadcopter would be located on the Z = 0mm plane. When this happens, no meaningful position estimation can be made. Some methods exhibit asymptotic behavior at Z = 0mm Among all methods, the CMD method appears to be the most robust at Z = 1000mm.

*Average Position Error Results*

The average position error results are summarized in Figure 2.25 for all simulations. It should be noted that CMD and CFP methods are combined (only one plot for both) for the 'Average X Error' and 'Average Z Error' plots. The reason for this is that the results for these methods are identical in the X and Z directions. However, this is not the case for the Y direction. It can be seen that no error bias exists in the X direction for any of the trilateration methods. A mean of zero error is maintained for all simulations. In the Y direction only the CFP method (uses secondary vertical axis) exhibits non-zero error bias. Figures 2.13, 2.18, and 2.23 show this as the XY error vectors are consistently pointing in the negative Y direction. This indicates that some inherent bias exists in the Y direction for the CFP method. The 'Average Z Error' plot suggests that most all methods exhibit minimal

error bias in the Z direction except the NLS method. For this method, The Z error tends to increase as the height increases. At Z = 7000mm, the error has been reduced to nearly 0.

*Mean Squared Error Results*

The mean squared error (MSE) results are summarized in Figure 2.26 for all simulations. Once again, results for CFP and CMD methods are represented as one line for X MSE and Y MSE as the results are identical for both methods. Also, the Y MSE results for the CFP method correspond to the secondary vertical axis (similar to that in Figure 2.25). MSE is an appropriate representation of total error as it measures both accuracy and precision. Here we find that X MSE and Y MSE generally increase as Z increases for all methods. This is not surprising as the system becomes increasingly sensitive to error as the quadcopter moves farther away from the origin. Mixed results are observed among the the four trilaterations for Z MSE. While the error becomes increasingly large with increasing Z when using the LS method, the NLS method shows a maximum error near Z = 2000mm then tends towards 0 as Z continues to increase. Both the CFP and CMD methods show identical results for Z MSE and are near zero for all simulated Z values.

The total MSE is shown for each method in Figure 2.27. This is simply a summation of X, Y, and Z MSE and serve as an overall representation of error at each height, Z. Again, the CFP plot corresponds with the scale shown on the secondary vertical axis. This comparison very clearly shows that the CMD method exhibits the

least overall error and is the least sensitive to error based on the postion of the helicopter in in the space simulated.

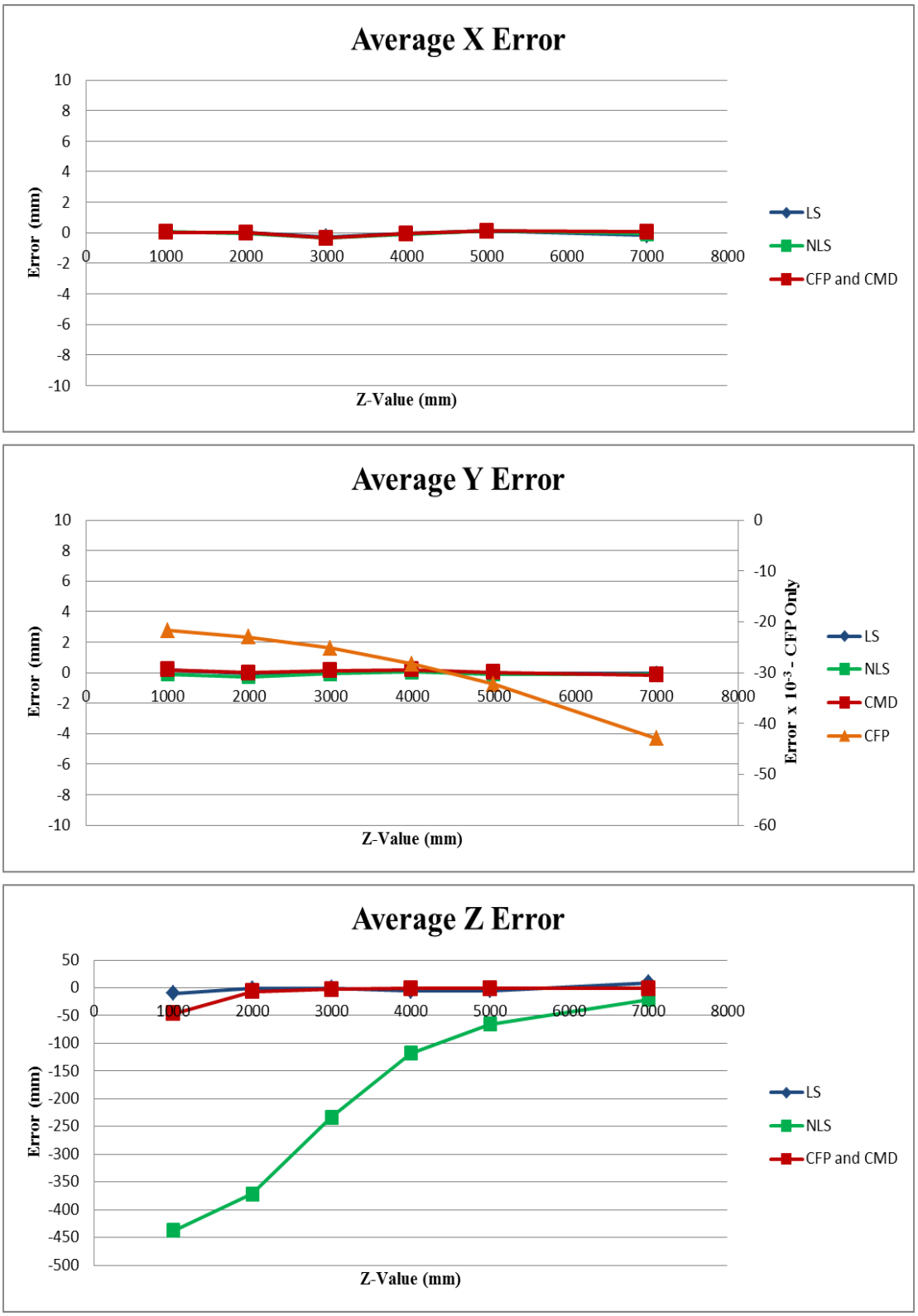



Figure 2.25: Average error in X, Y, and Z directions


Figure 2.26: Mean squared error in X, Y, and Z directions

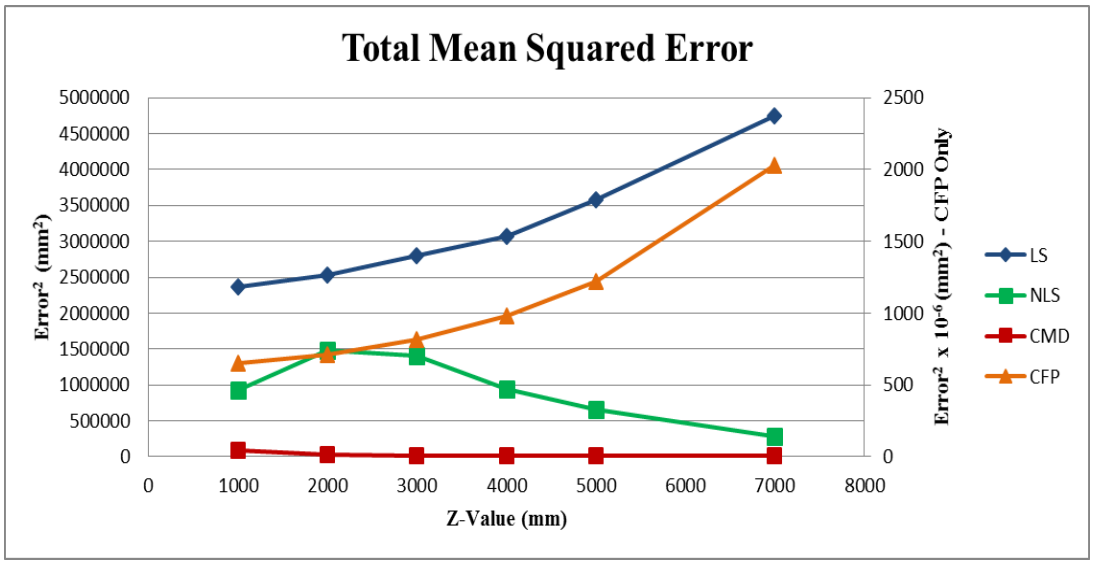


Figure 2.27: Total mean squared error

# CHAPTER III

# DATA COLLECTION, FLIGHT TESTING, AND SYSTEM IDENTIFICATION

## 3.1 System Hardware and Software

A prototype test setup has been developed using the Hexamite RFID sensor network and the DJI F450 radio controlled quadcopter platform. Additionally, a data acquisition system was developed to allow for data collection that was later used to generate transfer functions describing the quadcopter's response to several basic inputs.

### *3.1.1 Ultrasonic RFID Range Sensor Network*

Six ultrasonic RFID receiver sensors (Rx) and two ultrasonic RFID transmitters (Tx) are used to provide range measurements (up to 14 meters) from the quadcopter to the stationary receivers. The Rx/Tx sensor combination used is the Hexamite HX19 system. Each of these is shown in Figure 3.1. A mobile wooden platform was built to mount each of the six receivers. The location of each receiver is known and not subject to variation in position as a result. The configuration of the six receivers on this platform is shown in Figure 3.2. The two transmitters are secured to the front and rear of the F450 quadcopter as shown in Figure 3.3. Using two transmitters (as opposed to only one) allows for the heading angle to be calculated and also yields a more reliable estimation of the quadcopter's position.

The Hexamite HX19 system was chosen for this study for its high advertised range measurement accuracy (several millimeters), rather light weight components (most critical for the transmitters), low cost, and ease of integration with a data acquisition system.

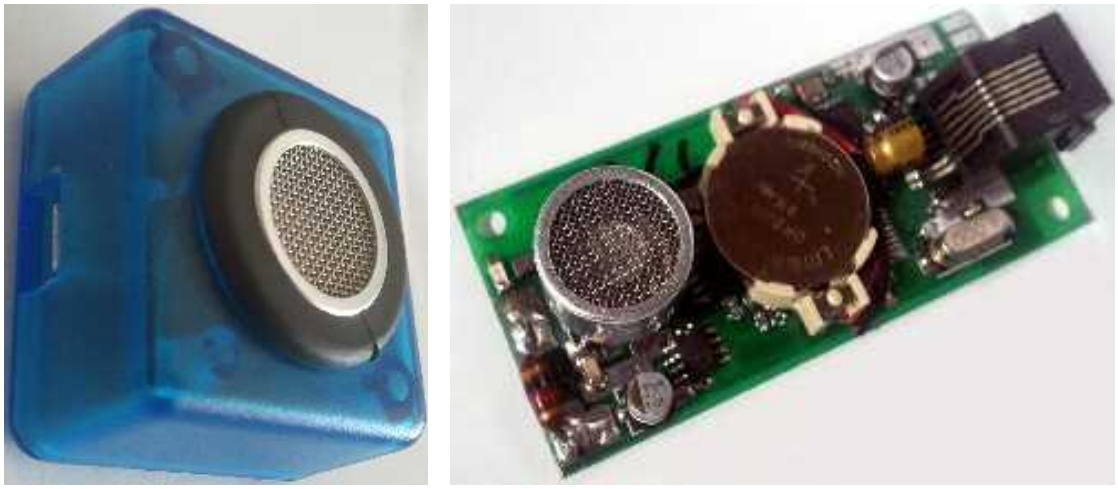


Figure 3.1: Hexamite ultrasonic RFID transmitter (left) and receiver (right)

Figure 3.2: Six sensor mobile platform

Figure 3.3: DJI F450 quadcopter platform with front/rear transmitters

### *3.1.2 Data Acquisition Program*

A customized program was developed for the purpose of: 1) integrating with the HX19 sensor system to facilitate the acquisition and logging of range measurement data and 2) integrating with the the Futaba 2.4GHz radio system to send controlled, automated, and isolated commands to the quadcopter using the 'buddy box' capability included on most modern hobby radio systems. Some of the basic functions of the program developed include the following:

1) Data acquisition, logging, and report generation

2) Trim adjustment (all channels)

3) Instantaneous position calculation and visualization (X, Y, Z, & heading)

4) Control command impulse delivery for flight testing and system identification

A screenshot of the program's user interface is shown in Figure 3.4.

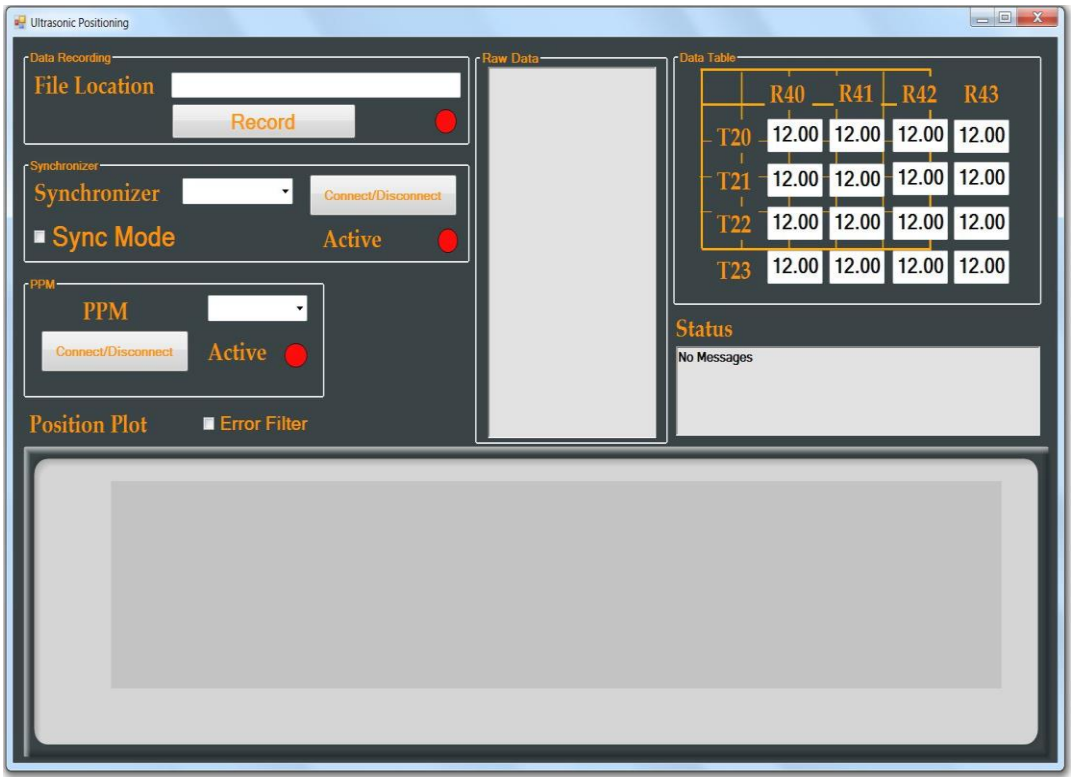



Figure 3.4: Data acquisition program GUI

## 3.2 Flight Testing and Data Post-Processing

A series of flight tests were conducted to verify that the data acquisition program was functioning properly and collecting data there were accurate,

meaningful, and useful. Ultimately, the goal is to have a system that provides real-time position feedback for the quadcopter. However, for the purpose of this study, data were recorded and post-processed in order to learn more about the system and characterize the high-level operation of the quadcopter. An overview of the data collection system and post-processing operations is shown in Figure 3.5.

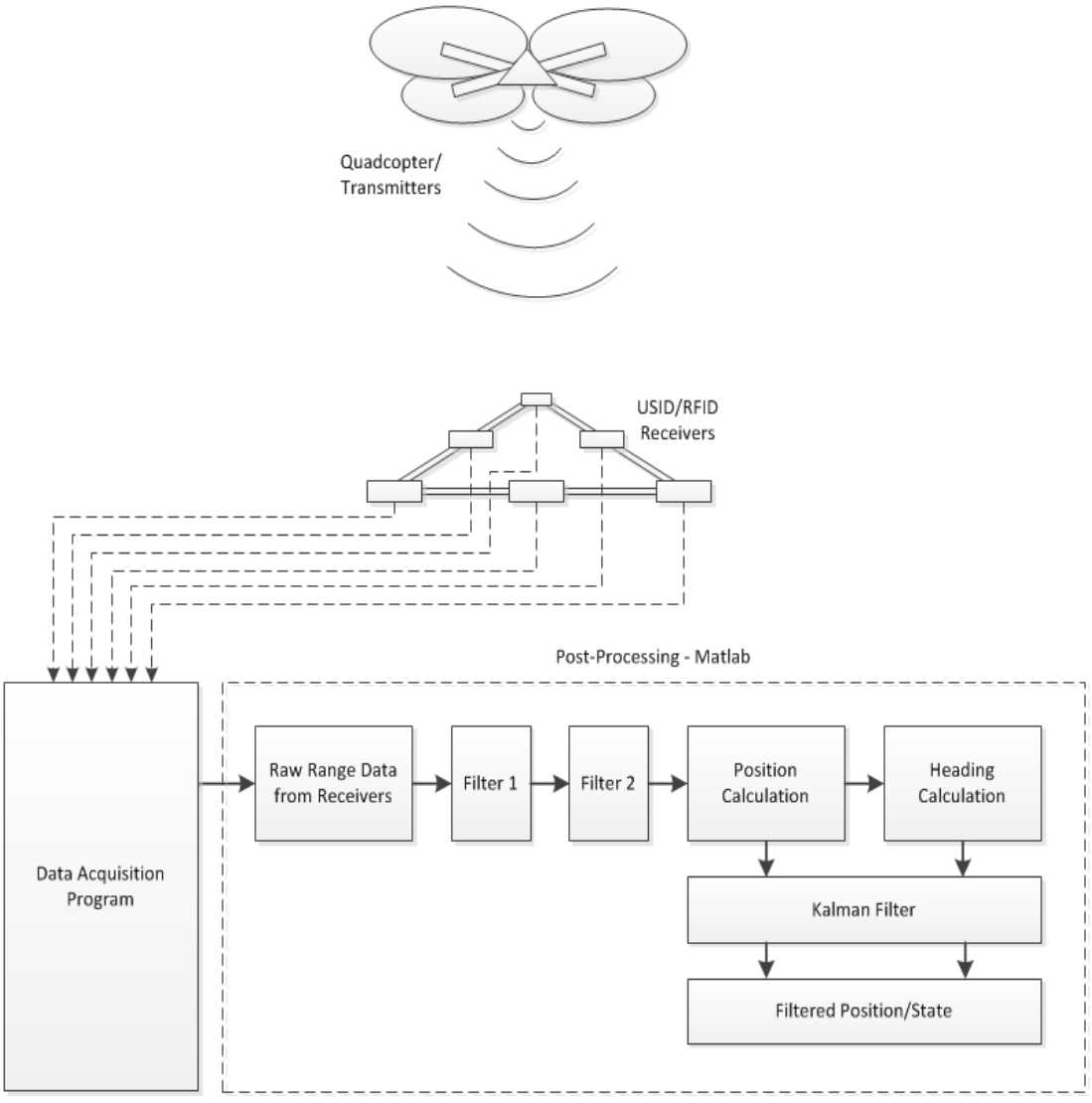



Figure 3.5: Data flow for quadcopter localization

### *3.2.1 Range Sensor Data*

The data collected using the HX19 sensor system is compiled using the aforementioned data acquisition (DAQ) program where timestamps are assigned to range measurements received from each receiver. The system consists of six receivers and two transmitters (on quadcopter). This allows for a total of 12 range measurements to be available at any given time. The DAQ program updates the range measurement for each Tx/Rx pair as often as possible, and an example of the resulting time-series data is shown in Figure 3.6.

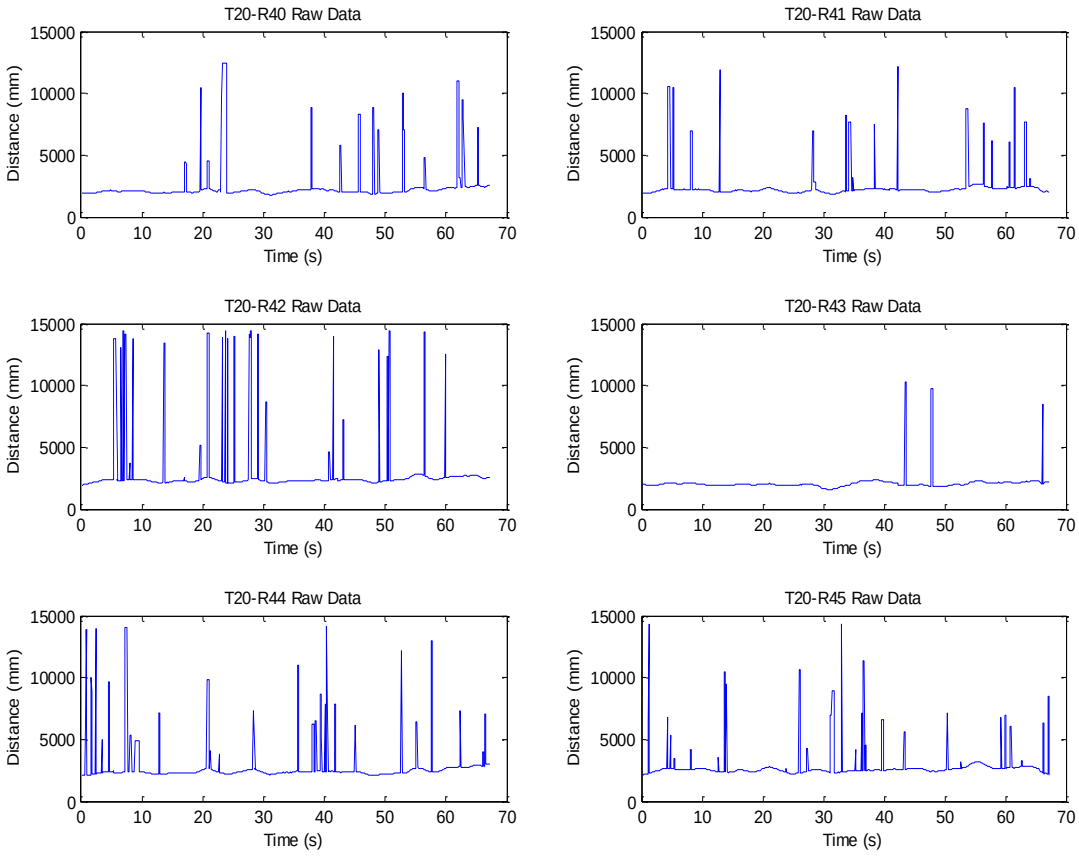



Figure 3.6: Raw data logged using DAQ system

The data set shown represents the measured distance between transmitter 'T20' and each of the six receivers, 'R40,' 'R41,' 'R42,' 'R43,' 'R44,' and 'R45.' A similar set of data is also available for the second of two transmitters, 'T21.' The receivers are found to produce occasionally error-ridden measurements. This is indicated in Figure 3.6 by the large spikes in the data stream. However, due to the nature of these errors some simple filtering can be applied in order to minimize their impact on the position calculation.

### *3.2.2 Filtering of Raw Range Data*

The errors that are observed in the raw data streams can be filtered out by way of ensuring that a few known physical constraints of the system are adhered to. These will serve as a 'check' to evaluate whether a particular set of range measurements make physical sense in the context of our system. The two simple constraints used for this filtering pertain to 1) the maximum possible speed of the quadcopter and 2) the fixed separation of the two transmitter mounted on the quadcopter.

#### *Filter 1: Velocity Constraint*

The quadcopter is simply not capable of moving at high speeds. And although the exact speed of the quadcopter is not known prior to calculating and comparing positions from consecutive sets of range measurements, it is known that the range measurement provided by a particular Tx/Rx pair can not change at a rate that is greater than the maximum speed of the quadcopter. Even this assumption is

conservative as the quadcopter would have to be moving along a path which is collinear with line defined by the initial coordinates of the receiver and that of the transmitter. Figure 3.7 shows six histograms for the difference in consecutive range measurement values for the same data sets shown in Figure 3.6. As expected, the majority of the values are near zero. The values that fall to the left and right of the center column on each plot are indicative range measurement errors. The value of 1000mm is used as the cutoff for maximum quadcopter speed. This translates as roughly 10 m/s (or 22.3 mph) given that the maximum sampling interval is roughly 0.1 seconds (data is not an evenly spaced time-series). The filter implemented, 'Filter 1,' removes all range measurement values indicating that the quadcopter is moving faster than 10 m/s. For these errors, the previous range measurement is simply adopted.

*Filter 2: Fixed Transmitter Separation*

The second physical constraint is that the physical distance between the two transmitters mounted on the quadcopter is fixed. This means the difference between range measurements from any one receiver to each of the transmitters should not exceed a certain value. The distance used here is 0.5 m. The physical distance between the two transmitters (see Figure 3.3) when measured with a tape measure. The filter implemented evaluates the data such that if the difference between the two measurements for one receiver to each of the transmitters is larger than 500 mm then

the previous measurement values are used. Figure 3.8 shows the plotted time-series data before and after filtering.

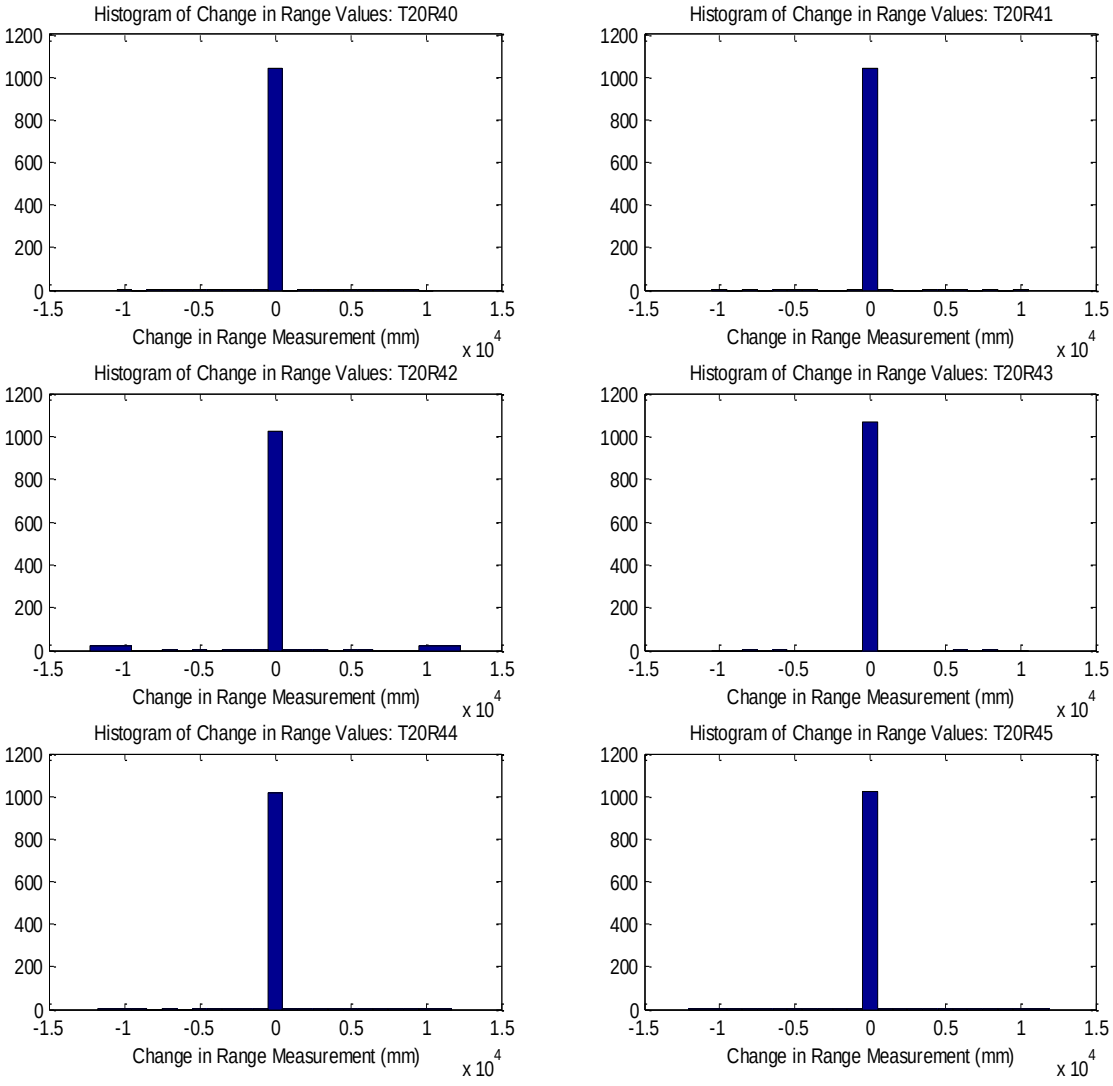



Figure 3.7: Histograms for change in consecutive range measurements

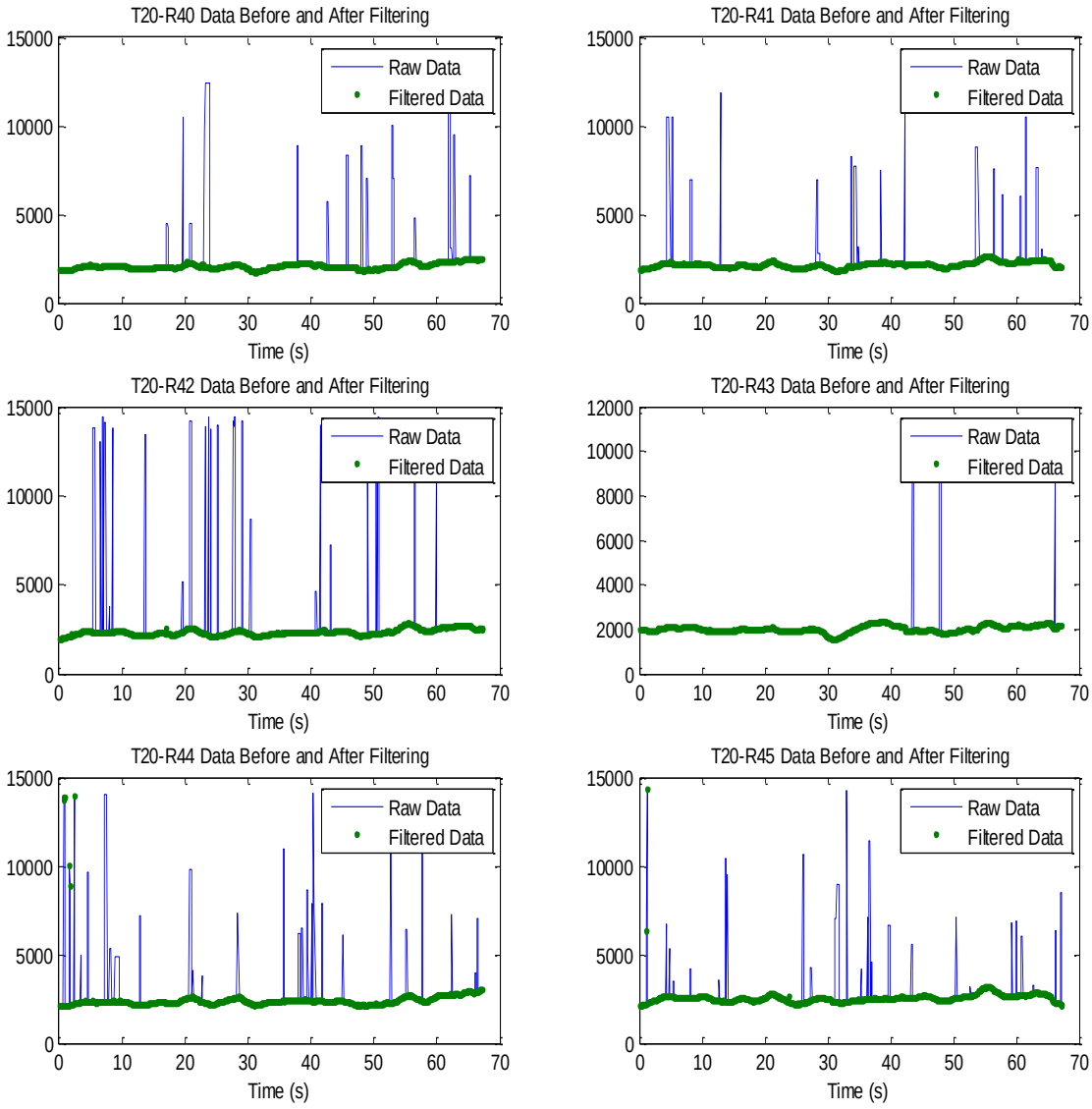
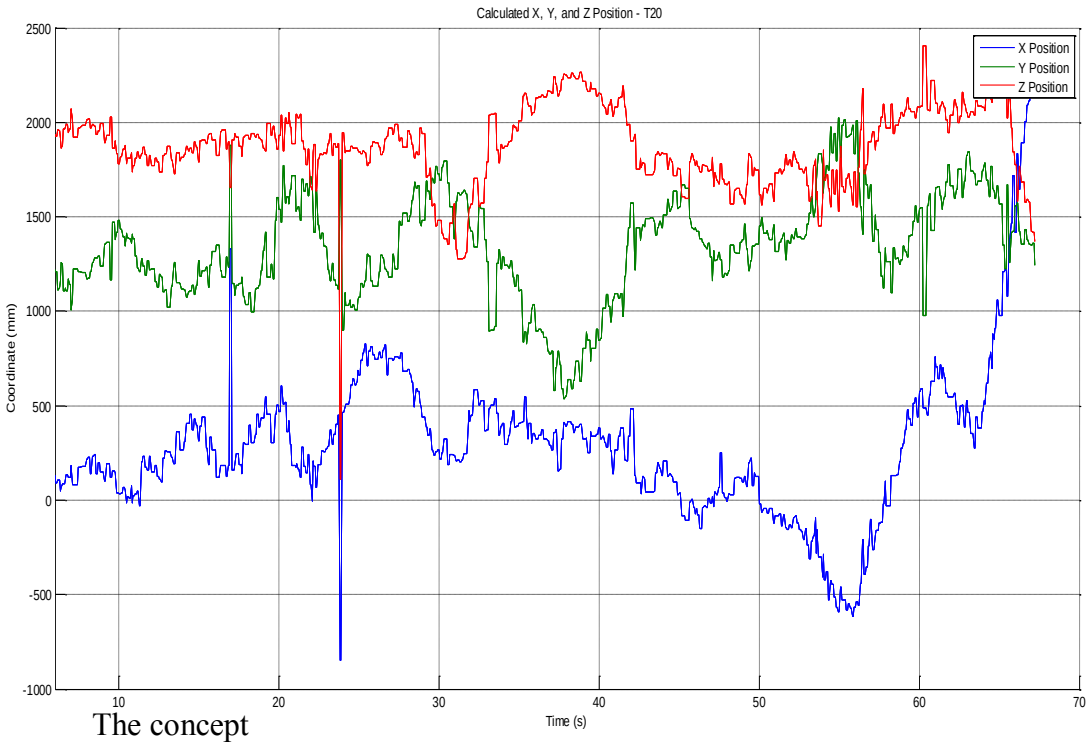



Figure 3.8: Range measurement data before and after filtering

### *3.2.3 Filtering of Position Data Using Kalman Filter*

Following the filtering of the raw data, a position calculation is made for each set of set of range measurements. The CMD trilateration method is used to calculate position as it was previously evaluated using a Monte Carlo Simulation error analysis and determined to be the most accurate and robust method among the several evaluated. An exemplary plot of calculated X, Y, and Z position is shown in Figure 3.9. Evidently, the calculation yields a noise signal that requires filtering in order to be used. At this point, a Kalman filter is used similar to that described in [7]. The data is filtered in each direction (X, Y, and Z) independently as summarized below.




The concept

Figure 3.9: X, Y, and Z position calculation for T20

employed by the Kalman filter is estimating the future state of a system based upon the current state of the system and some input to the system given the physics and inherent error of both the system itself and the measurement of the system's state. As applied to the quadcopter UAV, an estimation of position is needed given some error-ridden measured position (calculated from range measurements) and the physical state of the system. Three general models can be used for the state of the system: position (P), position and velocity (PV), and position, velocity, and acceleration (PVA). The PV model is used for the application of the quadcopter position estimation. It is also important to note that it is assumed that both the system (process) noise and measurement noise are independent from each other and are normally distributed.

To begin, the state estimation is defined as

$$X_{est} = \bar{x}_t + K(d_t - \bar{d}_t) \tag{3.1}$$

where $\bar{x}_t$ is the predicted state based on the previous state, $d_t$ is the actual measurement made (calculated position), $\bar{d}_t$ is the predicted measurement, and $K$ is the Kalman Gain. Initially, the state prediction is made based on the previous state:

$$\bar{x}_t = A * \ _{t-1} + B * U_t + \varepsilon_x \tag{3.2}$$

where $A$ is the system of physics equations defining the dynamics of the quadcopter, $_{t-1}$ is the previous state of the quadcopter, $B$ is a constant associated with the control input, $U_t$, and $\varepsilon_x$ is a state error term. The control term is unknown, however, and can be dropped from the equation. For a PV model, $\bar{x}_t$ takes the form of

$$\bar{x}_t = \begin{bmatrix} p_t \\ v_t \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{t-1} \\ v_{t-1} \end{bmatrix} + \varepsilon_x \tag{3.3}$$

where $p_t$ and $v_t$ represent the position and velocity at time *t*, respectively, $p_{t-1}$ and $v_{t-1}$ respresent the position and velocity at the time of the previous estimation, and $T$ is the time elapsed since the previous estimation was made. The measurement prediction is defined as

$$\bar{d}_t = C * \bar{x}_t + \varepsilon_d \tag{3.4}$$

where $C = [1 \quad 0]$ and $\varepsilon_d$ is the error associated with the measurement prediction. The error terms are defined as

$$\varepsilon_x = \begin{bmatrix} \sigma_p^2 & \sigma_p\sigma_v \\ \sigma_v\sigma_p & \sigma_v^2 \end{bmatrix}, \tag{3.5}$$

$$\varepsilon_d = \sigma_d^2 \tag{3.6}$$

where $\sigma_p$ and $\sigma_v$ are the standard deviations associated with the postion and velocity error, respectively. These values are derived from process error:

$$\sigma_p = \sigma_{process}\frac{T^2}{2} \tag{3.7}$$

and

$$\sigma_v = \sigma_{process} * T \tag{3.8}$$

where $\sigma_{process}$ is the standard deviation defining the error distribution (with zero mean) for the process. This error is considered to be that induced by the factors that are unaccounted for, such as the acceleration of the quadcopter [7]. $\sigma_d$ is simply the standard deviation describing the distribution of measurement error. Both of these values must be estimated. Next, a covariance estimation, $\bar{P}_t$, is made as follows:

$$\bar{P}_t = A * P_{t-1} * A^T + \varepsilon_x \tag{3.2.3i}$$

where

$$\bar{P}_{t=1} = \varepsilon_x = \sigma^2_{process} * \begin{bmatrix} \frac{T^4}{4} & \frac{T^3}{2} \\ \frac{T^3}{2} & T^2 \end{bmatrix}$$

and $P_{t-1}$ is the updated covariance estimation from the previous sample. The Kalman gain is now defined as

$$K_t = \bar{P}_t * C^T * (C * P_t * C^T + \varepsilon_d)^{-1}. \quad (3.2.3j)$$

This gain is then used to calculate a new position estimaion using equation (3.2.3a). The final step is to update the covariance matrix with the newly calculated Kalman gain value:

$$P_t = (I - K_t * C) * \bar{P}_t \quad (3.2.3k)$$

where $I$ is the 2x2 identity matrix. This sequence of calculations is performed for each set of data (timestamp and data value pair). Figure 3.10 generally summarizes this iterative process.

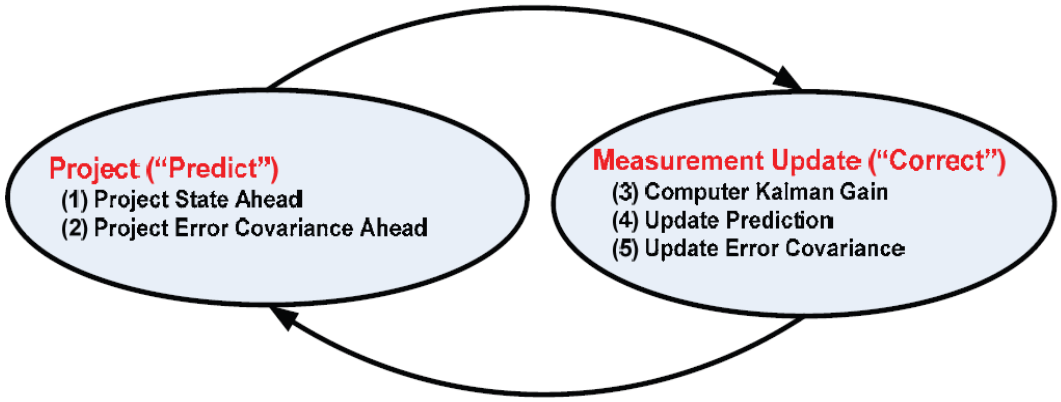



Figure 3.10: Iterative process for Kalman filtering [7]

The Kalman filter outlined here is programmed in Matlab (see Appendix C) as a universal filter that can independently be used for X, Y, or Z position data as well as for heading data. The filter uses two parameters that can be tuned to achieve optimal results based on the data being filtered. These parameters are $\sigma_{process}$ and $\sigma_d$. The ratio of these values determines how the filter 'weights' the state predictions and measurement predictions made. Two sets of these parameters are used, one for the position filter (same for X, Y, and Z positions) and one for the heading filter. Although the filter is designed to physically represent the X, Y, and Z positions of the quadcopter, it is applied to the calculated heading data as the observed result is acceptable. The filtering values used for each of the two filters are summarized in Table 3.1

Table 3.1: Filtering values for Kalman Filter

| Filter | $\sigma_{process}$ | $\sigma_d$ | $\sigma_{process}/\sigma_d$ |
|---|---|---|---|
| Position (X, Y, & Z) | 3 | 2 | 1.5 |
| Heading | 1 | 1 | 1 |

Figures 3.11 and 3.12 show the position data and heading data, respectively, before and after the Kalman filter is applied. In both cases, the filter effectively filters out the majority of the noise that exists and yields a waveform which is more continuous and manageable. Additional filters (averaging or frequency filtering) could be applied at this point, but for the purpose of this study, they are unnecessary.

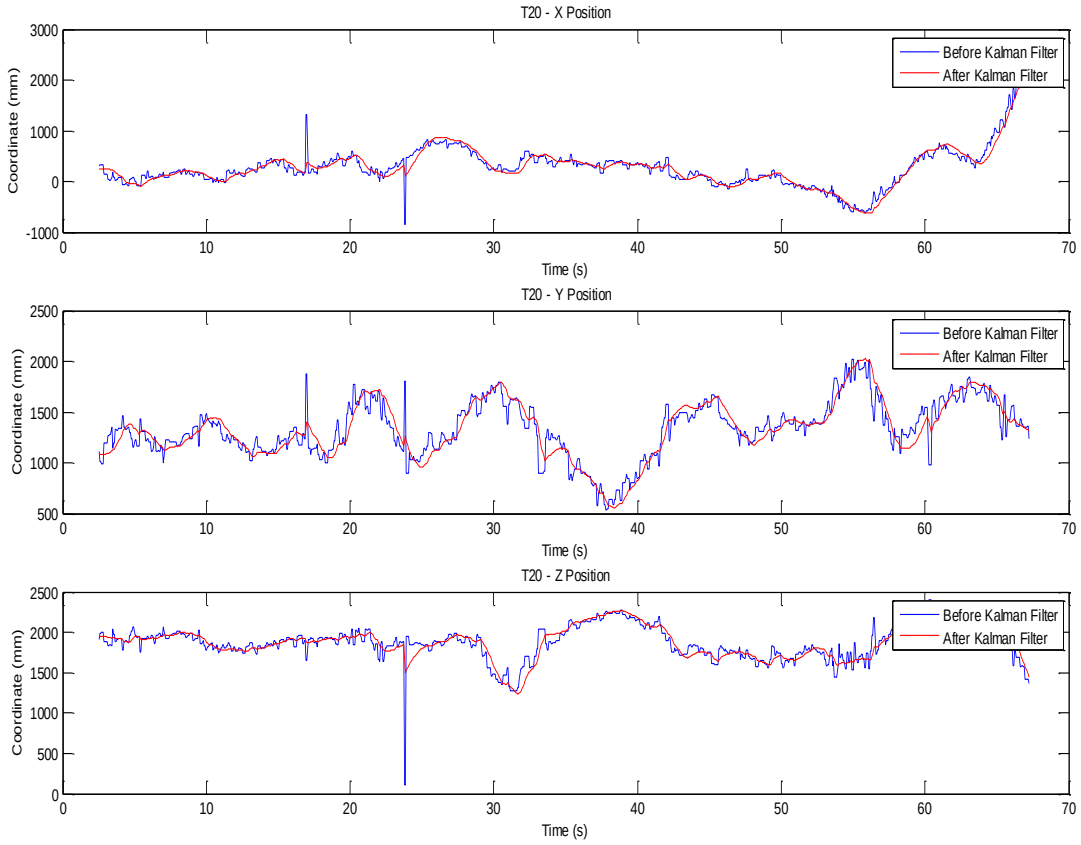



Figure 3.11: Position data filtered with Kalman Filter

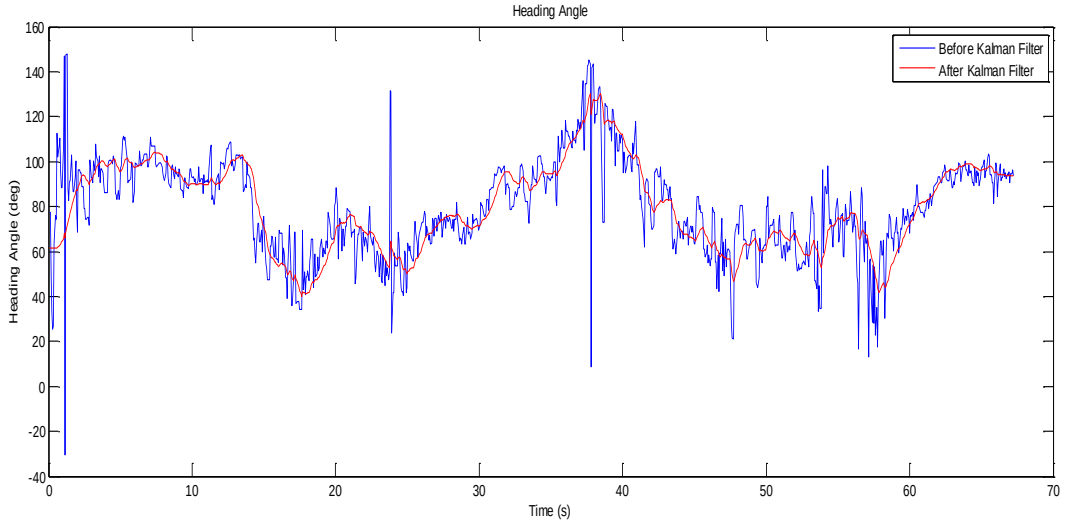



Figure 3.12: Heading angle data filtered with Kalman Filter

## 3.3 System Identification in Matlab

The final portion of this study involves the characterization of the system responses for the three most basic commands to the quadcopter. These are roll/pitch, throttle, and yaw/heading angle. The intention is to understand the system-level responses to step inputs for each of these types of movement. It should be noted that due to the symmetry of the quadcopter platform, it is assumed that the pitch response for front-to-back and side-to-side movements are identical. For this reason, it is only necessary to understand the system's response to one of these inputs in order to be able to model each of these similar responses. The front/back movement (Channel 2) is used.

For each movement type, an impulse command was sent to the quadcopter using one of the four available channels that corresponds to that particular response. Here the response to each of these inputs is brought into the System Indentification Toolbox within Matlab. Using this environment, a model is fitted to each response and a transfer function derived describing the response in terms of the input. These transfer functions are general and derived from data that is not entirely free from error. Therefore, these transfer functions serve as a basis for understanding the system-level operation of the F450 quadcopter, however, a more in-depth study is needed to fully characterize and model this system.

### *3.3.1 Thrust Modeling*

Figure 3.13 shows the altitude response to a thrust command. These datasets are used as the input and output waveforms for the System Identification Toolbox. It can be seen that there is some delay in the response and also that some error exists in the measured responses. The best fit model of low order (low complexity) is shown in Figure 3.14.

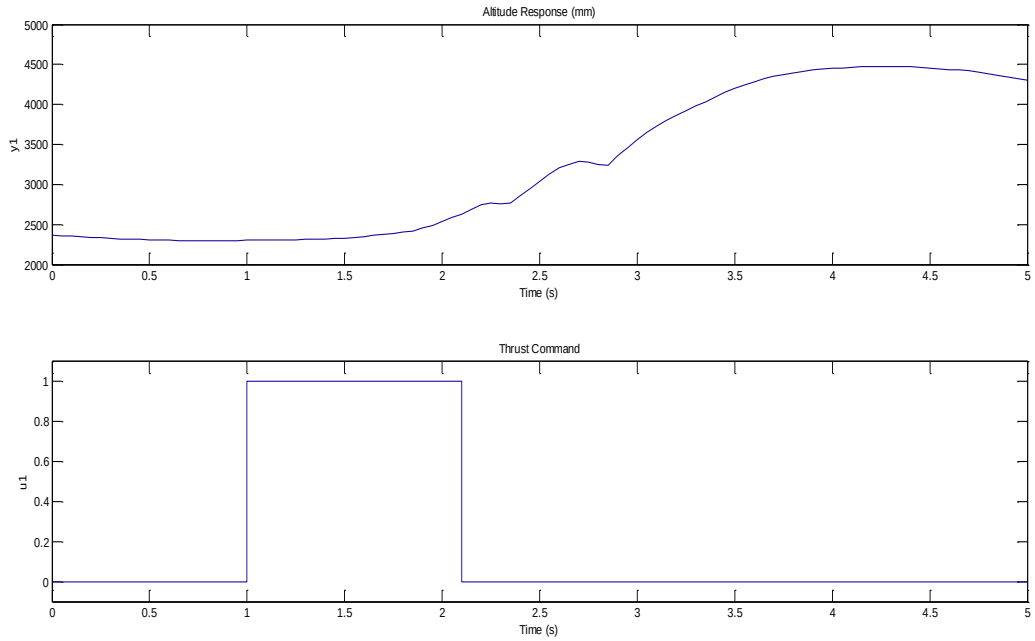



Figure 3.13: Thrust command and altitude response

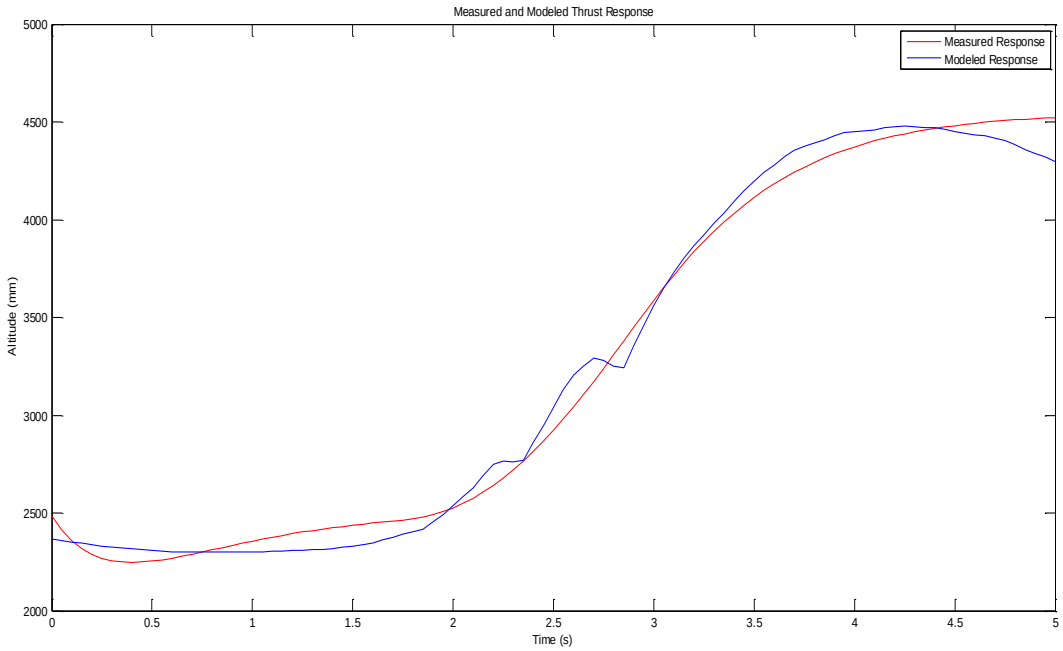



Figure 3.14: Measured and modeled response

The transfer function generated for this response is of the form

$$G(s) = \frac{K_p}{s(1+T_{p1}s)(1+T_{p2}s)} e^{-T_d s}$$

where

$K_p = 1856.4$,

$T_{p1} = 0.40885$,

$T_{p2} = 0.42283$, and

$T_d = 0.6585$.

These coefficients define the overall gain, locations of the real poles, and the time delay that characterize this response. Also notice that the denominator contains an integrator (s = 0). This is because the response must be integrated in order for an altitude to be output from the system (command is a thrust command not an altitude

command). The resulting transfer function that describes the altitude response to a given thrust command is

$$G_{thrust}(s) = \frac{1856.4}{s(1+0.40885s)(1+0.42283s)} e^{-0.6858s} \quad (3.3.1a)$$

The step response is shown in Figure 3.15.

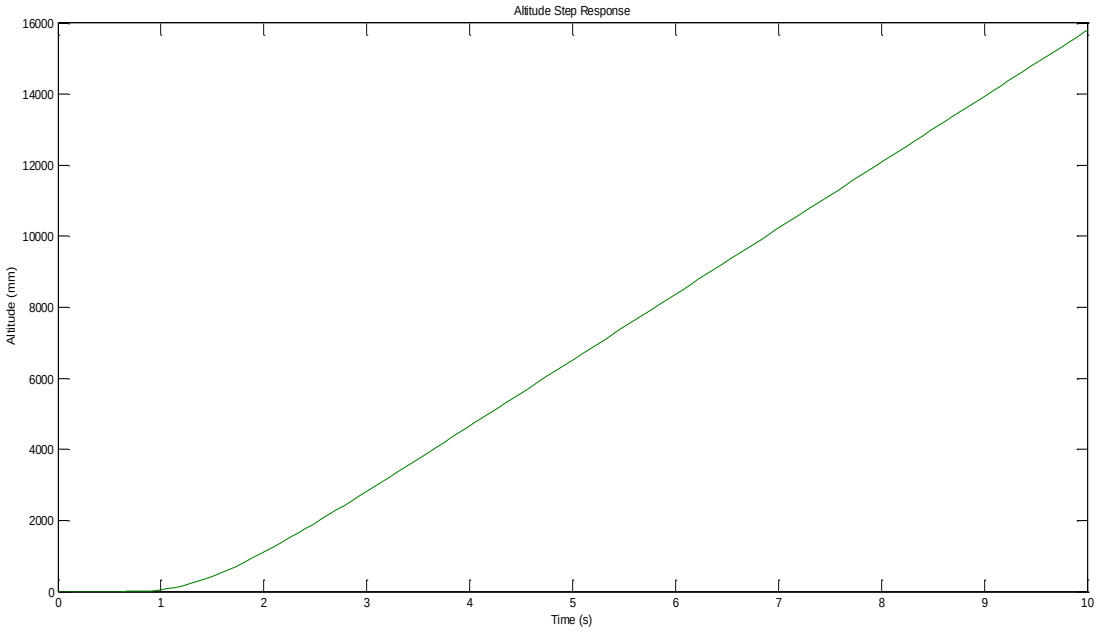



Figure 3.15: Altitude step response

### *3.3.2 Pitch Modeling*

The F450 quadcopter is capable of moving almost entirely laterally, and therefore, the term 'pitch' is not particularly accurate but will be used to describe both the side-to-side and front-to-back movements of the quadcopter. As previously noted, these movements are all expected to be the same or similar due to the symmetry of the architecture of the quadcopter platform. An impulse command of duration 0.5s is applied and the response measured. The command and response are shown in in Figure 3.16. The fitted model is shown in Figure 3.17.

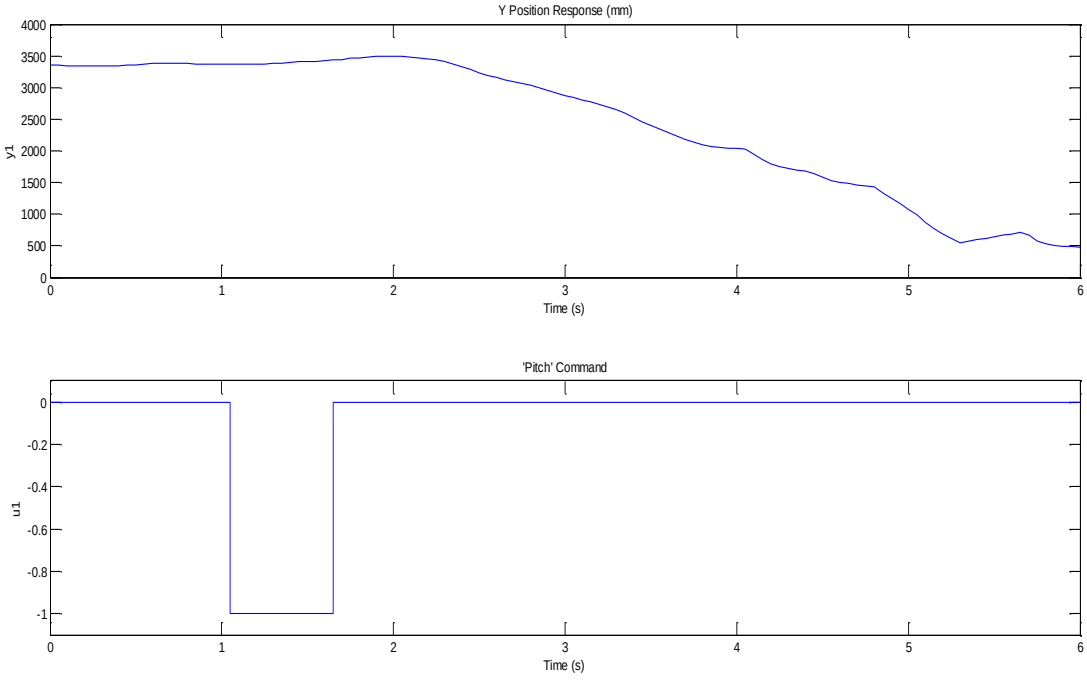



Figure 3.16: Pitch command and Y position response

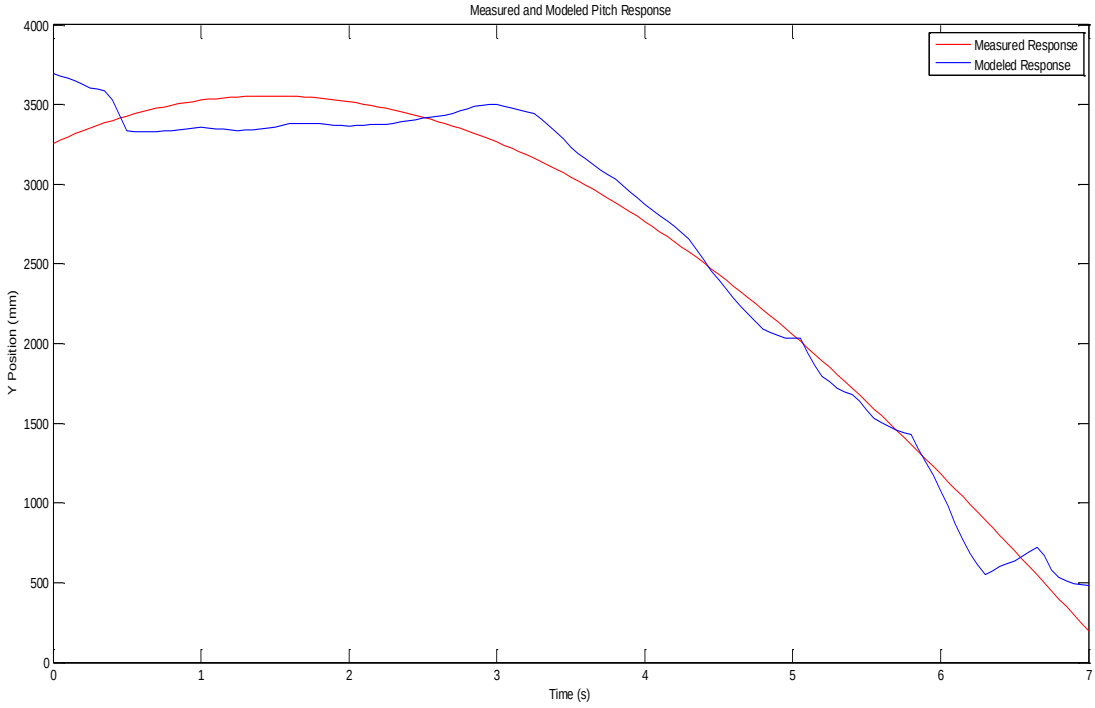



Figure 3.17: Measured and modeled pitch response

More error exists in the measure response for this type of movement, but a general transfer function can still be generated based on the measured response and know command impulse. This transfer function takes the form of

$$G(s) = \frac{K_p}{s(1+2\zeta T_w s+(T_w s)^2)} e^{-T_d s}$$

where

$K_p = 3425.9$,

$T_w = 5.4414$,

$\zeta = 0.40287$, and

$T_d = 0.36505$.

The form for this transfer function is different from that derived for thrust. The poles are underdamped and complex. Again, an integrator (1/s) is necessary in order to generate and output in terms of position as the command specifies a velocity. The final transfer function describing the position (X or Y) response to any 'pitch' command is

$$G_{pitch}(s) = \frac{3425.9}{s(1+2*0.40287*5.4414s+(5.4414s)^2)} e^{-0.36505s}.$$

The associated step response is shown in Figure 3.18. The underdamped nature of this transfer function is visible in the step response plot as a slight oscillation about the commanded position.

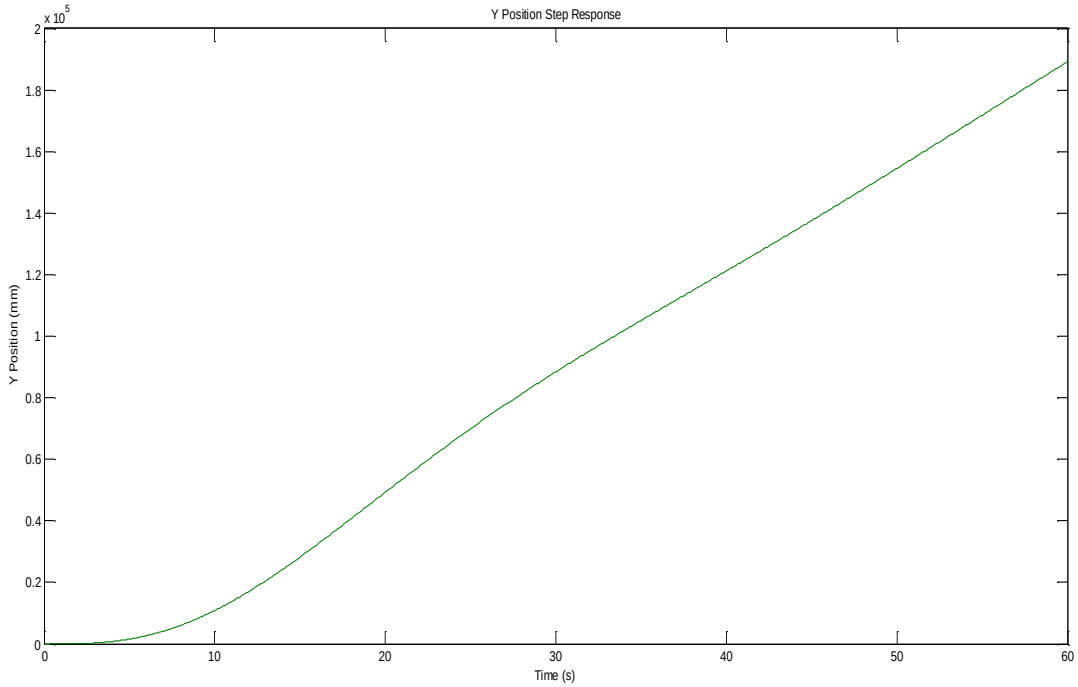



Figure 3.18: Y position step response

### *3.3.3 Yaw Rate Modeling*

Finally, the response to a yaw rate command is modeled based on the measured heading response to a known impulse command. The command and measured response are shown in Figure 3.19. The yaw rate command is roughly 1s in duration. Again, some error is prevalent in the measured heading (calculated from position measurement). However, a general transfer function can be found to model the response at hand. The modeled response is shown in Figure 3.20.

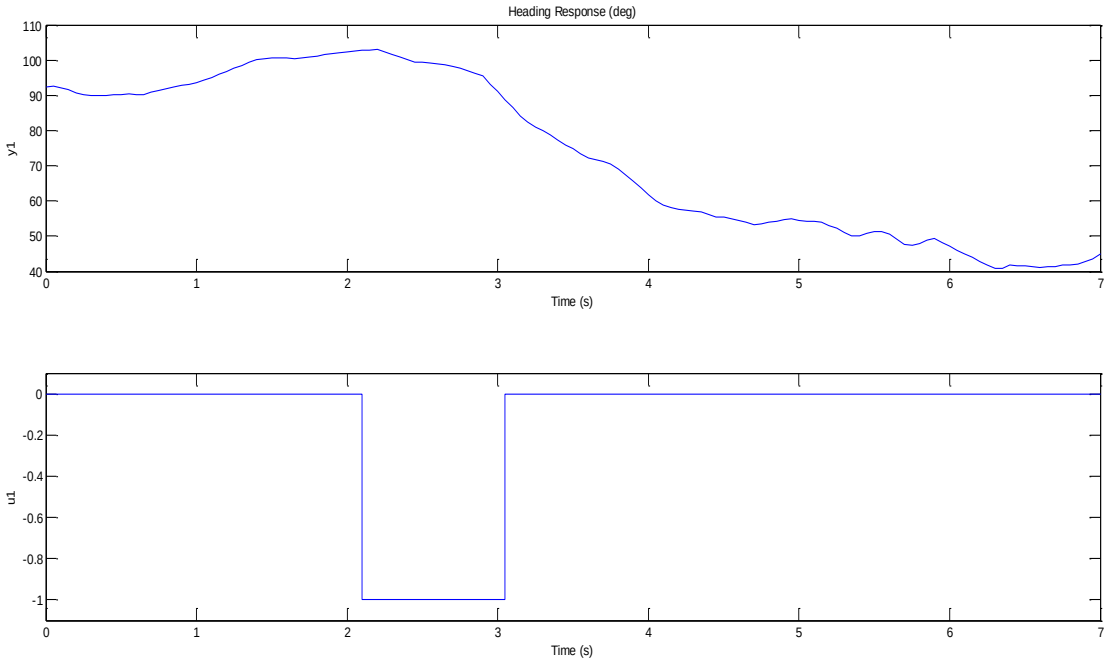



Figure 3.19: Yaw rate command and heading response

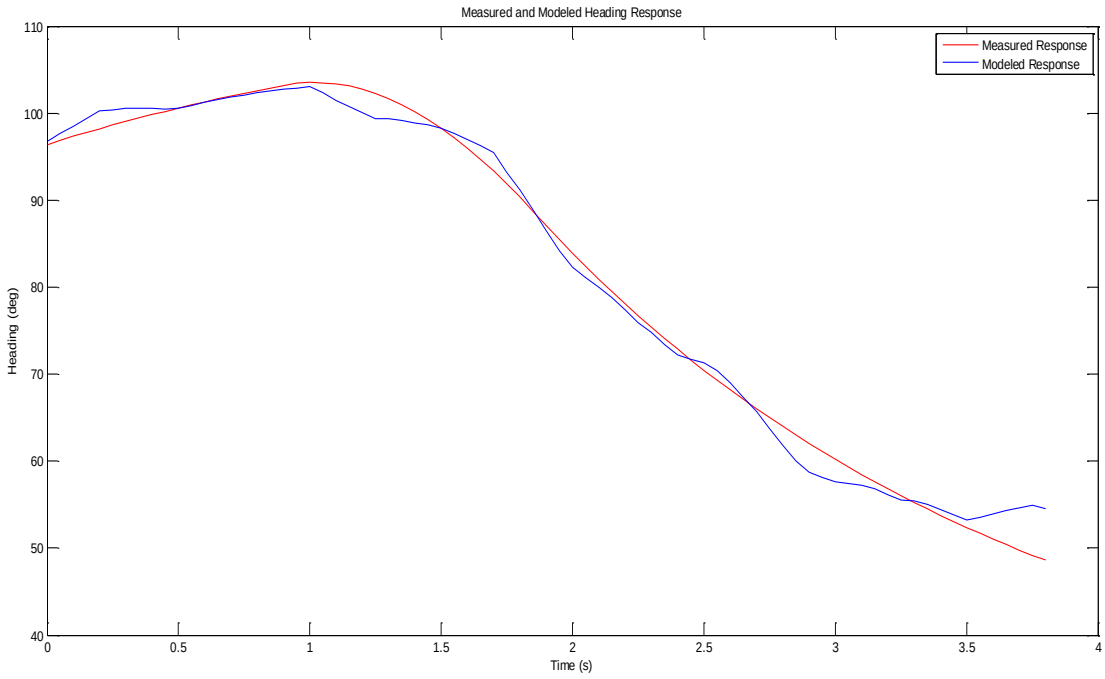



Figure 3.20: Measured and modeled heading response

The transfer function generated for this response takes the form of

$$G(s) = \frac{K_p}{s(1+T_{p1}s)} e^{-T_d s}$$

where

$K_p = 91.258$,

$T_{p1} = 1.8439$, and

$T_d = 0.0044$.

This is the most simple of the transfer functions generated yet. Being only a first order transfer function (with only one pole), it indicates a very stable heading

response to a yaw rate input. The final transfer function modeling heading response to a yaw rate command is

$$G_{Heading}(s) = \frac{91.258}{s(1+1.8439s)} e^{-0.0044s},$$

and the associated heading response to a step input is shown in Figure 3.21.

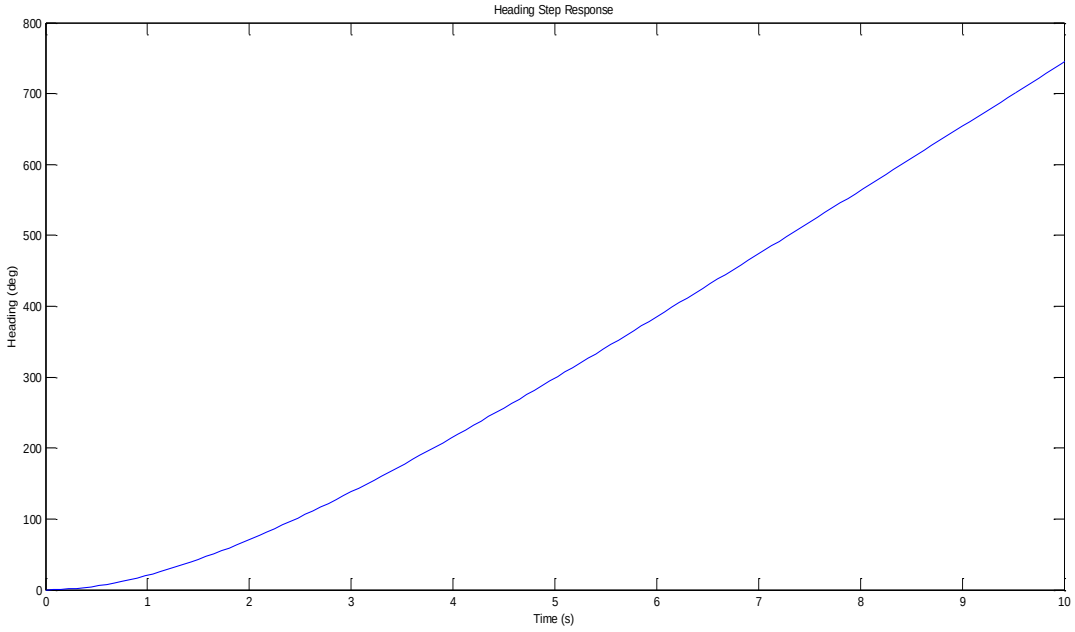



Figure 3.21: Heading step response

# CHAPTER IV

# CONCLUSIONS AND FUTURE WORK

## 4.1 Conclusions

This study presented the development of a quadcopter MAV localization system and the comparison of a variety of trilateration algorithms used to estimate the quadcopter's position in space. A physical system was developed and tested in parallel with the development of a a theoretical model using the Matlab software platform. A Monte Carlo simulation was performed and used to evaluate the accuracy and robustness of several trilateration algorithms, and the method employing Cayley-Menger Determinants was determined to be the most effective for the applicaton of the quadcopter MAV and six sensor localization system being used. This trlateration algorithm was further used to process real data acquired from flight testing.

Multiple controlled flight tests were conducted in order to gather data that were used to further understand both the positioning system and the high-level dynamics of the quadcopter MAV. Mulitple data filters were developed for purpose of handling the errors that are inherent to the sensor system used. Additionally, the

use of a tunable Kalman filter was developed and implemented and showed to effectively increase the robustness of the position estimation.

Finally, three of the quadcopter's fundamental dynamic responses were modeled using data from flight testing and the Matlab System Identification Toolbox. Simple linear transfer functions were fitted to the measured responses for thrust, pitch, and yaw rate commands (impulses). These modeled responses provide a basis for the control system that will need to be developed in order to automated the flight of the quadcopter MAV.

## 4.2 Future Work

The simulation and modeling performed in this study will serve as the foundation for control system design and localization system refinement in order to automate the navigation and flight of the quadcopter MAV. Additional design iterations of both software and hardware will allow for improved postion estimation reliability and accuracy which will, in turn, enable the development of a robust quadcopter controller.

# REFERENCES

[1] "HX19 Ultrasonic Positioning System." HX19 Ultrasonic Positioning System. Hexamite, 1999. Web. 2013. <http://www.hexamite.com/hx19.htm>.

[2] Hereman, Willy, and William S. Murphy, Jr. "Determination of a Position in Three Dimensions Using Trilateration and Approximate Distances."Decision Sciences (1995): 1-21.

[3] Manolakis, Di Metris E. "Efficient Solution and Performance Analysis of 3-D Position Estimation by Trilateration." IEEE Transactions on Aerospace and Electronic Systems 32.4 (1996): 1239-248.

[4] Hereman, Willy, and William S. Murphy, Jr. "Determination of a Position in Three Dimensions Using Trilateration and Approximate Distances." Decision Sciences (1995): 1-21.

[5] Liu, Hui, Houshang Darabi, Pat Banerjee, and Jing Liu. "Survey of Wireless Indoor Positioning Techniques and Systems." IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews) 37.6 (2007): 1067-080.

[6] Sobers, D. Michael, Jr., Girish Chowdhary, and Eric N. Johnson. "Indoor Navigation for Unmanned Aerial Vehicles." Proc. of AIAA Guidance, Navigation, and Control Conference, Chicago, IL. Reston, VA: American Institute of Aeronautics and Astronautics, 2009. 1-29.

[7] Shareef, Ali, and Yifeng Zhu. "Localization Using Extended Kalman Filters in Wireless Sensor Networks." www.intechweb.org. N.p., Apr. 2009.

APPENDIX A - Trilateration Calculations in Matlab

```
function [R_Simple]    =   SimpleMethod(NOR,RCM,MR)

% function  [R_Simple]   =   SimpleMethod(NOR,RCM,MR)
% inputs    :   NOR (number of receivers)
%           :   RCM (reciever coordinate matrix, NOR by 3 matrix)
%           :   MR  (measured range, NOR by 1 matrix)
% output    :   R_Simple (calculated x,y,z position using Simple
method

%-- input condition check
[r1,c1]     =   size(RCM);
[r2,c2]     =   size(MR);

if r1 ~= NOR || c1 ~= 3
    error('Receiver Coordinate Matrix must %d by %d',NOR,3)
elseif r2 ~= 1 || c2 ~= NOR
    error('Measured Range Matrix must be %d by %d',1,NOR')
elseif NOR ~= 3
    error('Simple Method only needs 3 receiver readings')
end



function [R_OP,i_count]     =   LeastSquare(NOR,RCM,MR,method)

% function [R_OP,i_count]    =   LeastSquare(NOR,RCM,MR)
% inputs    :   NOR (number of receivers)
%           :   RCM (reciever coordinate matrix, NOR by 3 matrix)
%           :   MR  (measured range, NOR by 1 matrix)
%           :   method -> 'nls' for nonlinear, 'ls' for linear
% outputs   :   R_OP (calculated x,y,z position based on 'method'
input)
%           :   i_count  (no. of iterations required to arrive at
position)


%-- input dimension check
[r1,c1]     =   size(RCM);
[r2,c2]     =   size(MR);

if NOR < 4
    error('Linear least square method not valid for NOR less than
4')
end
```

```
if r1 ~= NOR || c1 ~= 3
    error('Receiver Coordinate Matrix must %d by %d',r1,c1)
end

if r2 ~= 1 || c2 ~= NOR
    error('Measured Range Matrix must be %d by %d',r2,c2')
end

%-- linear least square method for initial guess
A   =   zeros(NOR-1,3);
b   =   zeros(NOR-1,1);

for i = 1:(NOR-1)
    A(i,1:3)    =   RCM(i+1,:) - RCM(1,:);
    b(i,1)      =   0.5*(MR(1)^2 - MR(i+1)^2 + norm(RCM(i+1,:)-
RCM(1,:))^2);
end

X           =   (A'*A)\(A'*b);      % linear least square solution
R_OP_LLS    =   X' + RCM(1,1:3);    % estimated position coordinate

i_count_ls = 1; % always 1 for linear least squares

%-- nonlinear least square method
i_count_nls = 0; % iterations before convergence

i   =   0;
tol =   1e-3;
while    tol > 1e-6 & i_count_nls <= 15

    i_count_nls = i_count_nls + 1;

    if i == 0
        Rold    =   R_OP_LLS;
    else
        Rold    =   Rnew;
    end

    x       =   Rold(1);
    y       =   Rold(2);
    z       =   Rold(3);

    JtJ     =   zeros(3,3);
    Jtf     =   zeros(3,1);

    if NOR == 4

        R1  =   MR(1);
        R2  =   MR(2);
        R3  =   MR(3);
        R4  =   MR(4);
```

```
    x1  =   RCM(1,1);   y1  =   RCM(1,2);   z1  =   RCM(1,3);
    x2  =   RCM(2,1);   y2  =   RCM(2,2);   z2  =   RCM(2,3);
    x3  =   RCM(3,1);   y3  =   RCM(3,2);   z3  =   RCM(3,3);
    x4  =   RCM(4,1);   y4  =   RCM(4,2);   z4  =   RCM(4,3);
    f1      =   sqrt((x-x1)^2 + (y-y1)^2 + (z-z1)^2) - R1;
    f2      =   sqrt((x-x2)^2 + (y-y2)^2 + (z-z2)^2) - R2;
    f3      =   sqrt((x-x3)^2 + (y-y3)^2 + (z-z3)^2) - R3;
    f4      =   sqrt((x-x4)^2 + (y-y4)^2 + (z-z4)^2) - R4;
    JtJ(1,1)    =   (x-x1)^2/(f1 + R1)^2 +...
                    (x-x2)^2/(f2 + R2)^2 +...
                    (x-x3)^2/(f3 + R3)^2 +...
                    (x-x4)^2/(f4 + R4)^2;
    JtJ(1,2)    =   (x-x1)*(y-y1)/(f1 + R1)^2 +...
                    (x-x2)*(y-y2)/(f2 + R2)^2 +...
                    (x-x3)*(y-y3)/(f3 + R3)^2 +...
                    (x-x4)*(y-y4)/(f4 + R4)^2;
    JtJ(1,3)    =   (x-x1)*(z-z1)/(f1 + R1)^2 +...
                    (x-x2)*(z-z2)/(f2 + R2)^2 +...
                    (x-x3)*(z-z3)/(f3 + R3)^2 +...
                    (x-x4)*(z-z4)/(f4 + R4)^2;
    JtJ(2,1)    =   JtJ(1,2);
    JtJ(2,2)    =   (y-y1)^2/(f1 + R1)^2 +...
                    (y-y2)^2/(f2 + R2)^2 +...
                    (y-y3)^2/(f3 + R3)^2 +...
                    (y-y4)^2/(f4 + R4)^2;
    JtJ(2,3)    =   (y-y1)*(z-z1)/(f1 + R1)^2 +...
                    (y-y2)*(z-z2)/(f2 + R2)^2 +...
                    (y-y3)*(z-z3)/(f3 + R3)^2 +...
                    (y-y4)*(z-z4)/(f4 + R4)^2;
    JtJ(3,1)    =   JtJ(1,3);
    JtJ(3,2)    =   JtJ(2,3);
    JtJ(3,3)    =   (z-z1)^2/(f1 + R1)^2 +...
                    (z-z2)^2/(f2 + R2)^2 +...
                    (z-z3)^2/(f3 + R3)^2 +...
                    (z-z4)^2/(f4 + R4)^2;
    Jtf(1,1)    =   (x-x1)*f1/(f1+R1) +...
                    (x-x2)*f2/(f2+R2) +...
                    (x-x3)*f3/(f3+R3) +...
                    (x-x4)*f4/(f4+R4);
    Jtf(2,1)    =   (y-y1)*f1/(f1+R1) +...
                    (y-y2)*f2/(f2+R2) +...
                    (y-y3)*f3/(f3+R3) +...
                    (y-y4)*f4/(f4+R4);
    Jtf(3,1)    =   (z-z1)*f1/(f1+R1) +...
                    (z-z2)*f2/(f2+R2) +...
                    (z-z3)*f3/(f3+R3) +...
                    (z-z4)*f4/(f4+R4);

elseif NOR == 5

    R1  =   MR(1);
    R2  =   MR(2);
    R3  =   MR(3);
```

```
R4  =   MR(4);
R5  =   MR(5);
x1  =   RCM(1,1);   y1  =   RCM(1,2);   z1  =   RCM(1,3);
x2  =   RCM(2,1);   y2  =   RCM(2,2);   z2  =   RCM(2,3);
x3  =   RCM(3,1);   y3  =   RCM(3,2);   z3  =   RCM(3,3);
x4  =   RCM(4,1);   y4  =   RCM(4,2);   z4  =   RCM(4,3);
x5  =   RCM(5,1);   y5  =   RCM(5,2);   z5  =   RCM(5,3);
f1      =   sqrt((x-x1)^2 + (y-y1)^2 + (z-z1)^2) - R1;
f2      =   sqrt((x-x2)^2 + (y-y2)^2 + (z-z2)^2) - R2;
f3      =   sqrt((x-x3)^2 + (y-y3)^2 + (z-z3)^2) - R3;
f4      =   sqrt((x-x4)^2 + (y-y4)^2 + (z-z4)^2) - R4;
f5      =   sqrt((x-x5)^2 + (y-y5)^2 + (z-z5)^2) - R5;
JtJ(1,1)    =   (x-x1)^2/(f1 + R1)^2 +...
                (x-x2)^2/(f2 + R2)^2 +...
                (x-x3)^2/(f3 + R3)^2 +...
                (x-x4)^2/(f4 + R4)^2 +...
                (x-x5)^2/(f5 + R5)^2;
JtJ(1,2)    =   (x-x1)*(y-y1)/(f1 + R1)^2 +...
                (x-x2)*(y-y2)/(f2 + R2)^2 +...
                (x-x3)*(y-y3)/(f3 + R3)^2 +...
                (x-x4)*(y-y4)/(f4 + R4)^2 +...
                (x-x5)*(y-y5)/(f5 + R5)^2;
JtJ(1,3)    =   (x-x1)*(z-z1)/(f1 + R1)^2 +...
                (x-x2)*(z-z2)/(f2 + R2)^2 +...
                (x-x3)*(z-z3)/(f3 + R3)^2 +...
                (x-x4)*(z-z5)/(f4 + R4)^2 +...
                (x-x5)*(z-z5)/(f5 + R5)^2;
JtJ(2,1)    =   JtJ(1,2);
JtJ(2,2)    =   (y-y1)^2/(f1 + R1)^2 +...
                (y-y2)^2/(f2 + R2)^2 +...
                (y-y3)^2/(f3 + R3)^2 +...
                (y-y4)^2/(f4 + R4)^2 +...
                (y-y5)^2/(f5 + R5)^2;
JtJ(2,3)    =   (y-y1)*(z-z1)/(f1 + R1)^2 +...
                (y-y2)*(z-z2)/(f2 + R2)^2 +...
                (y-y3)*(z-z3)/(f3 + R3)^2 +...
                (y-y4)*(z-z4)/(f4 + R4)^2 +...
                (y-y5)*(z-z5)/(f5 + R5)^2;
JtJ(3,1)    =   JtJ(1,3);
JtJ(3,2)    =   JtJ(2,3);
JtJ(3,3)    =   (z-z1)^2/(f1 + R1)^2 +...
                (z-z2)^2/(f2 + R2)^2 +...
                (z-z3)^2/(f3 + R3)^2 +...
                (z-z4)^2/(f4 + R4)^2 +...
                (z-z5)^2/(f5 + R5)^2;
Jtf(1,1)    =   (x-x1)*f1/(f1+R1) +...
                (x-x2)*f2/(f2+R2) +...
                (x-x3)*f3/(f3+R3) +...
                (x-x4)*f4/(f4+R4) +...
                (x-x5)*f5/(f5+R5);
Jtf(2,1)    =   (y-y1)*f1/(f1+R1) +...
                (y-y2)*f2/(f2+R2) +...
                (y-y3)*f3/(f3+R3) +...
                (y-y4)*f4/(f4+R4) +...
```

```
                    (y-y5)*f5/(f5+R5);
    Jtf(3,1)    =   (z-z1)*f1/(f1+R1) +...
                    (z-z2)*f2/(f2+R2) +...
                    (z-z3)*f3/(f3+R3) +...
                    (z-z4)*f4/(f4+R4) +...
                    (z-z5)*f4/(f5+R5);
else

    R1  =   MR(1);
    R2  =   MR(2);
    R3  =   MR(3);
    R4  =   MR(4);
    R5  =   MR(5);
    R6  =   MR(6);
    x1  =   RCM(1,1);   y1  =   RCM(1,2);   z1  =   RCM(1,3);
    x2  =   RCM(2,1);   y2  =   RCM(2,2);   z2  =   RCM(2,3);
    x3  =   RCM(3,1);   y3  =   RCM(3,2);   z3  =   RCM(3,3);
    x4  =   RCM(4,1);   y4  =   RCM(4,2);   z4  =   RCM(4,3);
    x5  =   RCM(5,1);   y5  =   RCM(5,2);   z5  =   RCM(5,3);
    x6  =   RCM(6,1);   y6  =   RCM(6,2);   z6  =   RCM(6,3);
    f1      =   sqrt((x-x1)^2 + (y-y1)^2 + (z-z1)^2) - R1;
    f2      =   sqrt((x-x2)^2 + (y-y2)^2 + (z-z2)^2) - R2;
    f3      =   sqrt((x-x3)^2 + (y-y3)^2 + (z-z3)^2) - R3;
    f4      =   sqrt((x-x4)^2 + (y-y4)^2 + (z-z4)^2) - R4;
    f5      =   sqrt((x-x5)^2 + (y-y5)^2 + (z-z5)^2) - R5;
    f6      =   sqrt((x-x6)^2 + (y-y6)^2 + (z-z6)^2) - R6;
    JtJ(1,1)    =   (x-x1)^2/(f1 + R1)^2 +...
                    (x-x2)^2/(f2 + R2)^2 +...
                    (x-x3)^2/(f3 + R3)^2 +...
                    (x-x4)^2/(f4 + R4)^2 +...
                    (x-x5)^2/(f5 + R5)^2 +...
                    (x-x6)^2/(f6 + R6)^2;
    JtJ(1,2)    =   (x-x1)*(y-y1)/(f1 + R1)^2 +...
                    (x-x2)*(y-y2)/(f2 + R2)^2 +...
                    (x-x3)*(y-y3)/(f3 + R3)^2 +...
                    (x-x4)*(y-y4)/(f4 + R4)^2 +...
                    (x-x5)*(y-y5)/(f5 + R5)^2 +...
                    (x-x6)*(y-y6)/(f6 + R6)^2;
    JtJ(1,3)    =   (x-x1)*(z-z1)/(f1 + R1)^2 +...
                    (x-x2)*(z-z2)/(f2 + R2)^2 +...
                    (x-x3)*(z-z3)/(f3 + R3)^2 +...
                    (x-x4)*(z-z4)/(f4 + R4)^2 +...
                    (x-x5)*(z-z5)/(f5 + R5)^2 +...
                    (x-x6)*(z-z6)/(f6 + R6)^2;
    JtJ(2,1)    =   JtJ(1,2);
    JtJ(2,2)    =   (y-y1)^2/(f1 + R1)^2 +...
                    (y-y2)^2/(f2 + R2)^2 +...
                    (y-y3)^2/(f3 + R3)^2 +...
                    (y-y4)^2/(f4 + R4)^2 +...
                    (y-y5)^2/(f5 + R5)^2 +...
                    (y-y6)^2/(f6 + R6)^2;
    JtJ(2,3)    =   (y-y1)*(z-z1)/(f1 + R1)^2 +...
                    (y-y2)*(z-z2)/(f2 + R2)^2 +...
```

```
                        (y-y3)*(z-z3)/(f3 + R3)^2 +...
                        (y-y4)*(z-z4)/(f4 + R4)^2 +...
                        (y-y5)*(z-z5)/(f5 + R5)^2 +...
                        (y-y6)*(z-z6)/(f6 + R6)^2;
        JtJ(3,1)    =   JtJ(1,3);
        JtJ(3,2)    =   JtJ(2,3);
        JtJ(3,3)    =   (z-z1)^2/(f1 + R1)^2 +...
                        (z-z2)^2/(f2 + R2)^2 +...
                        (z-z3)^2/(f3 + R3)^2 +...
                        (z-z4)^2/(f4 + R4)^2 +...
                        (z-z5)^2/(f5 + R5)^2 +...
                        (z-z6)^2/(f6 + R6)^2;
        Jtf(1,1)    =   (x-x1)*f1/(f1+R1) +...
                        (x-x2)*f2/(f2+R2) +...
                        (x-x3)*f3/(f3+R3) +...
                        (x-x4)*f4/(f4+R4) +...
                        (x-x5)*f5/(f5+R5) +...
                        (x-x6)*f6/(f6+R6);
        Jtf(2,1)    =   (y-y1)*f1/(f1+R1) +...
                        (y-y2)*f2/(f2+R2) +...
                        (y-y3)*f3/(f3+R3) +...
                        (y-y4)*f4/(f4+R4) +...
                        (y-y5)*f5/(f5+R5) +...
                        (y-y6)*f6/(f6+R6);
        Jtf(3,1)    =   (z-z1)*f1/(f1+R1) +...
                        (z-z2)*f2/(f2+R2) +...
                        (z-z3)*f3/(f3+R3) +...
                        (z-z4)*f4/(f4+R4) +...
                        (z-z5)*f5/(f5+R5) +...
                        (z-z6)*f6/(f6+R6);
    end

    Rnew    =   Rold' - JtJ\Jtf;
    Rnew    =   Rnew';
    tol     =   abs(norm(Rnew - Rold));
    i = i + 1;
end

R_OP_NLS = Rnew;

if strcmp(method,'ls')
    R_OP    =   R_OP_LLS;
    i_count = i_count_ls;
elseif strcmp(method,'nls')
    R_OP    =   R_OP_NLS;
    i_count = i_count_nls;
end
```

```
function    [R_OP_CFP]    =    ExplicitMethod_CFP(NOR,RCM,MR)

% function  [R_OP_CFP]    =    ExplicitMethod_CFP(NOR,RCM,MR)
% inputs    :    NOR (number of receivers)
%           :    RCM (reciever coordinate matrix, NOR by 3 matrix)
%           :    MR  (measured range, NOR by 1 matrix)
% output    :    R_OP_CFP (calculated x,y,z position using CFP method

%-- input condition check
[r1,c1]     =    size(RCM);
[r2,c2]     =    size(MR);

if r1 ~= NOR || c1 ~= 3
    error('Receiver Coordinate Matrix must %d by %d',NOR,3)
elseif r2 ~= 1 || c2 ~= NOR
    error('Measured Range Matrix must be %d by %d',1,NOR')
elseif NOR ~= 3
    error('Explicit Method only needs 3 receiver readings')
end

%-- constants for Closed-Form Solution

R1 = MR(1);
R2 = MR(2);
R3 = MR(3);

x1 = RCM(1,1);
x2 = RCM(2,1);
x3 = RCM(3,1);

y1 = RCM(1,2);
y2 = RCM(2,2);
y3 = RCM(3,2);

z1 = RCM(1,3);
z2 = RCM(2,3);
z3 = RCM(3,3);

x21 = x2 - x1;
x31 = x3 - x1;

y21 = y2 - y1;
y31 = y3 - y1;

z21 = z2 - z1;
z31 = z3 - z1;

W = zeros(2,2);
d = zeros(2,1);
rh = zeros(2,1);
beta = zeros(2,1);
```

```
W(1,1) = x21;
W(2,1) = x31;
W(1,2) = y21;
W(2,2) = y31;

W_inv = inv(W);

w1 = (W_inv(1,:))';
w2 = (W_inv(2,:))';

d(1) = z21;
d(2) = z31;

rh1 = [x1; y1];

S1 = norm(RCM(1,:));
S2 = norm(RCM(2,:));
S3 = norm(RCM(3,:));

S = [S1 S2 S3];

a_cf = 1 + ((d'/W')/W)*d;

delta2 = sqrt(S(1)^2 - S(2)^2);
delta3 = sqrt(S(1)^2 - S(3)^2);

delta = [delta2^2; delta3^2];

G = ((W')^(-1))/W;
e = (((1/2)*delta'/W')/W + rh1'/W)';

g11 = G(1,1);
g12 = G(1,2);
g21 = G(2,1);
g22 = G(2,2);

e1 = e(1);
e2 = e(2);

lam0 = -(2*rh1'/W*d - 2*z1 + d'/W'/W*delta)/(2*a_cf);
lam1 = (z21*(g11 + g12) + z31*(g22 + g12))/(2*a_cf);
lam2 = -(z21*g11 + z31*g12)/(2*a_cf);
lam3 = -(z31*g22 + z21*g12)/(2*a_cf);

lambda = [lam0; lam1; lam2; lam3];

lambda1 = [-w1'*d*lambda + (-w1'*delta)/2; ...
        -w1'*d*lambda + (y31 - y21)/(2*det(W)); ...
        -w1'*d*lambda + (-y31)/(2*det(W)); ...
```

```
        -w1'*d*lambda + y21/(2*det(W))];

lambda2 = [-w2'*d*lambda + (-w2'*delta)/2; ...
        -w2'*d*lambda + (x21 - x31)/(2*det(W)); ...
        -w2'*d*lambda + (-x31)/(2*det(W)); ...
        -w2'*d*lambda + (-x21)/(2*det(W))];

lambda3 = lambda;

Lambda_ = [lambda1(1) lambda1(6) lambda1(10) lambda1(14); ...
            lambda2(1) lambda2(6) lambda2(10) lambda2(14); ...
            lambda3'];

zeta0 = lam0^2 - (delta'/W'/W*delta/4 + rh1'/W*delta + S1^2)/a_cf;
zeta1 = 2*lam0*lam1 + (1 + e1 + e2)/a_cf;
zeta2 = 2*lam0*lam2 - e1/a_cf;
zeta3 = 2*lam0*lam3 - e2/a_cf;
zeta4 = lam1^2 - (g11 + 2*g12 + g22)/(4*a_cf);
zeta5 = lam2^2 - g11/(4*a_cf);
zeta6 = lam3^2 - g22/(4*a_cf);
zeta7 = 2*lam1*lam2 + (g11 + g12)/(2*a_cf);
zeta8 = 2*lam1*lam3 + (g22 + g12)/(2*a_cf);
zeta9 = 2*lam2*lam3 - g12/(2*a_cf);

zeta = [zeta0; zeta1; zeta2; zeta3; zeta4; ...
    zeta5; zeta6; zeta7; zeta8; zeta9];

mu = [(-w1'*d) (-w2'*d) 1];

%% Closed Form Solution

% Original Position Calculation:
beta2 = (R1^2 - R2^2 - S1^2 + S2^2)/2;
beta3 = (R1^2 - R3^2 - S1^2 + S3^2)/2;

beta(1) = beta2;
beta(2) = beta3;

b_cf = 2*(rh1'/W)*d - 2*z1 - ((2*d'/W')/W)*beta;
c_cf = S1^2 - R1^2 + ((beta'/W')/W)*beta - (2*rh1'/W)*beta;

z_cf = (-b_cf + sqrt(b_cf^2 - 4*a_cf*c_cf))/(2*a_cf);

rh = W\(beta - d*z_cf);

R_cfp1 = [rh(1) rh(2) z_cf];

% Improved Position Calculation (z calc):
u_cf = [1; R1^2; R2^2; R3^2];
v_cf = [1; R1^2; R2^2; R3^2; R1^4; R2^4; R3^4; ...
        R1^2*R2^2; R1^2*R3^2; R2^2*R3^2];
```

```
R_cfp2 = (Lambda_*u_cf)' + mu*(zeta'*v_cf)^(1/2);

R_OP_CFP   =    R_cfp2;



function    [R_OP_CMD]   =   ExplicitMethod_CMD(NOR,RCM,MR)

% function  [R_OP_CMD]   =   ExplicitMethod_CMD(NOR,RCM,MR)
% inputs    :   NOR (number of receivers)
%           :   RCM (reciever coordinate matrix, NOR by 3 matrix)
%           :   MR  (measured range, NOR by 1 matrix)
% output    :   R_OP_CMD (calculated x,y,z position using CMD
method)

%-- input condition check
[r1,c1]     =   size(RCM);
[r2,c2]     =   size(MR);

if r1 ~= NOR || c1 ~= 3
    error('Receiver Coordinate Matrix must %d by %d',NOR,3)
elseif r2 ~= 1 || c2 ~= NOR
    error('Measured Range Matrix must be %d by %d',1,NOR')
elseif NOR ~= 3
    error('Explicit Method only needs 3 receiver readings')
end

%-- constants for Cayley-Menger Determinants
P1  =   RCM(1,:);
P2  =   RCM(2,:);
P3  =   RCM(3,:);
v1  =   P2 - P1;
v2  =   P3 - P1;
l1  =   MR(1);
l2  =   MR(2);
l3  =   MR(3);
%-- Cayley-Menger Determinant calculations
CMD_p12 =   (norm(P2 - P1))^2;
CMD_p13 =   (norm(P3 - P1))^2;
CMD_p23 =   (norm(P3 - P2))^2;
CMD_k   =   (norm(cross(P2 - P1,P3 - P1)))^2;
%-- Cayley-Menger Bi-determinant solution
CMBD_k1 =   -(1/4)*det([0   1       1       1; ...
                        1   0       CMD_p13 l1^2; ...
                        1   CMD_p12 CMD_p23 l2^2; ...
                        1   CMD_p13 0       l3^2]);

CMBD_k2 =   -(1/4)*det([0   1       1       1; ...
                        1   0       CMD_p12 l1^2; ...
```

```
                        1   CMD_p12 0       l2^2; ...
                        1   CMD_p13 CMD_p23 l3^2]);

CMBD_k3 =   (1/8)*det([ 0   1           1           1           1;
...
                        1   0           CMD_p12     CMD_p13
l1^2; ...
                        1   CMD_p12     0           CMD_p23
l2^2; ...
                        1   CMD_p13     CMD_p23     0
l3^2; ...
                        1   l1^2        l2^2        l3^2        0]);

k1  =   -(CMBD_k1/CMD_k);
k2  =   CMBD_k2/CMD_k;
k3  =   sqrt(CMBD_k3)/CMD_k;
P   =   P1 + k1*v1 + k2*v2;
P4  =   P + k3*(cross(v1,v2));

R_OP_CMD   =    P4;
```

APPENDIX B – Monte Carlo Simulation Matlab Program

```
%% Trilateration Error Analysis - Monte Carlo Simulation


clc
clear

addpath('H:\Masters Thesis\Matlab\Test Data Processing');
addpath('H:\Masters Thesis\Matlab\Test Data
Processing\Trilateration');

%% Set Parameters for Analysis:

% Specify z location of slice for analysis:
Z_Plane_Value = 5000; % in mm, height at which slice is taken

% Specify number of points:
Num_Norm_Pts = 100000; % Number of random points for simulation

% Specify BIN size for error analysis:
BIN_Size = 1000; % This dictates the BIN size for XYZ Pixels (round
to nearest 'BIN_Size')

% Define space (in mm):
Xmin = -8000;
Xmax = 8000;
Ymin = -8000;
Ymax = 8000;
% Zmin = Z_Plane_Value - BIN_Size;
% Zmax = Z_Plane_Value + BIN_Size;
Zmin = Z_Plane_Value;
Zmax = Z_Plane_Value;

% Define max error for each receiver (in mm):
E40max = 15;
E41max = 15;
E42max = 15;
E43max = 15;
E44max = 15;
E45max = 15;

%% Define Sensor System, Helicopter Space & Other Parameters:

% Define Sensor Position Coordinates:

NOR = 6; % Number of receivers
```

```
Rx40    =   [325.00,  533.85,  22.00];
Rx41    =   [975.00,  533.85,  22.00];
Rx42    =   [650.00,    0.00,  22.00];
Rx43    =   [650.00, 1125.83,   0.00];
Rx44    =   [  0.00,    0.00,   0.00];
Rx45    =   [1300.0,    0.00,   0.00];


% Alter sensor position (pseudo position) for Simple Method:

Rx1_SM  = [     0,       0,      0];
Rx2_SM  = [1295.4,       0,      0];
Rx3_SM  = [     0,  1295.4,      0];

RCM     = [Rx40;Rx41;Rx42;Rx43;Rx44;Rx45];
RCM_SM  = [Rx1_SM; Rx2_SM; Rx3_SM]; % for Simple Method calculation
only
RCM_3   = [Rx43; Rx44; Rx45]; % for algorithms that only use 3
receivers

% Apply translation to center receivers at (0,0,0):
T_RCM = [-650   -375.278    0; ...
         -650   -375.278    0; ...
         -650   -375.278    0; ...
         -650   -375.278    0; ...
         -650   -375.278    0; ...
         -650   -375.278    0];

T_RCM_3 = [-650   -375.278    0; ...
           -650   -375.278    0; ...
           -650   -375.278    0];

RCM = RCM + T_RCM;

RCM_3 = RCM_3 + T_RCM_3;

% Redefine

Xavg = (Xmin + Xmax)/2;
Yavg = (Ymin + Ymax)/2;
Zavg = (Zmin + Zmax)/2;

Xrange = Xmax - Xmin;
Yrange = Ymax - Ymin;
Zrange = Zmax - Zmin;

% Define std deviation for each receiver:
Sigma_Per_Band = 3; % number of sigmnas per band (upper & lower
symmetrical over mean)
```

```
Sigma40 = E40max/Sigma_Per_Band;
Sigma41 = E41max/Sigma_Per_Band;
Sigma42 = E42max/Sigma_Per_Band;
Sigma43 = E43max/Sigma_Per_Band;
Sigma44 = E44max/Sigma_Per_Band;
Sigma45 = E45max/Sigma_Per_Band;

%% Generate Actual Distances and Sensor Output:

% Generate random actual position of quadcopter by coordinate (x, y,
& z)
Actual_Pos_X = ((Xmax-Xmin))*(rand(Num_Norm_Pts,1)-0.5) + mean([Xmin
Xmax]);
Actual_Pos_Y = ((Ymax-Ymin))*(rand(Num_Norm_Pts,1)-0.5) + mean([Ymin
Ymax]);
Actual_Pos_Z = ((Zmax-Zmin))*(rand(Num_Norm_Pts,1)-0.5) + mean([Zmin
Zmax]);

Actual_Pos = [Actual_Pos_X Actual_Pos_Y Actual_Pos_Z];

% Generate normally distributed error
Error_Dist_R40 = Sigma40*randn(Num_Norm_Pts,1); % Mean of 0 with Std
Dev specified above for each receiver
Error_Dist_R41 = Sigma41*randn(Num_Norm_Pts,1);
Error_Dist_R42 = Sigma42*randn(Num_Norm_Pts,1);
Error_Dist_R43 = Sigma43*randn(Num_Norm_Pts,1);
Error_Dist_R44 = Sigma44*randn(Num_Norm_Pts,1);
Error_Dist_R45 = Sigma45*randn(Num_Norm_Pts,1);

% Plot histogram of error distribution for each receiver:
figure(1); clf;
subplot(321); hist(Error_Dist_R40); xlabel('Error BIN (mm)');
ylabel('Frequency'); title('Histogram of Error Values: Rx1');
axis([-30 30 0 40000]);
subplot(322); hist(Error_Dist_R41); xlabel('Error BIN (mm)');
ylabel('Frequency'); title('Histogram of Error Values: Rx2');
axis([-30 30 0 40000]);
subplot(323); hist(Error_Dist_R42); xlabel('Error BIN (mm)');
ylabel('Frequency'); title('Histogram of Error Values: Rx3');
axis([-30 30 0 40000]);
subplot(324); hist(Error_Dist_R43); xlabel('Error BIN (mm)');
ylabel('Frequency'); title('Histogram of Error Values: Rx4');
axis([-30 30 0 40000]);
subplot(325); hist(Error_Dist_R44); xlabel('Error BIN (mm)');
ylabel('Frequency'); title('Histogram of Error Values: Rx5');
axis([-30 30 0 40000]);
subplot(326); hist(Error_Dist_R45); xlabel('Error BIN (mm)');
ylabel('Frequency'); title('Histogram of Error Values: Rx6');
axis([-30 30 0 40000]);

Error_Dist = [Error_Dist_R40 Error_Dist_R41 Error_Dist_R42
Error_Dist_R43 ...
```

```
                Error_Dist_R44 Error_Dist_R45];

Error_Dist_3 = [Error_Dist_R43 Error_Dist_R44 Error_Dist_R45];

Actual_Distance     = zeros(Num_Norm_Pts,NOR);

Pos_LS              = zeros(Num_Norm_Pts,3);
Error_LS            = zeros(Num_Norm_Pts,3);
Pos_NLS             = zeros(Num_Norm_Pts,3);
Error_NLS           = zeros(Num_Norm_Pts,3);
Pos_CMD             = zeros(Num_Norm_Pts,3);
Error_CMD           = zeros(Num_Norm_Pts,3);
Pos_CFP             = zeros(Num_Norm_Pts,3);
Error_CFP           = zeros(Num_Norm_Pts,3);

for i=1:Num_Norm_Pts
    Actual_Distance(i,1) = norm(Actual_Pos(i,:) - RCM(1,:));
    Actual_Distance(i,2) = norm(Actual_Pos(i,:) - RCM(2,:));
    Actual_Distance(i,3) = norm(Actual_Pos(i,:) - RCM(3,:));
    Actual_Distance(i,4) = norm(Actual_Pos(i,:) - RCM(4,:));
    Actual_Distance(i,5) = norm(Actual_Pos(i,:) - RCM(5,:));
    Actual_Distance(i,6) = norm(Actual_Pos(i,:) - RCM(6,:));
end

Actual_Distance_3 = [Actual_Distance(:,4) Actual_Distance(:,5)
Actual_Distance(:,6)]; % Simply use Actual_Distance calc for Rx43,
Rx44, & Rx45

Sensor_Output = Actual_Distance + Error_Dist;

Sensor_Output_3 = Actual_Distance_3 + Error_Dist_3;

% Initiate iteration counting variabls for 'ls' and 'nls' methods:
i_count_ls = zeros(Num_Norm_Pts,1);
i_count_nls = zeros(Num_Norm_Pts,1);

%% Calculate positions using different methods:

for i = 1:Num_Norm_Pts

    [Pos_LS(i,:), i_count_ls(i)]    =
LeastSquare(NOR,RCM,Sensor_Output(i,:),'ls');
    Error_LS(i,:)                   = Pos_LS(i,:) - Actual_Pos(i,:);

    [Pos_NLS(i,:), i_count_nls(i)]  =
LeastSquare(NOR,RCM,Sensor_Output(i,:),'nls');
    Error_NLS(i,:)                  = Pos_NLS(i,:) -
Actual_Pos(i,:);

    Pos_CMD(i,:)                    =
ExplicitMethod_CMD(3,RCM_3,Sensor_Output_3(i,:));
```

```
    Error_CMD(i,:)                  = Pos_CMD(i,:) -
Actual_Pos(i,:);

    Pos_CFP(i,:)                    =
ExplicitMethod_CFP(3,RCM_3,Sensor_Output_3(i,:));
    Error_CFP(i,:)                  = Pos_CFP(i,:) -
Actual_Pos(i,:);

end

%% Define Pixel Size for Error Analysis (NxNxN cube):

XYZ_BIN = FindBIN(Actual_Pos, BIN_Size); %

% Shift BIN values so none are negative and all are real:
XYZ_BIN = real(XYZ_BIN); % Isolate real components
XYZ_BIN = [XYZ_BIN(:,1) - Xmin/BIN_Size + 1, ... % Shift BINs so
mininmum is zero:
           XYZ_BIN(:,2) - Ymin/BIN_Size + 1, ...
           XYZ_BIN(:,3) - Zmin/BIN_Size + 1];

X_BIN = XYZ_BIN(:,1);
Y_BIN = XYZ_BIN(:,2);
Z_BIN = XYZ_BIN(:,3);

% Calculate number of pixels:
Num_Pixels = (Xrange/BIN_Size + 1)*(Yrange/BIN_Size +
1)*(Zrange/BIN_Size + 1);

ErrorMatrix_Pixel = zeros(Xrange/BIN_Size + 1,Yrange/BIN_Size +
1,Zrange/BIN_Size + 1);

%% Analyze LS Error:
X_ErrorSum_Pixel_LS = ErrorMatrix_Pixel;
Y_ErrorSum_Pixel_LS = ErrorMatrix_Pixel;
Z_ErrorSum_Pixel_LS = ErrorMatrix_Pixel;

X_MSE_Error_Pixel_LS = ErrorMatrix_Pixel;
Y_MSE_Error_Pixel_LS = ErrorMatrix_Pixel;
Z_MSE_Error_Pixel_LS = ErrorMatrix_Pixel;

MSE_Error_Pixel_LS = ErrorMatrix_Pixel;
NumPts_Pixel_LS = ErrorMatrix_Pixel;

for i = 1:Num_Norm_Pts

    X_ErrorSum_Pixel_LS(X_BIN(i),Y_BIN(i)) =
X_ErrorSum_Pixel_LS(X_BIN(i),Y_BIN(i))...
                                                          +
Error_LS(i,1);
```

```
    Y_ErrorSum_Pixel_LS(X_BIN(i),Y_BIN(i)) =
Y_ErrorSum_Pixel_LS(X_BIN(i),Y_BIN(i))...
                                                        +
Error_LS(i,2);
    Z_ErrorSum_Pixel_LS(X_BIN(i),Y_BIN(i)) =
Z_ErrorSum_Pixel_LS(X_BIN(i),Y_BIN(i))...
                                                        +
Error_LS(i,3);

    X_MSE_Error_Pixel_LS(X_BIN(i),Y_BIN(i)) =
X_MSE_Error_Pixel_LS(X_BIN(i),Y_BIN(i))...
                                                        +
Error_LS(i,1)^2;
    Y_MSE_Error_Pixel_LS(X_BIN(i),Y_BIN(i)) =
Y_MSE_Error_Pixel_LS(X_BIN(i),Y_BIN(i))...
                                                        +
Error_LS(i,2)^2;
    Z_MSE_Error_Pixel_LS(X_BIN(i),Y_BIN(i)) =
Z_MSE_Error_Pixel_LS(X_BIN(i),Y_BIN(i))...
                                                        +
Error_LS(i,3)^2;

    MSE_Error_Pixel_LS(X_BIN(i),Y_BIN(i)) =
MSE_Error_Pixel_LS(X_BIN(i),Y_BIN(i))...
                                                        +
sum(Error_LS(i,:).^2);

    NumPts_Pixel_LS(X_BIN(i),Y_BIN(i)) =
NumPts_Pixel_LS(X_BIN(i),Y_BIN(i)) + 1;

end

X_ErrorAvg_Pixel_LS = X_ErrorSum_Pixel_LS./NumPts_Pixel_LS;
Y_ErrorAvg_Pixel_LS = Y_ErrorSum_Pixel_LS./NumPts_Pixel_LS;
Z_ErrorAvg_Pixel_LS = Z_ErrorSum_Pixel_LS./NumPts_Pixel_LS;

X_MSE_Error_Pixel_LS = X_MSE_Error_Pixel_LS./NumPts_Pixel_LS;
Y_MSE_Error_Pixel_LS = Y_MSE_Error_Pixel_LS./NumPts_Pixel_LS;
Z_MSE_Error_Pixel_LS = Z_MSE_Error_Pixel_LS./NumPts_Pixel_LS;

MSE_Error_Pixel_LS = MSE_Error_Pixel_LS./NumPts_Pixel_LS;
MSE_Error_Total_LS = sum(sum(MSE_Error_Pixel_LS))/Num_Pixels;

%% Analyze NLS Error

X_ErrorSum_Pixel_NLS = ErrorMatrix_Pixel;
Y_ErrorSum_Pixel_NLS = ErrorMatrix_Pixel;
Z_ErrorSum_Pixel_NLS = ErrorMatrix_Pixel;

X_MSE_Error_Pixel_NLS = ErrorMatrix_Pixel;
Y_MSE_Error_Pixel_NLS = ErrorMatrix_Pixel;
Z_MSE_Error_Pixel_NLS = ErrorMatrix_Pixel;
```

```
MSE_Error_Pixel_NLS = ErrorMatrix_Pixel;
NumPts_Pixel_NLS = ErrorMatrix_Pixel;

for i = 1:Num_Norm_Pts

    X_ErrorSum_Pixel_NLS(X_BIN(i),Y_BIN(i)) =
X_ErrorSum_Pixel_NLS(X_BIN(i),Y_BIN(i))...
                                                    +
Error_NLS(i,1);
    Y_ErrorSum_Pixel_NLS(X_BIN(i),Y_BIN(i)) =
Y_ErrorSum_Pixel_NLS(X_BIN(i),Y_BIN(i))...
                                                    +
Error_NLS(i,2);
    Z_ErrorSum_Pixel_NLS(X_BIN(i),Y_BIN(i)) =
Z_ErrorSum_Pixel_NLS(X_BIN(i),Y_BIN(i))...
                                                    +
Error_NLS(i,3);

    X_MSE_Error_Pixel_NLS(X_BIN(i),Y_BIN(i)) =
X_MSE_Error_Pixel_NLS(X_BIN(i),Y_BIN(i))...
                                                    +
Error_NLS(i,1)^2;
    Y_MSE_Error_Pixel_NLS(X_BIN(i),Y_BIN(i)) =
Y_MSE_Error_Pixel_NLS(X_BIN(i),Y_BIN(i))...
                                                    +
Error_NLS(i,2)^2;
    Z_MSE_Error_Pixel_NLS(X_BIN(i),Y_BIN(i)) =
Z_MSE_Error_Pixel_NLS(X_BIN(i),Y_BIN(i))...
                                                    +
Error_NLS(i,3)^2;

    MSE_Error_Pixel_NLS(X_BIN(i),Y_BIN(i)) =
MSE_Error_Pixel_NLS(X_BIN(i),Y_BIN(i))...
                                                    +
sum(Error_NLS(i,:).^2);

    NumPts_Pixel_NLS(X_BIN(i),Y_BIN(i)) =
NumPts_Pixel_NLS(X_BIN(i),Y_BIN(i)) + 1;

end

X_ErrorAvg_Pixel_NLS = X_ErrorSum_Pixel_NLS./NumPts_Pixel_NLS;
Y_ErrorAvg_Pixel_NLS = Y_ErrorSum_Pixel_NLS./NumPts_Pixel_NLS;
Z_ErrorAvg_Pixel_NLS = Z_ErrorSum_Pixel_NLS./NumPts_Pixel_NLS;

X_MSE_Error_Pixel_NLS = X_MSE_Error_Pixel_NLS./NumPts_Pixel_NLS;
Y_MSE_Error_Pixel_NLS = Y_MSE_Error_Pixel_NLS./NumPts_Pixel_NLS;
Z_MSE_Error_Pixel_NLS = Z_MSE_Error_Pixel_NLS./NumPts_Pixel_NLS;

MSE_Error_Pixel_NLS = MSE_Error_Pixel_NLS./NumPts_Pixel_NLS;
MSE_Error_Total_NLS = sum(sum(MSE_Error_Pixel_NLS))/Num_Pixels;
```

```
%% Analyze CMD Error

X_ErrorSum_Pixel_CMD = ErrorMatrix_Pixel;
Y_ErrorSum_Pixel_CMD = ErrorMatrix_Pixel;
Z_ErrorSum_Pixel_CMD = ErrorMatrix_Pixel;

X_MSE_Error_Pixel_CMD = ErrorMatrix_Pixel;
Y_MSE_Error_Pixel_CMD = ErrorMatrix_Pixel;
Z_MSE_Error_Pixel_CMD = ErrorMatrix_Pixel;

MSE_Error_Pixel_CMD = ErrorMatrix_Pixel;
NumPts_Pixel_CMD = ErrorMatrix_Pixel;

for i = 1:Num_Norm_Pts

    X_ErrorSum_Pixel_CMD(X_BIN(i),Y_BIN(i)) =
X_ErrorSum_Pixel_CMD(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CMD(i,1);
    Y_ErrorSum_Pixel_CMD(X_BIN(i),Y_BIN(i)) =
Y_ErrorSum_Pixel_CMD(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CMD(i,2);
    Z_ErrorSum_Pixel_CMD(X_BIN(i),Y_BIN(i)) =
Z_ErrorSum_Pixel_CMD(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CMD(i,3);

    X_MSE_Error_Pixel_CMD(X_BIN(i),Y_BIN(i)) =
X_MSE_Error_Pixel_CMD(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CMD(i,1)^2;
    Y_MSE_Error_Pixel_CMD(X_BIN(i),Y_BIN(i)) =
Y_MSE_Error_Pixel_CMD(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CMD(i,2)^2;
    Z_MSE_Error_Pixel_CMD(X_BIN(i),Y_BIN(i)) =
Z_MSE_Error_Pixel_CMD(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CMD(i,3)^2;

    MSE_Error_Pixel_CMD(X_BIN(i),Y_BIN(i)) =
MSE_Error_Pixel_CMD(X_BIN(i),Y_BIN(i))...
                                                    +
sum(Error_CMD(i,:).^2);

    NumPts_Pixel_CMD(X_BIN(i),Y_BIN(i)) =
NumPts_Pixel_CMD(X_BIN(i),Y_BIN(i)) + 1;

end
```

```
X_ErrorAvg_Pixel_CMD = X_ErrorSum_Pixel_CMD./NumPts_Pixel_CMD;
Y_ErrorAvg_Pixel_CMD = Y_ErrorSum_Pixel_CMD./NumPts_Pixel_CMD;
Z_ErrorAvg_Pixel_CMD = Z_ErrorSum_Pixel_CMD./NumPts_Pixel_CMD;

X_MSE_Error_Pixel_CMD = X_MSE_Error_Pixel_CMD./NumPts_Pixel_CMD;
Y_MSE_Error_Pixel_CMD = Y_MSE_Error_Pixel_CMD./NumPts_Pixel_CMD;
Z_MSE_Error_Pixel_CMD = Z_MSE_Error_Pixel_CMD./NumPts_Pixel_CMD;

MSE_Error_Pixel_CMD = MSE_Error_Pixel_CMD./NumPts_Pixel_CMD;
MSE_Error_Total_CMD = sum(sum(MSE_Error_Pixel_CMD))/Num_Pixels;

%% Analyze CFP Error

X_ErrorSum_Pixel_CFP = ErrorMatrix_Pixel;
Y_ErrorSum_Pixel_CFP = ErrorMatrix_Pixel;
Z_ErrorSum_Pixel_CFP = ErrorMatrix_Pixel;

X_MSE_Error_Pixel_CFP = ErrorMatrix_Pixel;
Y_MSE_Error_Pixel_CFP = ErrorMatrix_Pixel;
Z_MSE_Error_Pixel_CFP = ErrorMatrix_Pixel;

MSE_Error_Pixel_CFP = ErrorMatrix_Pixel;
NumPts_Pixel_CFP = ErrorMatrix_Pixel;

for i = 1:Num_Norm_Pts

    X_ErrorSum_Pixel_CFP(X_BIN(i),Y_BIN(i)) =
X_ErrorSum_Pixel_CFP(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CFP(i,1);
    Y_ErrorSum_Pixel_CFP(X_BIN(i),Y_BIN(i)) =
Y_ErrorSum_Pixel_CFP(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CFP(i,2);
    Z_ErrorSum_Pixel_CFP(X_BIN(i),Y_BIN(i)) =
Z_ErrorSum_Pixel_CFP(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CFP(i,3);

    X_MSE_Error_Pixel_CFP(X_BIN(i),Y_BIN(i)) =
X_MSE_Error_Pixel_CFP(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CFP(i,1)^2;
    Y_MSE_Error_Pixel_CFP(X_BIN(i),Y_BIN(i)) =
Y_MSE_Error_Pixel_CFP(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CFP(i,2)^2;
    Z_MSE_Error_Pixel_CFP(X_BIN(i),Y_BIN(i)) =
Z_MSE_Error_Pixel_CFP(X_BIN(i),Y_BIN(i))...
                                                    +
Error_CFP(i,3)^2;
```

```
    MSE_Error_Pixel_CFP(X_BIN(i),Y_BIN(i)) =
MSE_Error_Pixel_CFP(X_BIN(i),Y_BIN(i))...
                                                        +
sum(Error_CFP(i,:).^2);

    NumPts_Pixel_CFP(X_BIN(i),Y_BIN(i)) =
NumPts_Pixel_CFP(X_BIN(i),Y_BIN(i)) + 1;

end

X_ErrorAvg_Pixel_CFP = X_ErrorSum_Pixel_CFP./NumPts_Pixel_CFP;
Y_ErrorAvg_Pixel_CFP = Y_ErrorSum_Pixel_CFP./NumPts_Pixel_CFP;
Z_ErrorAvg_Pixel_CFP = Z_ErrorSum_Pixel_CFP./NumPts_Pixel_CFP;

X_MSE_Error_Pixel_CFP = X_MSE_Error_Pixel_CFP./NumPts_Pixel_CFP;
Y_MSE_Error_Pixel_CFP = Y_MSE_Error_Pixel_CFP./NumPts_Pixel_CFP;
Z_MSE_Error_Pixel_CFP = Z_MSE_Error_Pixel_CFP./NumPts_Pixel_CFP;

MSE_Error_Pixel_CFP = MSE_Error_Pixel_CFP./NumPts_Pixel_CFP;
MSE_Error_Total_CFP = sum(sum(MSE_Error_Pixel_CFP))/Num_Pixels;

%% Sum MSE for each method:

MSE_Error_Total_All = [MSE_Error_Total_LS; ...
                       MSE_Error_Total_NLS; ...
                       MSE_Error_Total_CMD; ...
                       MSE_Error_Total_CFP];

MSE_Error_Normalized_All =
MSE_Error_Total_All/min(MSE_Error_Total_All);

figure(2); clf; title('Comparison of Normalized MSE for All
Methods');
bar(MSE_Error_Normalized_All);

%% Define Z value (height) at which to observe XY plan & XY
meshgrid:

Z_Index = (Z_Plane_Value-Zmin)/BIN_Size + 1;
[X_Mesh,Y_Mesh] = meshgrid(Xmin:BIN_Size:Xmax,Ymin:BIN_Size:Ymax);

% Find max MSE for scaling plots:
MSE_Max_LS = max(max(MSE_Error_Pixel_LS(:,:,Z_Index)));
MSE_Max_NLS = max(max(MSE_Error_Pixel_NLS(:,:,Z_Index)));
MSE_Max_CMD = max(max(MSE_Error_Pixel_CMD(:,:,Z_Index)));
MSE_Max_CFP = max(max(MSE_Error_Pixel_CFP(:,:,Z_Index)));

MSE_Max = max([MSE_Max_LS,MSE_Max_NLS,MSE_Max_CMD,MSE_Max_CFP]);

%% LS Plots
```

```
Z_Str = int2str(Z_Plane_Value);

figure(10); clf; hold on; grid on; xlabel('X Coordinate (mm)');
ylabel('Y Coordinate (mm)'); zlabel('MSE (mm^2)');
title(strcat('Mean Squared Error - LS Method, Z= ',Z_Str,' mm'));
%surf(X_Mesh,Y_Mesh,X_ErrorAvg_Pixel_LS(:,:,Z_Index));
surf(X_Mesh,Y_Mesh,MSE_Error_Pixel_LS(:,:,Z_Index));
axis([Xmin Xmax Ymin Ymax 0 MSE_Max]);
plot3(RCM(:,1),RCM(:,2),RCM(:,3),'.');
LS_Handle = get(gcf,'CurrentAxes');
set(gca,'ZScale','log');

% figure(11); clf; hold on;
% contour(X_Mesh,Y_Mesh,real(MSE_Error_Pixel_LS(:,:,Z_Index)));
% [U_MSE_LS, V_MSE_LS] =
gradient(real(MSE_Error_Pixel_LS(:,:,Z_Index)),BIN_Size);
% quiver(X_Mesh,Y_Mesh,U_MSE_LS,V_MSE_LS);

figure(12); clf; hold on; grid on;
title(strcat('XY MSE and XY Error Biasing - LS Method, Z= ',Z_Str,'
mm'));
contour(X_Mesh,Y_Mesh,real(X_MSE_Error_Pixel_LS(:,:,Z_Index)+Y_MSE_E
rror_Pixel_LS(:,:,Z_Index)));
quiver(X_Mesh,Y_Mesh,X_ErrorAvg_Pixel_LS,Y_ErrorAvg_Pixel_LS);
plot(RCM(:,1),RCM(:,2),'Color','black','Marker','.','LineStyle','non
e','MarkerSize',15);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
legend('XY MSE','Avg XY Error Vectors','Receiver Locations');

%% NLS Plots

figure(20); clf; hold on; grid on; xlabel('X Coordinate (mm)');
ylabel('Y Coordinate (mm)'); zlabel('MSE (mm^2)');
title(strcat('Mean Squared Error - NLS Method, Z=',Z_Str,' mm'));
%surf(X_Mesh,Y_Mesh,X_ErrorAvg_Pixel_NLS(:,:,Z_Index));
surf(X_Mesh,Y_Mesh,MSE_Error_Pixel_NLS(:,:,Z_Index));
axis([Xmin Xmax Ymin Ymax 0 MSE_Max]);
plot3(RCM(:,1),RCM(:,2),RCM(:,3),'.');
NLS_Handle = get(gcf,'CurrentAxes');
set(gca,'ZScale','log');

% figure(21); clf; hold on;
% contour(X_Mesh,Y_Mesh,real(MSE_Error_Pixel_NLS(:,:,Z_Index)));
% [U_MSE_NLS, V_MSE_NLS] =
gradient(real(MSE_Error_Pixel_NLS(:,:,Z_Index)),BIN_Size);
% quiver(X_Mesh,Y_Mesh,U_MSE_NLS,V_MSE_NLS);

figure(22); clf; hold on; grid on;
title(strcat('XY MSE and XY Error Biasing - NLS Method, Z=',Z_Str,'
mm'));
contour(X_Mesh,Y_Mesh,real(X_MSE_Error_Pixel_NLS(:,:,Z_Index)+Y_MSE_
Error_Pixel_NLS(:,:,Z_Index)));
```

```
quiver(X_Mesh,Y_Mesh,X_ErrorAvg_Pixel_NLS,Y_ErrorAvg_Pixel_NLS);
plot(RCM(:,1),RCM(:,2),'Color','black','Marker','.','LineStyle','non
e','MarkerSize',15);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
legend('XY MSE','Avg XY Error Vectors','Receiver Locations');


%% CFP Plots

figure(30); clf; hold on; grid on; xlabel('X Coordinate (mm)');
ylabel('Y Coordinate (mm)'); zlabel('MSE (mm^2)');
title(strcat('Mean Squared Error - CFP Method, Z=',Z_Str,' mm'));
%surf(X_Mesh,Y_Mesh,X_ErrorAvg_Pixel_CFP(:,:,Z_Index));
surf(X_Mesh,Y_Mesh,real(MSE_Error_Pixel_CFP(:,:,Z_Index)));
axis([Xmin Xmax Ymin Ymax 0 MSE_Max]);
plot3(RCM_3(:,1),RCM_3(:,2),RCM_3(:,3),'.');
CFP_Handle = get(gcf,'CurrentAxes');
set(gca,'ZScale','log');

% figure(31); clf; hold on;
% contour(X_Mesh,Y_Mesh,real(MSE_Error_Pixel_CFP(:,:,Z_Index)));
% [U_MSE_CFP, V_MSE_CFP] =
gradient(real(MSE_Error_Pixel_CFP(:,:,Z_Index)),BIN_Size);
% quiver(X_Mesh,Y_Mesh,U_MSE_CFP,V_MSE_CFP);

figure(32); clf; hold on; grid on;
title(strcat('XY MSE and XY Error Biasing - CFP Method, Z=',Z_Str,'
mm'));
contour(X_Mesh,Y_Mesh,real(X_MSE_Error_Pixel_CFP(:,:,Z_Index)+Y_MSE_
Error_Pixel_CFP(:,:,Z_Index)));
quiver(X_Mesh,Y_Mesh,X_ErrorAvg_Pixel_CFP,Y_ErrorAvg_Pixel_CFP);
plot(RCM_3(:,1),RCM_3(:,2),'Color','black','Marker','.','LineStyle',
'none','MarkerSize',15);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
legend('XY MSE','Avg XY Error Vectors','Receiver Locations');

%% CMD Plots

figure(40); clf; hold on; grid on; xlabel('X Coordinate (mm)');
ylabel('Y Coordinate (mm)'); zlabel('MSE (mm^2)');
title(strcat('Mean Squared Error - CMD Method, Z=',Z_Str,' mm'));
%surf(X_Mesh,Y_Mesh,X_ErrorAvg_Pixel_CMD(:,:,Z_Index));
surf(X_Mesh,Y_Mesh,real(MSE_Error_Pixel_CMD(:,:,Z_Index)));
axis([Xmin Xmax Ymin Ymax 0 MSE_Max]);
plot3(RCM_3(:,1),RCM_3(:,2),RCM_3(:,3),'.');
CMD_Handle = get(gcf,'CurrentAxes');
set(gca,'ZScale','log');

% figure(41); clf; hold on;
% contour(X_Mesh,Y_Mesh,real(MSE_Error_Pixel_CMD(:,:,Z_Index)));
% [U_MSE_CMD, V_MSE_CMD] =
gradient(real(MSE_Error_Pixel_CMD(:,:,Z_Index)),BIN_Size);
```

```
% quiver(X_Mesh,Y_Mesh,U_MSE_CMD,V_MSE_CMD);

figure(42); clf; hold on; grid on;
title(strcat('XY MSE and XY Error Biasing - CMD Method, Z=',Z_Str,'
mm'));
contour(X_Mesh,Y_Mesh,real(X_MSE_Error_Pixel_CMD(:,:,Z_Index)+Y_MSE_
Error_Pixel_CMD(:,:,Z_Index)));
quiver(X_Mesh,Y_Mesh,X_ErrorAvg_Pixel_CMD,Y_ErrorAvg_Pixel_CMD);
plot(RCM_3(:,1),RCM_3(:,2),'Color','black','Marker','.','LineStyle',
'none','MarkerSize',15);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
legend('XY MSE','Avg XY Error Vectors','Receiver Locations');

%% Z Error Plots

% figure(50); clf; surf(X_Mesh,Y_Mesh,Z_ErrorAvg_Pixel_LS);
% axis([-5000 5000 -5000 5000 -800 400]);
% figure(51); clf; surf(X_Mesh,Y_Mesh,Z_ErrorAvg_Pixel_NLS);
% axis([-5000 5000 -5000 5000 -800 400]);
% figure(52); clf; surf(X_Mesh,Y_Mesh,real(Z_ErrorAvg_Pixel_CFP));
% axis([-5000 5000 -5000 5000 -800 400]);
% figure(53); clf; surf(X_Mesh,Y_Mesh,real(Z_ErrorAvg_Pixel_CMD));
% axis([-5000 5000 -5000 5000 -800 400]);

figure(54); clf; title(strcat('Average Z Error, Z=,',Z_Str,' mm'));
subplot(221); surf(X_Mesh,Y_Mesh,Z_ErrorAvg_Pixel_LS);
axis([Xmin Xmax Ymin Ymax -800 400]);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
zlabel('Error (mm)');
title('Avg Z-Error - LS Method');
subplot(222); surf(X_Mesh,Y_Mesh,Z_ErrorAvg_Pixel_NLS);
axis([Xmin Xmax Ymin Ymax -800 400]);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
zlabel('Error (mm)');
title('Avg Z-Error - NLS Method');
subplot(223); surf(X_Mesh,Y_Mesh,real(Z_ErrorAvg_Pixel_CFP));
axis([Xmin Xmax Ymin Ymax -800 400]);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
zlabel('Error (mm)');
title('Avg Z-Error - CFP Method');
subplot(224); surf(X_Mesh,Y_Mesh,real(Z_ErrorAvg_Pixel_CMD));
axis([Xmin Xmax Ymin Ymax -800 400]);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
zlabel('Error (mm)');
title('Avg Z-Error - CMD Method');


%% Other Plots

% figure(60); clf; surf(X_Mesh,Y_Mesh,Z_ErrorAvg_Pixel_LS);
% Z_err_HandleLS = get(gcf,'CurrentAxes');
% set(gca,'ZScale','log');
```

```
% axis([Xmin Xmax Ymin Ymax min(min(Z_ErrorAvg_Pixel_LS))
max(max(Z_ErrorAvg_Pixel_LS))]);
%
% figure(61); clf; surf(X_Mesh,Y_Mesh,Z_ErrorAvg_Pixel_NLS);
% Z_err_HandleNLS = get(gcf,'CurrentAxes');
% set(gca,'ZScale','log');
% axis([Xmin Xmax Ymin Ymax min(min(Z_ErrorAvg_Pixel_NLS))
max(max(Z_ErrorAvg_Pixel_NLS))]);
%
% figure(62); clf; surf(X_Mesh,Y_Mesh,Z_ErrorAvg_Pixel_CFP);
% Z_err_HandleCFP = get(gcf,'CurrentAxes');
% set(gca,'ZScale','log');
% axis([Xmin Xmax Ymin Ymax min(min(Z_ErrorAvg_Pixel_CFP))
max(max(Z_ErrorAvg_Pixel_CFP))]);
%
% figure(63); clf; surf(X_Mesh,Y_Mesh,Z_ErrorAvg_Pixel_CMD);
% Z_err_HandleCMD = get(gcf,'CurrentAxes');
% set(gca,'ZScale','log');
% axis([Xmin Xmax Ymin Ymax min(min(Z_ErrorAvg_Pixel_CMD))
max(max(Z_ErrorAvg_Pixel_CMD))]);

%% Metrics

% LS Metrics
Metric_MSE_Total_LS = MSE_Error_Total_LS;
Metric_X_MSE_LS = sum(sum(X_MSE_Error_Pixel_LS))/Num_Pixels;
Metric_Y_MSE_LS = sum(sum(Y_MSE_Error_Pixel_LS))/Num_Pixels;
Metric_Z_MSE_LS = sum(sum(Z_MSE_Error_Pixel_LS))/Num_Pixels;

Metric_X_ErrorAvg_LS = sum(sum(X_ErrorAvg_Pixel_LS))/Num_Pixels;
Metric_Y_ErrorAvg_LS = sum(sum(Y_ErrorAvg_Pixel_LS))/Num_Pixels;
Metric_Z_ErrorAvg_LS = sum(sum(Z_ErrorAvg_Pixel_LS))/Num_Pixels;

% NLS Metrics
Metric_MSE_Total_NLS = MSE_Error_Total_NLS;
Metric_X_MSE_NLS = sum(sum(X_MSE_Error_Pixel_NLS))/Num_Pixels;
Metric_Y_MSE_NLS = sum(sum(Y_MSE_Error_Pixel_NLS))/Num_Pixels;
Metric_Z_MSE_NLS = sum(sum(Z_MSE_Error_Pixel_NLS))/Num_Pixels;

Metric_X_ErrorAvg_NLS = sum(sum(X_ErrorAvg_Pixel_NLS))/Num_Pixels;
Metric_Y_ErrorAvg_NLS = sum(sum(Y_ErrorAvg_Pixel_NLS))/Num_Pixels;
Metric_Z_ErrorAvg_NLS = sum(sum(Z_ErrorAvg_Pixel_NLS))/Num_Pixels;

% CFP Metrics
Metric_MSE_Total_CFP = MSE_Error_Total_CFP;
Metric_X_MSE_CFP = sum(sum(X_MSE_Error_Pixel_CFP))/Num_Pixels;
Metric_Y_MSE_CFP = sum(sum(Y_MSE_Error_Pixel_CFP))/Num_Pixels;
Metric_Z_MSE_CFP = sum(sum(Z_MSE_Error_Pixel_CFP))/Num_Pixels;

Metric_X_ErrorAvg_CFP = sum(sum(X_ErrorAvg_Pixel_CFP))/Num_Pixels;
Metric_Y_ErrorAvg_CFP = sum(sum(Y_ErrorAvg_Pixel_CFP))/Num_Pixels;
Metric_Z_ErrorAvg_CFP = sum(sum(Z_ErrorAvg_Pixel_CFP))/Num_Pixels;
```

```
% CMD Metrics
Metric_MSE_Total_CMD = MSE_Error_Total_CMD;
Metric_X_MSE_CMD = sum(sum(X_MSE_Error_Pixel_CMD))/Num_Pixels;
Metric_Y_MSE_CMD = sum(sum(Y_MSE_Error_Pixel_CMD))/Num_Pixels;
Metric_Z_MSE_CMD = sum(sum(Z_MSE_Error_Pixel_CMD))/Num_Pixels;

Metric_X_ErrorAvg_CMD = sum(sum(X_ErrorAvg_Pixel_CMD))/Num_Pixels;
Metric_Y_ErrorAvg_CMD = sum(sum(Y_ErrorAvg_Pixel_CMD))/Num_Pixels;
Metric_Z_ErrorAvg_CMD = sum(sum(Z_ErrorAvg_Pixel_CMD))/Num_Pixels;

% Summary of Metrics
Metric_Summary = real([Metric_X_ErrorAvg_LS  Metric_X_ErrorAvg_NLS
Metric_X_ErrorAvg_CFP   Metric_X_ErrorAvg_CMD; ...
                       Metric_Y_ErrorAvg_LS  Metric_Y_ErrorAvg_NLS
Metric_Y_ErrorAvg_CFP   Metric_Y_ErrorAvg_CMD; ...
                       Metric_Z_ErrorAvg_LS  Metric_Z_ErrorAvg_NLS
Metric_Z_ErrorAvg_CFP   Metric_Z_ErrorAvg_CMD; ...
                       Metric_X_MSE_LS       Metric_X_MSE_NLS
Metric_X_MSE_CFP        Metric_X_MSE_CMD;       ...
                       Metric_Y_MSE_LS       Metric_Y_MSE_NLS
Metric_Y_MSE_CFP        Metric_Y_MSE_CMD;       ...
                       Metric_Z_MSE_LS       Metric_Z_MSE_NLS
Metric_Z_MSE_CFP        Metric_Z_MSE_CMD;       ...
                       Metric_MSE_Total_LS   Metric_MSE_Total_NLS
Metric_MSE_Total_CFP    Metric_MSE_Total_CMD]);


%% Format MSE Figure:
figure(1); hold on; grid on; xlabel('X Coordinate (mm)'); ylabel('Y
Coordinate (mm)'); zlabel('MSE (mm^2)');

%% Format Bias Figure:
figure(1);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');

%% Format Z Avg Figure:
figure(1); hold on;
h = get(gca,'Children');

%%
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
zlabel('Error (mm)');
subplot(222);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
zlabel('Error (mm)');
subplot(223);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
zlabel('Error (mm)');
subplot(224);
xlabel('X Coordinate (mm)');ylabel('Y Coordinate (mm)');
zlabel('Error (mm)');
```

APPENDIX C – Data Post-Processing and Kalman Filter Matlab Programs

```
%% System ID Preprocess

%% Set System and Analysis Parameters

clc
clear

RootPath = 'H:\Masters Thesis\';

addpath(strcat(RootPath,'Matlab\'));
addpath(strcat(RootPath,'Matlab\Test Data Processing\'));

% System Parameters:
NOR     =   6;
Rx40    =   [339.6750,  540.2600,   22];
Rx41    =   [987.3615,  540.2600,   22];
Rx42    =   [647.70,    0,          22];
Rx43    =   [679.35,    1080.52,    0];
Rx44    =   [0,         0,          0];
Rx45    =   [1295.4,    0,          0];
RCM     =   [Rx40;Rx41;Rx42;Rx43;Rx44;Rx45];

%% Import Data

FlightData = importdata(strcat(RootPath,'Test Data\Sys ID
Testing\West Hills_05_07_2013\05_07_2013_SysID_Ch3_Throttle.txt'));
Input_Channel = 3;

time = FlightData.data(:,1);
T20_R40_Data = FlightData.data(:,8);
T20_R41_Data = FlightData.data(:,9);
T20_R42_Data = FlightData.data(:,10);
T20_R43_Data = FlightData.data(:,11);
T20_R44_Data = FlightData.data(:,12);
T20_R45_Data = FlightData.data(:,13);

T21_R40_Data = FlightData.data(:,14);
T21_R41_Data = FlightData.data(:,15);
T21_R42_Data = FlightData.data(:,16);
T21_R43_Data = FlightData.data(:,17);
T21_R44_Data = FlightData.data(:,18);
T21_R45_Data = FlightData.data(:,19);

Ch1_Input = FlightData.data(:,33);
Ch2_Input = FlightData.data(:,34);
Ch3_Input = FlightData.data(:,35);
Ch4_Input = FlightData.data(:,35);
```

```
All_Data = [T20_R40_Data, T20_R41_Data, T20_R42_Data, T20_R43_Data,
T20_R44_Data, T20_R45_Data, ...
            T21_R40_Data, T21_R41_Data, T21_R42_Data, T21_R43_Data,
T21_R44_Data, T21_R45_Data, ...
            Ch1_Input, Ch2_Input, Ch3_Input, Ch4_Input];

%% Visualize data:

figure(1); clf;
subplot(321); plot(time,T20_R40_Data); title('T20-R40 Raw Data');
subplot(322); plot(time,T20_R41_Data); title('T20-R41 Raw Data');
subplot(323); plot(time,T20_R42_Data); title('T20-R42 Raw Data');
subplot(324); plot(time,T20_R43_Data); title('T20-R43 Raw Data');
subplot(325); plot(time,T20_R44_Data); title('T20-R44 Raw Data');
subplot(326); plot(time,T20_R45_Data); title('T20-R45 Raw Data');

figure(2); clf;
subplot(321); plot(time,T21_R40_Data); title('T21-R40 Raw Data');
subplot(322); plot(time,T21_R41_Data); title('T21-R41 Raw Data');
subplot(323); plot(time,T21_R42_Data); title('T21-R42 Raw Data');
subplot(324); plot(time,T21_R43_Data); title('T21-R43 Raw Data');
subplot(325); plot(time,T21_R44_Data); title('T21-R44 Raw Data');
subplot(326); plot(time,T21_R44_Data); title('T21-R45 Raw Data');

figure(3); clf;
subplot(321); plot(time,T20_R40_Data - T21_R40_Data);
title('Difference T20R40 - T21R40');
subplot(322); plot(time,T20_R41_Data - T21_R41_Data);
title('Difference T20R41 - T21R41');
subplot(323); plot(time,T20_R42_Data - T21_R42_Data);
title('Difference T20R42 - T21R42');
subplot(324); plot(time,T20_R43_Data - T21_R43_Data);
title('Difference T20R43 - T21R43');
subplot(325); plot(time,T20_R44_Data - T21_R44_Data);
title('Difference T20R44 - T21R44');
subplot(326); plot(time,T20_R45_Data - T21_R45_Data);
title('Difference T20R45 - T21R45');

% Histogram of change in consecutive range values for all Tx/Rx
combos:
XValues = [-10000:1000:10000];

figure(4); clf;
subplot(321); hist(diff(T20_R40_Data),XValues); title('Histogram of
Change in Range Values: T20R40');
xlabel('Change in Range Measurement (mm)');
subplot(322); hist(diff(T20_R41_Data),XValues); title('Histogram of
Change in Range Values: T20R41');
xlabel('Change in Range Measurement (mm)');
subplot(323); hist(diff(T20_R42_Data),XValues); title('Histogram of
Change in Range Values: T20R42');
xlabel('Change in Range Measurement (mm)');
```

```
subplot(324); hist(diff(T20_R43_Data),XValues); title('Histogram of
Change in Range Values: T20R43');
xlabel('Change in Range Measurement (mm)');
subplot(325); hist(diff(T20_R44_Data),XValues); title('Histogram of
Change in Range Values: T20R44');
xlabel('Change in Range Measurement (mm)');
subplot(326); hist(diff(T20_R45_Data),XValues); title('Histogram of
Change in Range Values: T20R45');
xlabel('Change in Range Measurement (mm)');

figure(5); clf;
subplot(321); hist(diff(T21_R40_Data),XValues); title('Histogram of
Change in Range Values: T21R40');
xlabel('Change in Range Measurement (mm)');
subplot(322); hist(diff(T21_R41_Data),XValues); title('Histogram of
Change in Range Values: T21R41');
xlabel('Change in Range Measurement (mm)');
subplot(323); hist(diff(T21_R42_Data),XValues); title('Histogram of
Change in Range Values: T21R42');
xlabel('Change in Range Measurement (mm)');
subplot(324); hist(diff(T21_R43_Data),XValues); title('Histogram of
Change in Range Values: T21R43');
xlabel('Change in Range Measurement (mm)');
subplot(325); hist(diff(T21_R44_Data),XValues); title('Histogram of
Change in Range Values: T21R44');
xlabel('Change in Range Measurement (mm)');
subplot(326); hist(diff(T21_R45_Data),XValues); title('Histogram of
Change in Range Values: T21R45');
xlabel('Change in Range Measurement (mm)');

% Histogram of 'difference' values (T20 - T21):
Num_Bins = 20;

figure(6); clf;
subplot(321); hist(T20_R40_Data - T21_R40_Data,Num_Bins);
title('Histogram of Differences: T20R40 - T21R40');
subplot(322); hist(T20_R41_Data - T21_R41_Data,Num_Bins);
title('Histogram of Differences: T20R41 - T21R41');
subplot(323); hist(T20_R42_Data - T21_R42_Data,Num_Bins);
title('Histogram of Differences: T20R42 - T21R42');
subplot(324); hist(T20_R43_Data - T21_R43_Data,Num_Bins);
title('Histogram of Differences: T20R43 - T21R43');
subplot(325); hist(T20_R44_Data - T21_R44_Data,Num_Bins);
title('Histogram of Differences: T20R44 - T21R44');
subplot(326); hist(T20_R45_Data - T21_R45_Data,Num_Bins);
title('Histogram of Differences: T20R45 - T21R45');

%% Filter Raw Range Data:

addpath(strcat(RootPath,'Matlab\Test Data Processing\Filters'));

% Apply filter 1 to all range data:
```

```
Delta_R = 1000; % Max speed of approx. 10 m/s (22.3 mph)
Num_Lead_Pts = 40;

T20R40_New = FilterRawData1(T20_R40_Data,Delta_R,Num_Lead_Pts);
T20R41_New = FilterRawData1(T20_R41_Data,Delta_R,Num_Lead_Pts);
T20R42_New = FilterRawData1(T20_R42_Data,Delta_R,Num_Lead_Pts);
T20R43_New = FilterRawData1(T20_R43_Data,Delta_R,Num_Lead_Pts);
T20R44_New = FilterRawData1(T20_R44_Data,Delta_R,Num_Lead_Pts);
T20R45_New = FilterRawData1(T20_R45_Data,Delta_R,Num_Lead_Pts);

T21R40_New = FilterRawData1(T21_R40_Data,Delta_R,Num_Lead_Pts);
T21R41_New = FilterRawData1(T21_R41_Data,Delta_R,Num_Lead_Pts);
T21R42_New = FilterRawData1(T21_R42_Data,Delta_R,Num_Lead_Pts);
T21R43_New = FilterRawData1(T21_R43_Data,Delta_R,Num_Lead_Pts);
T21R44_New = FilterRawData1(T21_R44_Data,Delta_R,Num_Lead_Pts);
T21R45_New = FilterRawData1(T21_R45_Data,Delta_R,Num_Lead_Pts);

% Apply filter 2 to all range data:
Max_Dist_Tx = 500; % Separation between T20 and T21
%Num_Lead_Pts = 10;

[T20R40_New2,T21R40_New2] =
FilterRawData2(T20R40_New,T21R40_New,Max_Dist_Tx,Num_Lead_Pts);
[T20R41_New2,T21R41_New2] =
FilterRawData2(T20R41_New,T21R41_New,Max_Dist_Tx,Num_Lead_Pts);
[T20R42_New2,T21R42_New2] =
FilterRawData2(T20R42_New,T21R42_New,Max_Dist_Tx,Num_Lead_Pts);
[T20R43_New2,T21R43_New2] =
FilterRawData2(T20R43_New,T21R43_New,Max_Dist_Tx,Num_Lead_Pts);
[T20R44_New2,T21R44_New2] =
FilterRawData2(T20R44_New,T21R44_New,Max_Dist_Tx,Num_Lead_Pts);
[T20R45_New2,T21R45_New2] =
FilterRawData2(T20R45_New,T21R45_New,Max_Dist_Tx,Num_Lead_Pts);

% Visualize data:
figure(11); clf;
subplot(321); plot(time,T20R40_New2,'b',time,T21R40_New2,'r');
title('R40 Data After Filtering');
legend('T20R40','T21R40');
subplot(322); plot(time,T20R41_New2,'b',time,T21R41_New2,'r');
title('R41 Data After Filtering');
legend('T20R41','T21R41');
subplot(323); plot(time,T20R42_New2,'b',time,T21R42_New2,'r');
title('R42 Data After Filtering');
legend('T20R42','T21R42');
subplot(324); plot(time,T20R43_New2,'b',time,T21R43_New2,'r');
title('R43 Data After Filtering');
legend('T20R43','T21R43');
subplot(325); plot(time,T20R44_New2,'b',time,T21R44_New2,'r');
title('R44 Data After Filtering');
legend('T20R44','T21R44');
subplot(326); plot(time,T20R45_New2,'b',time,T21R45_New2,'r');
title('R45 Data After Filtering');
```

```
legend('T20R45','T21R45');

% Compare filtered data with raw data:
figure(12); clf;
subplot(321); plot(time,T20_R40_Data,time,T20R40_New2,'.');
legend('Raw Data','Filtered Data'); title('T20-R40 Data Before and
After Filtering');
xlabel('Time (s)');
subplot(322); plot(time,T20_R41_Data,time,T20R41_New2,'.');
legend('Raw Data','Filtered Data'); title('T20-R41 Data Before and
After Filtering');
xlabel('Time (s)');
subplot(323); plot(time,T20_R42_Data,time,T20R42_New2,'.');
legend('Raw Data','Filtered Data'); title('T20-R42 Data Before and
After Filtering');
xlabel('Time (s)');
subplot(324); plot(time,T20_R43_Data,time,T20R43_New2,'.');
legend('Raw Data','Filtered Data'); title('T20-R43 Data Before and
After Filtering');
xlabel('Time (s)');
subplot(325); plot(time,T20_R44_Data,time,T20R44_New2,'.');
legend('Raw Data','Filtered Data'); title('T20-R44 Data Before and
After Filtering');
xlabel('Time (s)');
subplot(326); plot(time,T20_R45_Data,time,T20R45_New2,'.');
legend('Raw Data','Filtered Data'); title('T20-R45 Data Before and
After Filtering');
xlabel('Time (s)');

figure(13); clf;
subplot(321); plot(time,T21_R40_Data,time,T21R40_New2,'.');
legend('Raw Data','Filtered Data'); title('T21-R40 Data Before and
After Filtering');
xlabel('Time (s)');
subplot(322); plot(time,T21_R41_Data,time,T21R41_New2,'.');
legend('Raw Data','Filtered Data'); title('T21-R41 Data Before and
After Filtering');
xlabel('Time (s)');
subplot(323); plot(time,T21_R42_Data,time,T21R42_New2,'.');
legend('Raw Data','Filtered Data'); title('T21-R42 Data Before and
After Filtering');
xlabel('Time (s)');
subplot(324); plot(time,T21_R43_Data,time,T21R43_New2,'.');
legend('Raw Data','Filtered Data'); title('T21-R43 Data Before and
After Filtering');
xlabel('Time (s)');
subplot(325); plot(time,T21_R44_Data,time,T21R44_New2,'.');
legend('Raw Data','Filtered Data'); title('T21-R44 Data Before and
After Filtering');
xlabel('Time (s)');
subplot(326); plot(time,T21_R45_Data,time,T21R45_New2,'.');
legend('Raw Data','Filtered Data'); title('T21-R45 Data Before and
After Filtering');
```

```
xlabel('Time (s)');

% Based on comparison of filtered data, choose receivers
% to use for position calculation:

RCM_3 = [Rx41; Rx42; Rx45];

%% Position Calculation:

R_T20 = [T20R41_New2 T20R42_New2 T20R45_New2];
R_T21 = [T21R42_New2 T21R42_New2 T21R45_New2];

addpath(strcat(RootPath,'Matlab\Test Data
Processing\Trilateration'));

Pos_CMD_T20 = zeros(length(time),3);
Pos_CMD_T21 = zeros(length(time),3);

for i = 1:length(time)
    Pos_CMD_T20(i,:) = ExplicitMethod_CMD(3,RCM_3,R_T20(i,:));
    Pos_CMD_T21(i,:) = ExplicitMethod_CMD(3,RCM_3,R_T21(i,:));
end

figure(21); hold on;
plot(time,Pos_CMD_T20(:,1),time,Pos_CMD_T20(:,2),time,Pos_CMD_T20(:,
3));
title('Calculated X, Y, and Z Position - T20'); legend('X
Position','Y Position','Z Position');
xlabel('Time (s)'); ylabel('Coordinate (mm)'); grid on;

%% Use Kalman Filter to filter position data:

% Set parameters for Kalman filter:
Sigma_Process = 3;       % specify std dev for process noise/error
Sigma_Measurement = 2;   % specify std dev for measurement
noise/error
Num_Lead_Pts = Num_Lead_Pts;

% Remove lead points from waveform for filtering:
Pos_CMD_T20_2 = Pos_CMD_T20((Num_Lead_Pts+1):length(time),:);
Pos_CMD_T21_2 = Pos_CMD_T21((Num_Lead_Pts+1):length(time),:);

time_filt = time(Num_Lead_Pts+1:length(time));

Pos_T20_Filt = zeros(size(Pos_CMD_T20_2));
Pos_T21_Filt = zeros(size(Pos_CMD_T21_2));

Pos_T20_Filt(:,1) =
RangeKalmanFilter(time_filt,Pos_CMD_T20_2(:,1),Sigma_Process,Sigma_M
easurement);
```

```
Pos_T20_Filt(:,2) =
RangeKalmanFilter(time_filt,Pos_CMD_T20_2(:,2),Sigma_Process,Sigma_M
easurement);
Pos_T20_Filt(:,3) =
RangeKalmanFilter(time_filt,Pos_CMD_T20_2(:,3),Sigma_Process,Sigma_M
easurement);

Pos_T21_Filt(:,1) =
RangeKalmanFilter(time_filt,Pos_CMD_T21_2(:,1),Sigma_Process,Sigma_M
easurement);
Pos_T21_Filt(:,2) =
RangeKalmanFilter(time_filt,Pos_CMD_T21_2(:,2),Sigma_Process,Sigma_M
easurement);
Pos_T21_Filt(:,3) =
RangeKalmanFilter(time_filt,Pos_CMD_T21_2(:,3),Sigma_Process,Sigma_M
easurement);

figure(22); clf; title('T20 Position Before and After Kalman
Filter');
subplot(311);
plot(time_filt,Pos_CMD_T20_2(:,1),'b',time_filt,Pos_T20_Filt(:,1),'r
');
xlabel('Time (s)'); ylabel('Coordinate (mm)');
legend('Before Kalman Filter','After Kalman Filter'); title('T20 - X
Position');
subplot(312);
plot(time_filt,Pos_CMD_T20_2(:,2),'b',time_filt,Pos_T20_Filt(:,2),'r
');
xlabel('Time (s)'); ylabel('Coordinate (mm)');
legend('Before Kalman Filter','After Kalman Filter'); title('T20 - Y
Position');
subplot(313);
plot(time_filt,Pos_CMD_T20_2(:,3),'b',time_filt,Pos_T20_Filt(:,3),'r
');
xlabel('Time (s)'); ylabel('Coordinate (mm)');
legend('Before Kalman Filter','After Kalman Filter'); title('T20 - Z
Position');

figure(23); clf; title('T21 Position Before and After Kalman
Filter');
subplot(311);
plot(time_filt,Pos_CMD_T21_2(:,1),'b',time_filt,Pos_T21_Filt(:,1),'r
');
xlabel('Time (s)'); ylabel('Coordinate (mm)');
legend('Before Kalman Filter','After Kalman Filter'); title('T21 - X
Position');
subplot(312);
plot(time_filt,Pos_CMD_T21_2(:,2),'b',time_filt,Pos_T21_Filt(:,2),'r
');
xlabel('Time (s)'); ylabel('Coordinate (mm)');
legend('Before Kalman Filter','After Kalman Filter'); title('T21 - Y
Position');
```

```
subplot(313);
plot(time_filt,Pos_CMD_T21_2(:,3),'b',time_filt,Pos_T21_Filt(:,3),'r
');
xlabel('Time (s)'); ylabel('Coordinate (mm)');
legend('Before Kalman Filter','After Kalman Filter'); title('T21 - Z
Position');

% Take average of T20 and T21 positions:
Pos_T_Filt = (Pos_T20_Filt + Pos_T21_Filt)/2;

figure(24); plot(time_filt,Pos_T_Filt);
legend('X-Position','Y-Position','Z-Position');

%% Heading Calculation:
Rad2Deg = 180/pi;

PSI_CMD = zeros(length(time),1);

for i = 1:length(time)
    PSI_CMD(i) = atan2(real((Pos_CMD_T20(i,2) - Pos_CMD_T21(i,2))),
...
                       real((Pos_CMD_T20(i,1) - Pos_CMD_T21(i,1))));
end

% figure(31); clf;
% plot(time,PSI_CMD*Rad2Deg)

Heading_CMD_Deg = PSI_CMD*Rad2Deg;

% Apply Kalman Filter to Heading
Sigma_Process       = 1;   % specify std dev for process noise/error
Sigma_Measurement   = 1;   % specify std dev for measurement
noise/error

Heading_CMD_Deg_Filt = RangeKalmanFilter(time,Heading_CMD_Deg, ...

Sigma_Process,Sigma_Measurement);

figure(32); clf;
plot(time,Heading_CMD_Deg,'b',time,Heading_CMD_Deg_Filt,'r');
title('Heading Angle'); legend('Before Kalman Filter','After Kalman
Filter');
xlabel('Time (s)'); ylabel('Heading Angle (deg)');

%% Parse Data and Resample

Impulse_Time        =   86.7; % in seconds
Sampling_Interval   =   0.05;
Time_Before_Impulse     =   1; % in seconds
Time_After_Impulse      =   4;
```

```
Impulse_Tspan       = [(Impulse_Time-
Time_Before_Impulse):Sampling_Interval:(Impulse_Time+Time_After_Impu
lse)]';

%% Define Inputs and Outputs for System Identification

Input_U         =   Ch3_Input;

Input_Norm      =   (Input_U - median(Input_U))/max(abs(Input_U-
median(Input_U)));
Impulse_Input   =
interp1(time,Input_Norm,Impulse_Tspan,'nearest');

Pos_T20_Output  =
interp1(time_filt,Pos_T20_Filt,Impulse_Tspan)+2500;
Pos_T21_Output  =   interp1(time_filt,Pos_T21_Filt,Impulse_Tspan);
Heading_Output  =
interp1(time,Heading_CMD_Deg_Filt,Impulse_Tspan);

Input       =   -Impulse_Input;
Output_T20  =   real(Pos_T20_Output(:,3));
Output_T21  =   real(Pos_T21_Output(:,3));
Output_Heading  = Heading_Output;

figure(34); clf;
subplot(2,1,1); plot(Impulse_Tspan,Input,'green'); legend('Impulse
Signal'); title('Ch3 Command');
xlabel('Time (s)');
subplot(2,1,2);
plot(Impulse_Tspan,Output_T20,'blue',Impulse_Tspan,Output_T21,'red')
; legend('T20 Position','T21 Position');
xlabel('Time (s)'); ylabel('Coordinate (mm)');

%% Display Commands and Responses for All Channels

Ch1_Norm = (Ch1_Input - median(Ch1_Input))/max(abs(Ch1_Input-
median(Ch1_Input)));
Ch2_Norm = (Ch2_Input - median(Ch2_Input))/max(abs(Ch2_Input-
median(Ch2_Input)));
Ch3_Norm = (Ch3_Input - median(Ch3_Input))/max(abs(Ch3_Input-
median(Ch3_Input)));
Ch4_Norm = (Ch4_Input - median(Ch4_Input))/max(abs(Ch4_Input-
median(Ch4_Input)));

figure(35); clf;
subplot(2,1,1); plot(Impulse_Tspan,Input,'green'); legend('Heading
Command Signal');
title('Heading Command'); xlabel('Time (s)');
subplot(2,1,2); plot(Impulse_Tspan,Output_Heading); legend('Heading
Response');
title('Heading Response'); ylabel('Heading (deg)');
%%
```

```
figure(36); clf;
subplot(211); plot(time,Ch1_Norm,'green');
title('Ch1 Command'); xlabel('Time (s)');
subplot(212);
plot(time_filt,Pos_T20_Filt(:,1),'blue',time_filt,Pos_T21_Filt(:,1),
'red');
title('Position Response - Left/Right'); xlabel('Time (s)');
legend('T20 X Position','T21 X Position');

figure(37); clf;
subplot(211); plot(time,Ch2_Norm,'green');
title('Ch2 Command'); xlabel('Time (s)');
subplot(212);
plot(time_filt,Pos_T20_Filt(:,2),'blue',time_filt,Pos_T21_Filt(:,2),
'red');
title('Position Response - Front/Back'); xlabel('Time (s)');
legend('T20 Y Position','T21 Y Position');

figure(38); clf;
subplot(211); plot(time,Ch3_Norm,'green');
title('Ch3 Command'); xlabel('Time (s)');
subplot(212);
plot(time_filt,Pos_T20_Filt(:,3),'blue',time_filt,Pos_T21_Filt(:,3),
'red');
title('Position Response - Altitude'); xlabel('Time (s)');
legend('T20 Z Position','T21 Z Position');

figure(39); clf;
subplot(211); plot(time,Ch4_Norm,'green');
title('Ch4 Command'); xlabel('Time (s)');
subplot(212); plot(time,Heading_CMD_Deg_Filt,'blue');
title('Heading Response (deg)'); xlabel('Time (s)');
legend('Heading');
```

```
function    [Filtered_Distance]   =
RangeKalmanFilter(Time_Vector,Distance_Vector,Process_Noise,Measurem
ent_Noise)

% function  [Filtered_Distance]   =
RangeKalmanFilter(Time_Vector,Disance_Vector)
% inputs    :   Time_Vector (vector of time stamps corresponding to
each
%               data point
%           :   Distance_Vector (vector of distance measurements)
%           :   Process_Noise (std dev of process noise, aka-error)
%           :   Measurement_Noise (std dev of measurement noise,
aka-error)
% outputs   :   Filtered_Distance (distance measurements after
filtering)

Num_Pts = length(Time_Vector);

% Initialize estimation variables:
dt_avg = mean(diff(Time_Vector));

X_initial = [mean(Distance_Vector(1:10)); 0];
X_estimate = [];
P_mag_estimate = [];
Predict_State = [];
Predict_Var = [];
R_estimate = []; % Range estimate (postion)
V_estimate = []; % 'Velocity' estimate (change in range)

Filtered_Distance = [];

% Set noise parameters:
Process_Noise = Process_Noise; % (10?) std dev, variability in
process (acceleration)
Measurement_Noise = Measurement_Noise; % (20?) std dev, variability
in measurement system
Error_Measurement = Measurement_Noise^2; % Measurement noise
covariance
Error_Process = Process_Noise^2 * [dt_avg^4/4 dt_avg^3/2; dt_avg^3/2
dt_avg^2]; % Process noise covariance
P = Error_Process; % Initialize P best guess (using dt_avg)

for t = 1:Num_Pts

    if t == 1
        X_estimate = X_initial;
    else

        dt = Time_Vector(t) - Time_Vector(t-1);
        Error_Process = Process_Noise^2 * [dt^4/4 dt^3/2; dt^3/2
dt^2];
```

```
        % Define governing equations:
        A = [1 dt; 0 1];
        B = [0; 0]; % There is no known control input (U)
        C = [1 0];

        % Define noise parameters dependent on dt:

        X_estimate = A*X_estimate; % B is assumed to be 0 because U
is unknown
        Predict_State = [Predict_State; X_estimate(1)];
        P = A*P*A' + Error_Process;
        Predict_Var = [Predict_Var; P]; % Predict covariance
        K = P*C'*inv(C*P*C' + Error_Measurement); % Calculate Kalman
Gain

        % Update state estimation
        X_estimate = X_estimate + K*(Distance_Vector(t) -
C*X_estimate);

        %Update covariance estimation
        P = (eye(2) - K*C)*P;
    end

    Filtered_Distance = [Filtered_Distance; X_estimate(1)];

end
```