University of Kentucky
UKnowledge

Theses and Dissertations--Biosystems and Agricultural Engineering

Biosystems and Agricultural Engineering

2013

PRECISE EVALUATION OF GNSS POSITION AND LATENCY ERRORS IN DYNAMIC AGRICULTURAL APPLICATIONS

Michael P. Sama
University of Kentucky, michael.sama@uky.edu

Click here to let us know how access to this document benefits you.

Recommended Citation

Sama, Michael P., "PRECISE EVALUATION OF GNSS POSITION AND LATENCY ERRORS IN DYNAMIC AGRICULTURAL APPLICATIONS" (2013). *Theses and Dissertations--Biosystems and Agricultural Engineering*. 14.
https://uknowledge.uky.edu/bae_etds/14

This Doctoral Dissertation is brought to you for free and open access by the Biosystems and Agricultural Engineering at UKnowledge. It has been accepted for inclusion in Theses and Dissertations--Biosystems and Agricultural Engineering by an authorized administrator of UKnowledge. For more information, please contact UKnowledge@lsv.uky.edu.

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained and attached hereto needed written permission statements(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine).

I hereby grant to The University of Kentucky and its agents the non-exclusive license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless a preapproved embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's dissertation including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Michael P. Sama, Student

Dr. Timothy Stombaugh, Major Professor

Dr. Dwayne Edwards, Director of Graduate Studies

PRECISE EVALUATION OF GNSS POSITION AND LATENCY ERRORS IN DYNAMIC AGRICULTURAL APPLICATIONS

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the College of Engineering
at the University of Kentucky

By
Michael Patrick Sama
Lexington, Kentucky
Director: Dr. Timothy Stombaugh, Associate Extension Professor of
Biosystems and Agricultural Engineering
Lexington, Kentucky
2013

Copyright © Michael Patrick Sama 2013

ABSTRACT OF DISSERTATION

PRECISE EVALUATION OF GNSS POSITION AND LATENCY ERRORS IN DYNAMIC AGRICULTURAL APPLICATIONS

A method for precisely synchronizing an external serial data stream to the pulse-per-second (PPS) output signal from a global navigation satellite-based system (GNSS) receiver was investigated. A signal timing device was designed that used a digital signal processor (DSP) with serial inputs and input captures to generate time stamps for asynchronous serial data based on an 58593.75 Hz internal timer. All temporal measurements were made directly in hardware to eliminate software latency. The resolution of the system was 17.1 μs, which translated to less than one millimeter of horizontal position error at travel speeds typical of most agricultural operations.

The dynamic error of a TTS was determined using a rotary test fixture. Tests were performed at angular velocities ranging from 0 to 3.72 rad/s and a radius of 0.635 m. Average latency from the TTS was shown to be consistently near 0.252 s for all angular velocities and less variable when using a reflector based machine target versus a prism target. Sight distance from the target to the TTS was shown to have very little effect on accuracy between 4 and 30 m. The TTS was determined to be a limited as a position reference for dynamic GNSS and vehicle auto-guidance testing based on angular velocity.

The dynamic error of a GNSS receiver was determined using the rotary test fixture and modeled as discrete probability density functions for varying angular velocities and filter levels. GNSS position and fixture data were recorded for angular velocities of 0.824, 1.423, 2.018, 2.618, and 3.222 rad/s at a 1 m radius. Filter levels were adjusted to four available settings including; no filter, normal filter, high filter, and max filter. Each data set contained 4 hours of continuous operation and was replicated three times. Results showed that higher angular velocities increased the variability of the distribution of error while not having a significant effect on average error. The distribution of error tended to change from normal distributions at lower angular velocities to uniform distributions at higher angular velocities. Internal filtering was shown to consistently increase dynamic error for all angular velocities.

KEYWORKS: Global Navigation Satellite-based Systems, Precision Agriculture, Tracking Total Station, Dynamic Error, Standardized Testing

_____Michael Patrick Sama_____
Student's Signature

_____
Date

PRECISE EVALUATION OF GNSS POSITION AND LATENCY ERRORS IN DYNAMIC AGRICULTURAL APPLICATIONS

By

Michael Patrick Sama

Timothy Stombaugh

Director of Dissertation

Dwayne Edwards

Director of Graduate Studies

_____

ACKNOWLEDGMENTS

Throughout my graduate studies, I have benefited from the insight and direction of several individuals.

I would like to thank my advisor, Tim Stombaugh, and committee members; Scott Shearer; Michael Montross; and James Lumpp, for their contributions to the studies in this dissertation. Additionally I wish to thank the department chairs; Richard Gates; Scott Shearer; and Sue Nokes, for the opportunities they've provided me throughout my time as a graduate student, staff member, and future faculty member.

Additional thanks go to the Biosystems and Agricultural Engineering graduate students and staff. I was fortunate to have worked with three outstanding machinery Ph.D. students; Joe Luck; Rodrigo Zandonadi; and Santosh Pitla. Together, we were able to take on some impressive projects. The staff at the machine shop; Carl King; Lee Rechtin; Ed Hutchens; and Brett Childers, were instrumental in providing technical and fabrication assistance.

My family has assisted me throughout this process with their unwavering support. First, I want to thank my wife, Diana, for her constant love and support. Next, I wish to thank my parents; Frank and Sheila, siblings; Joseph, Michelle, and Neal for providing motivation to always better myself. I would also like to thank my brother-in-law, Chris Reilly, for introducing me to Biosystems and Agricultural Engineering.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Determining Dynamic GNSS Accuracy

The standard method for evaluating dynamic Global Navigation Satellite based Systems (GNSS) error involves constraining the position and speed of a receiver to a known value. In certain applications, the actual position at any point in time is not as important as the path that was taken. An example of this is in row crop production where the error along the prescribed path does not interfere with productivity when compared to error off the path. Error off the path is commonly referred to as off-track error and is computed as the shortest distance to the actual path (ISO, 2010). Testing for off-track error simply requires a spatially defined test fixture and has been well documented (Han et al., 2004); (Taylor et al., 2004); (Stombaugh et al., 2008); (Easterly et al., 2010); and (Gavric et al., 2011). This measure of accuracy is effectively limited to one orthogonal dimension. Computing along-track error is not possible without additional constraints to determine the receiver's actual position with respect to time. Along-track error is particularly important in instances where variable rate applications are performed.

One method for determining three-dimensional position error utilized a tracking total station (TTS) to track the location of a GNSS receiver (Sama et al., 2009). A TTS is a survey grade instrument capable of precisely tracking a prism or reflector target under dynamic conditions (Krischner and Stempfhuber, 2008). The TTS and GNSS devices operate asynchronously; therefore, the position data streams from both devices do not necessarily line up temporally or spatially. A method is needed for relating each device's sampling interval so that the position of one measurement device can be accurately

projected to another. Once known, the TTS could potentially serve as a relative position reference for a GNSS receiver synchronous to the GNSS sampling interval. Attempts have been made to dynamically reference the position of a TTS to a test fixture but not to a GNSS receiver (Depenthal, 2008).

1.2 Objectives

The proposed objectives were to develop a method for determining and modeling the two-dimensional position error of a GNSS receiver under dynamic conditions and to investigate a method for incorporating position uncertainty in agricultural mapping applications. The majority of this dissertation deals with the process of building a dynamic GNSS error model and can be subdivided into the following individual objectives.

- The synchronization of serial data streams relative to GNSS time
- The development of a rotary test fixture for controlling the angular velocity of a TTS target and GNSS receiver
- Evaluation of a TTS for feasibility of use in dynamic GNSS accuracy testing under ISO 12188-1 conditions
- Investigation of a dynamic GNSS error distributions for use in agricultural mapping applications

The method for incorporating position uncertainty using a dynamic GNSS error model has been treated independently as a separate individual objective.

- Application of a dynamic GNSS error model

Each objective has been divided into a chapter, each with its own introduction, materials and methods, results and discussion, and conclusions.

1.3 Note on Terminology

Throughout this dissertation several terms relating to the Global Positioning System (GPS) and Global Navigation Satellite System (GNSS) will be used. GPS refers to a specific satellite based navigation system where as GNSS is a general term that includes GPS as well as other global systems such as the Russian Global Navigation Satellite System (GLONASS). For clarity, GNSS will be used in most instances when referring to a satellite based navigation receiver except when the term GPS is included in the title of a standard, publication, or product. The term GNSS accuracy includes elements of accuracy and precision in the form of mean error and standard deviation of error, respectively. A complete list of abbreviated terms can be found in Appendix 10.

1.4: Literature Review

The literature review for this dissertation is provided in three segments. The first is a summary of existing research on dynamic GNSS testing in chronological order. The second segment is a summary of the ION STD 101 standard. The third segment is a summary of the ISO 12188-1 standard along with a brief explanation on its implementation.

1.4.1 Existing Research on Dynamic GNSS Testing

Efforts to understand how vehicle dynamics affect GNSS receiver accuracies in agricultural applications have dated back to the mid 1990's. Saunders et al. (1996) were some of the first to publish the concept of a repeatable test course by which different

GNSS receivers could be compared under dynamic conditions. They tested three different commercial GNSS receivers at varying price points. All of the GNSS receivers tested utilized some form of differential correction. Receivers were mounted to the roof of a test vehicle and driven at 4 km/h around a surveyed test route. Navigation data were recorded at sampling rate of 1 Hz. Results showed that two of the GNSS receivers were not suitable for precision farming applications due to issues such as signal corruption. They concluded that a direct comparison between GNSS receivers was an effective method for determining which device would perform the best under actual field conditions.

Borgelt et al. (1996) evaluated the accuracy of GNSS using course/acquisition (C/A) code processing techniques. Their experiment consisted of a bar code reader attached to the GNSS receiver which read a series of bar codes placed at predetermined locations. They found that the predetermined locations could be measured using a GNSS receiver to within 1 m. The authors concluded that the 1 m level of accuracy observed was sufficient for yield mapping applications when used in a combine harvester. They also pointed out that carrier phase kinematic position methods have the potential to result in higher accuracies, but that their experimental method would need to be revised to mitigate data acquisition problems when assessing sub-centimeter level GNSS receivers. This paper provided a good indication of some of the problems that future research would deal with – specifically, how to test a real-time-kinematic (RTK) GNSS receiver when, in many cases, a RTK GNSS receiver is the most accurate means of determining 3D position in agricultural applications.

Stombaugh et al. (2002) presented the concept developing a test standard specific to assessing dynamic GNSS accuracy in agriculture. They showed that differences in the way GNSS manufacturers presented performance specifications made it difficult to directly compare different GNSS receivers. They also showed that the static performance specifications typically given are not consistent with dynamic performance measured on a rotary test fixture. They proposed a framework for GNSS manufacturers in agriculture to follow that included standardized terminology as well as recommendations on dynamic test fixture paths. The proposed paths were mostly typical of row crop production and included multiple straight parallel paths connected with constant radii turns.

Han et al. (2002) published a paper on the use of Kalman filters to post process GNSS position measurements in parallel tracking applications. They collected GNSS data from two receivers mounted on the cab of a tractor along the centerline of the direction of travel. The tractor was driven in parallel passes separated by 12.2 m at a speed of 1.54 m/s. The GNSS receivers were sampled at 1 Hz. They found that the maximum cross-track error between the GNSS receiver and the actual path was 9.83 m but could be reduced to 2.76 m through the use of a Kalman filter. However, the average cross-track error was only slightly reduced from 0.58 m to 0.56 m. They concluded that the GNSS receiver bias error was the cause of the lack in reduction of average cross-track error and that more research would be needed to determine how to reduce GNSS receiver bias error for parallel tracking applications.

Ehsani et al. (2003) presented a paper on determining the dynamic accuracy of five low-cost GNSS receivers that were not typically used in precision agricultural applications. Their method utilized a test vehicle with the GNSS receivers mounted in a straight line

along the direction of travel. An RTK GNSS receiver was used to determine the actual path of the test vehicle. The vehicle was driven in a manner to simulate the parallel passes of an agricultural operation. Data were collected for 15 minutes over multiple parallel passes, a process which was repeated 5 times over a 2 month period. Results showed that cross-track error in the north and south directions was higher than in the east and west directions for all receivers tested and that the dynamic accuracies of the low-cost GNSS receivers were consistent with the static accuracy specifications. They noted that their method was limited to linear paths and would need to be updated to incorporate curved paths and to assess pass-to-pass accuracy.

Han et al. (2004) published a study on a method for evaluating dynamic GNSS performance under parallel tracking applications. They proposed that pass-to-pass accuracy was the most important performance specification relative to agricultural applications. Their method included eight different GNSS receivers with varying levels of differential correction. GNSS receivers were mounted individually to a test vehicle and driven manually along an existing field. A total of 68 tests were performed at varying times over a day, with several tests removed due to issues with the RTK correction. Results were presented as unsigned cumulative distribution functions, from which 95% accuracy and 2-inch success rates were determined. They concluded that all GNSS receivers tested had a large bias and that pass-to-pass accuracy was highly variable between tests with respect to each GNSS receiver. This variability led to their decision to use a cumulative distribution function to express the ensemble dynamic GNSS error from multiple individual tests.

Taylor et al. (2004) published a study on dynamic testing of GNSS receivers that closely resembled the subsequent ISO 12188-1 standard. They used a test fixture to precisely control the path of a GNSS receiver. Their test fixture consisted of a 0.8 km rail track and rail car that travelled at two speeds in both directions. They tested two GNSS receivers that were designed specifically for precision agriculture applications over a 24 hour period and discovered several interesting results. The first discovery was that the cross-track errors of each GNSS receiver were correlated over short periods of time. This meant that, while over longer periods cross-track error appears to be random, over short periods there is a deterministic component that makes analysis using standard statistical methods difficult. The second discovery was that cross-track error was periodic, which meant that data must be taken for a long enough time to fully capture the process. They concluded that pass-to-pass accuracy tests were more meaningful due to the lack of distinct frequency content and the shorter test durations required.

Cole et al. (2004) presented a paper on the development of a test fixture for measuring dynamic GNSS performance. They constructed a closed circuit monorail track and cart system designed to replicate traditional movements in an agricultural application. The fixture consisted of two 91 m straight sections spaced 12 m apart that were connected by a 6 m radius turn on one end and a variable radius turn on the other. An RTK GNSS receiver was used to determine the tolerance of the test fixture and was found to be approximately 2.5 cm. A low-cost GNSS receiver was subsequently tested against the path generated by the RTK GNSS receiver and found to have deviated by a maximum of 1.75 m. The authors concluded that the test fixture would be useful for the development of a dynamic GNSS receiver test standard specific to agriculture.

Smith and Thomson (2005) published a paper focusing on methods to determine the latency of a GNSS receiver used in aerial agricultural applications. Position latency results in linear changes in position error with respect to speed. They used an Air Tractor 402B aircraft to fly a GNSS receiver across a light beam which spatially located the aircraft. The position measurement from the GNSS receiver was compared with the position reference to determine deviations in the direction of flight. Results showed maximum deviations of 9 m when travelling at 60 m/s. The authors concluded that the deviations in position were due to timing or data processing after GNSS position measurements were made.

Chan et al. (2006) published a study that focused on the small scale dynamic accuracy of GNSS receivers intended for civil engineering projects. They fabricated a 2D movable platform capable of 50 mm of motion in both directions at a maximum frequency of 2 Hz. This allowed for sinusoidal testing in a single dimension, sinusoidal testing in two dimensions and more arbitrary patterns derived from wind-induced models on buildings. They found that when undergoing sinusoidal motion, minimum amplitudes of 5 mm in the horizontal planes and 10 mm in the vertical plane at frequencies less than 1 Hz were required to accurately detect motion using a GNSS receiver when sampled at 20 Hz. They concluded that this level of accuracy was sufficient for measuring the dynamic displacement of tall structures under wind loading.

Wu et al. (2006) published an article on the influence of travel direction on dynamic GNSS accuracy. They hypothesized that the distribution of GNSS satellites in the sky had an effect on the accuracy of a GNSS receiver travelling in a given direction. Results show that there was a significant difference in the dilution of precision (DOP) in the

north and east directions at their test location. They found that this affected the cross-track error based upon the direction of travel. The authors concluded that the orientation at which dynamic GNSS testing is conducted influences accuracy measurements.

Harbuck et al. (2006) presented a paper that incorporated the vehicle, guidance system, and GNSS receiver into the accuracy measurement. They focused on determining the ability of an agricultural vehicle operation under GNSS auto-guidance to navigate the same straight path over a 15 week test period. They used a tracking total station (TTS) to verify the path of the vehicle during each pass and found that an RTK GNSS receiver used with the auto-guidance system was able to track the same path to within 10 cm. Two commercially available satellite based differential correction methods on the same GNSS receiver resulted in a maximum of 60 cm to 140 cm of error. They concluded that the wide range in performance emphasized the need to address temporal variations in accuracy specifications, as the results from their tests exceeded the manufacturer specifications for pass-to-pass accuracy.

Thomson et al. (2007) published a study on latency issues when using GNSS receivers in aircraft for geo-referencing images used in remote sensing. They used an imaging system that was synchronized with the GNSS data output to determine the location of each image and compared that information to reference data taken on the ground. They found that one differentially corrected GNSS receiver lagged the actual position by as much as 8 m where as two other low-cost standalone GNSS receivers led the actual position by over 126 m. The authors concluded some form of position compensation is needed when using standalone GNSS receivers at high speeds for remote sensing.

Stombaugh et al. (2008) presented a paper on standardized evaluation of dynamic GNSS performance. They updated the test fixture designed by Cole et al. (2004) to include a feedback control system for maintaining constant speed throughout the test fixture. Tests were performed at 2.5 m/s on multiple receivers to provide standard developers with data as part of the draft ISO 12188-1 standard. They identified several of the steps that need to be taken to collect, format, and process standardized GNSS accuracy data. Results showed that the expected accuracy due to DOP and the standard deviation of cross-track error of a GNSS receiver were highly correlated and varied throughout the day. This emphasized the importance of replicating tests throughout durations long enough to fully capture the average performance of a GNSS receiver. They concluded that the data presented would be useful to help set constraints on dynamic GNSS receiver accuracy testing at the ISO level.

Sama et al. (2009) presented a paper on dynamic GNSS testing and applications. They focused on developing a ground based position reference for validating dynamic GNSS performance. A tracking total station (TTS) was used in conjunction with a rotary test fixture to determine the feasibility of the TTS as a position reference. The local coordinate system defined in ISO-12188 was tested against the local coordinate system used in the TTS and found to be consistent to within 3 mm over a 40 meter range. The authors pointed out that some form of temporal synchronization was needed to interpolate TTS measurements to GNSS measurements. Otherwise, the TTS data could only be used as a reference path and not an actual reference position. Assuming temporal synchronization was possible, they presented two interpolation methods that could be used to calculate the position of a GNSS receiver that was co-located with a TTS target.

The first was a polar coordinate method and the second was a quadratic function where the three nearest TTS points in time relative to the GNSS measurement were used to define a second-order curve. The second method was deemed to be a superior method because it could be efficiently implemented using linear algebra techniques on a computer and could better represent paths other than circles. They concluded that the use of a TTS as a position reference for dynamic GNSS accuracy testing was feasible if future work in data synchronization was completed.

Perez-Ruiz et al. (2010) published a paper on how the type of GNSS correction signal affects performance in assisted guidance systems used in agricultural vehicles. They tested two agricultural GNSS receivers that, when combined, were capable of receiving correction data from five difference sources. They mounted both devices to the cab of a tractor and recorded position data along six 600 m rows over several weeks. They found that all correction methods produced accuracies to within 1 m, which they deemed sufficient for yield monitoring. Accuracies of less than 0.5 m, sufficient for broadcast seeding, fertilizing, and herbicide application, were observed for some correction methods but varied in terms of how many measurements fell within the tolerance. Accuracies less than 0.04 m were not found using any correction method, including RTK correction, which they defined as necessary for transplanting and drill seeding. They concluded that the level of accuracy for the GNSS correction technology at the time was sufficient for performing many of the field operations used in farming, but there are still some limitations on the most precise operations.

Gavric et al. (2011) published an article relating to short- and long-term dynamic accuracies for GNSS receivers using a test fixture. They designed an oval test fixture

with 18 m straight sections separated by 7.6 m and connected by constant radius turns. These dimensions were 5 times smaller than what had been defined in the recently published ISO-12188-1 standard. They tested a low-cost GNSS receiver for 24 hours and found that the cross-track error was less than 1.3 m and the pass-to-pass error was less than 0.7 m for 95% of the data recorded. They concluded that this level of accuracy was sufficient for some agricultural operations but did not identify any specific operations.

1.4.2 ION STD 101

The Institute of Navigation (ION) standard on recommended test procedures for GPS receivers (ION, 1997) has been used as a basis for much of the research relating to dynamic GNSS accuracy in agriculture. The standard defined common testing procedures for determining static and dynamic accuracy as well as other performance characteristics including initialization time and reacquisition time.

Initialization time was defined for two instances, Initialized Time to First Fix (INIT TTFF) and Warm Start Time to First Fix (WARM TTFF). The difference between these two conditions was how long the receiver was powered off. In both instances, the objective was to determine the amount of time it took for a GNSS receiver to output uncorrected and corrected 3D navigation data to accuracy levels of 600 meters and 20 meters, respectively. While the accuracy level may not be near what is required for modern auto-guidance systems in agriculture, the concept of how long it takes for a GNSS receiver to output navigation data is still relevant.

Reacquisition time is a specific test to determine how long a GNSS receiver takes to resume normal function after a temporary blockage of satellite signals. A key difference

from WARM TTFF is that the GNSS receiver is maintained in a powered on state. This type of test is also relevant to agriculture when working around the borders of fields where trees may obstruct satellite coverage.

The static component of ION STD 101 had the purpose of determining the accuracy to which a GNSS receiver can determine its position relative to a reference position. The reference position must be known to an accuracy better than 10 times the expected accuracy of the GNSS receiver being tested. Data must be recorded for a 24 hour period and specified as either a single measure of the difference between the measured data and the reference position or the ensemble mean of several single measures. Accuracies at the 50^{th}, 68^{th}, 95^{th}, and 99.99^{th} percentiles are required for measured data with the 95^{th} percentile being the primary measure of accuracy for predicting performance.

The dynamic component of ION STD 101 added testing criterion to allow the determination of the accuracy of a GNSS receiver in a moving vehicle. Movement was defined as a trajectory that must be repeated for 1 hour durations and replicated 3 times, equally spaced, over a single 24 hour period. The standard provided the provision that this component of the test can be conducted using simulated GPS signals, which can aide in determining the effects of vehicle dynamics on GNSS accuracy. Accuracy specifications determined from the dynamic testing are specified in the same manner as the static component.

1.4.3 ISO 12188-1

The International Organization for Standardization (ISO) standard on test procedures for positioning and guidance systems in agriculture (ISO, 2010) was developed as result of

research relating to dynamic GNSS accuracy up to its inception. The standard places an emphasis on the dynamics under which a GNSS receiver must be tested. Instead of a loosely defined trajectory, the standard prescribes a test fixture or course consisting of straight parallel passes of a minimum length connected by at least one constant radius U-turn. The speeds at which tests are performed are specified at (0.1 ± 0.05) m/s, (2.5 ± 0.2) m/s, and (5.0 ± 0.2) m/s.

Position error is segmented into several specifications including (1) absolute dynamic accuracy, (2) relative dynamic accuracy, (3) absolute vertical position accuracy, (4) relative vertical position accuracy, (5) short-term dynamic accuracy, (6) long-term cross track accuracy, (7) U-turn accuracy, and (8) absolute accuracy after signal loss. Specifications (1) through (4) are expressed as the mean plus the standard deviation of all signed errors $(\overline{x} + S_x)$ in their respective directions. Specifications (5) through (8) are expressed as the root mean squared of the mean plus the standard deviation of error $\left(\sqrt{2}(\overline{x} + S_x)\right)$. Short-term dynamic accuracy and long-term cross-track accuracy isolates deviations tangent to the test fixture from absolute dynamic accuracy. These are particularly useful specifications for agriculture as it gives a producer an indication of how accurately they can navigate the same path on a short-term and a long-term basis.

A localized projection is used to convert the geographical coordinates generated by a GNSS receiver into a Cartesian coordinate system. The projection is based on an ellipsoid and converts latitudes and longitudes in decimal degrees into meters in two orthogonal directions (Equation 1).

$$F_{lon} = \frac{\pi}{180}\left(\frac{a^2}{\sqrt{a^2\cos^2\varphi + b^2\sin^2\varphi}} + h\right)\cos(\varphi)$$

$$F_{lat} = \frac{\pi}{180}\left(\frac{a^2 b^2}{(a^2\cos^2\varphi + b^2\sin^2\varphi)^{\frac{3}{2}}} + h\right)$$

Equation 1

F_{lat} and F_{lon} are the conversion factors in units of meters per degree. The latitude location of the test site is represented by φ and must be within 1000 m of any point on the test course. Choosing φ to be south-west of a bounding box containing the entire test course is a convenient method for ensuring all output coordinates are in the first quadrant of the Cartesian coordinate system (i.e. positive). The final three parameters h, a, and b represent the average height of the test course above the ellipsoid in meters, semi-major axis of the ellipsoid in meters, and semi-minor axis of the ellipsoid in meters, respectively. The semi-major and semi-minor axes vary with the form of the ellipsoid model used in the GNSS receiver. The most commonly used ellipsoid model is the World Geodetic System 1984 (WGS 84). Values of $a = 6378137$ and $b = 6356752.3142$ from WGS 84 are used in this dissertation. Applying the projection requires special precautions to eliminate rounding error. GNSS receivers capable of providing centimeter level accuracies can generate 10 or more significant figures in the angular minute measurement. The tester should carry all constants to enough significant figures to preserve the precision of the projection. Geographical coordinates transmitted using the NMEA 0183 Interface Standard (NMEA, 2000) combine integer degrees and decimal minutes into a single decimal number using the format shown in Table 1.

Table 1: NMEA 0183 Coordinate Format

Latitude	Longitude
DDMM.MMMMMMMM	DDDMM.MMMMMMMM
D = Degree Digit, M = Minute Digit	

Converting coordinates from the NMEA 0183 format into decimal degrees can be achieved by isolating the integer degrees and decimal minutes, converting decimal minutes into decimal degrees, and recombining. Assuming a coordinate is stored as a floating point number, Equation 2 coverts the NMEA 0183 format into decimal degrees using a floor (round down) operation to split out the integer degree component and the modulus operator to split out the decimal minute component. A complete implementation of the ISO 12188-1 projection using MATLAB (MathWorks, 2012) can be found in Appendix 8.

$$Decimal\ Degrees = floor\left(\frac{X}{100}\right) + \frac{mod(X, 100)}{60}$$

Equation 2

X = Latitude or Longitude in NEMA 0183 Format

CHAPTER 2: SYNCHRONIZING SERIAL DATA STREAMS WITH GNSS TIME

2.1 Introduction

Many management operations in agriculture rely on global navigation satellite-based systems (GNSS) for navigation, data acquisition, and precise application of materials. Standards have been developed to prescribe the methods for evaluating and expressing GNSS accuracy under dynamic conditions (ISO, 2010). Those standards mandate that a position reference is needed with an accuracy that is one order of magnitude better than the GNSS device being tested. One device capable of determining three-dimensional position of sufficient spatial precision to evaluate high precision GNSS receivers is a tracking total station (TTS). A TTS is a survey grade instrument capable of precisely tracking the position of a prism or other target object under dynamic conditions (Krischner and Stempfhuber, 2008). Preliminary testing has shown how a TTS can be used to track the location of a moving GNSS receiver (Sama et al., 2009).

When testing a GNSS receiver using a TTS as a reference, the two devices operate asynchronously relative to each other; therefore, the position data streams from both devices do not necessarily line up temporally or spatially. Spatial synchronization is possible with precise benchmark backsight calibration of the TTS, but a method is needed for relating the sampling interval of each device so that the position measurement of one device can be accurately projected or interpolated to the other temporally. Latency in the TTS measurement may result in position error, but it has been shown that latency compensation is possible if the TTS signal latency is consistent (Boniger and Tronicke, 2010).

Modern high-precision GNSS receivers commonly include a pulse-per-second (PPS) signal output, which is precisely synchronized to GPS time. This highly precise timing signal could serve as the reference point for a system to synchronize GNSS and TTS data streams. Embedded systems have been designed which can be synchronized to the PPS signal to within 100 ns (Berns and Wilkes, 2000). This level or precision is unnecessary for most precision agriculture operations. For example, a machine capable of travelling at 10 m/s (which is very fast for typical agricultural field operations) only requires a temporal resolution of 0.1 ms to achieve a spatial resolution of 1 mm. Behn et al. (2008) developed a method for time-stamping multiple RS-232 serial streams using a MINI-ITX (17 cm x 17 cm) computer running a Linux operating system. They primarily focused on synchronizing the computer clock with a GPS clock but did not examine the effects of software latency due to receiving and processing the serial data streams. Hwang et al. (2004) outlined some important parameters to consider when synchronizing embedded systems to GPS time. They point out that the small variability in frequency between two quartz crystal oscillators can generate large discrepancies in measured time over long periods; however, continuous synchronization using a PPS signal eliminated accumulated error by updating the relative time stamp every second.

Modern digital signal processors include input capture (IC) interfaces that allow discrete events to be time-stamped directly through hardware. Once configured, a rising or falling edge causes the controller to latch the current state of an internal timer or counter into the IC register for processing at a later time. This eliminates any software latency from manually copying registers once a discrete event has been identified. Using ICs also

eliminates the need to factor in serial data baud rates because the time stamp is always generated by the beginning of the message as opposed to after it has been fully received.

2.1.1 Objective

The objective of this study was to fabricate and test a signal timing device that was capable of simultaneously time-stamping a PPS signal and an asynchronous serial data stream. The device will be used as part of a larger study to evaluate RTK GNSS receivers under dynamic conditions and must conform to the accuracy requirements prescribed by the ISO-12188 standard.

2.2 Materials and Methods

2.2.1 Hardware Description

A signal timing device was designed around the dsPIC30F4011 (Microchip, Chandler, AZ) digital signal processor (DSP). The primary features of the DSP utilized in this study were two universal asynchronous receiver transmitter (UART) interfaces and two input capture (IC) interfaces. The device consisted of two printed circuit boards (PCBs) stacked on top of one another with surface mount and through-hole components populating each PCB (Figure 1).

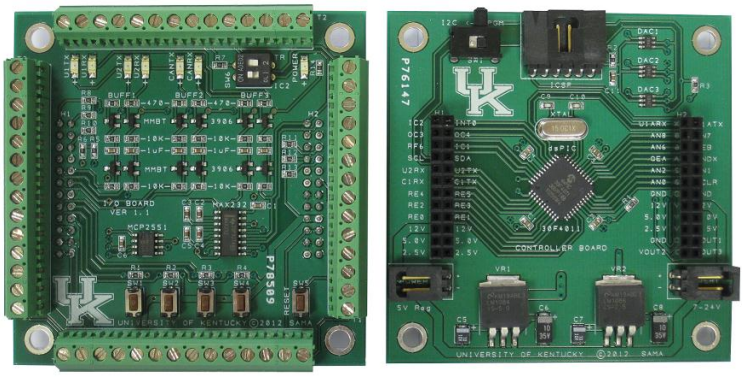


Figure 1: Signal Timing Device (Left: Top PCB, Right: Bottom PCB)

The bottom PCB contained the DSP, voltage regulators, programming interface, and bypass capacitors. The top PCB contained the RS-232 level shifter, light-emitting diodes (LED), and terminal headers. The DSP clock source was provided by a 15.0 MHz crystal oscillator with a specified frequency tolerance of ±30 ppm (±450 Hz) and a frequency stability of ±50 ppm (±750 Hz). The external clock source was divided by four to generate an internal instruction clock frequency of 3.75 MHz, which was the base clock used for generating baud rates and timers. A 16-bit timer was used for time-stamping the PPS signal from the GNSS receiver and the serial output from the TTS. The timer prescaler was set to 64, which reduced the timing frequency to approximately 58.6 KHz. This ensured that the timer did not cycle more than once between each PPS event; otherwise, software routines would have been needed to keep track of timing overflow cycles, which would have prevented all time-stamping from being performed exclusively in hardware. The temporal resolution of this method was 17.1 μs, which corresponded to the maximum deviation between when an event occurs and the previous timer value.

The PPS signal from the GNSS receiver was connected to the IC1 pin. Since each GNSS data point included a UTC time stamp that was synchronous with the PPS output, no processing was necessary on the GNSS data stream. The TTS serial stream was fed into UART1 through a level shifter. The level shifter converted the RS-232 signal voltages to the TTL levels used by the DSP. The TTL serial stream was also fed into the IC2 pin for edge detection. The propagation delay for a high- to low-level output through the MAX232 level shifter was specified to be 500 ns, which was several orders of magnitude smaller than the resolution of the signal timing device.

2.2.2 Software Description

The software running on the DSP was completely event driven through the use of hardware interrupts. An interrupt service routine (ISR) was written to process serial data as each character was received. A PPS signal event generated an interrupt where the state of the IC1 time stamp was stored into a local variable. The time stamp, along with an identifier, was transmitted to a PC from UART2. Serial data entering UART1 from the TTS was processed on a character basis. The format of the TTS data stream is shown in Table 2.

Table 2: Sample Data from a Trimble SCS930 Universal Total Station

ASCII	**HEX**
0	30 0D
37=#.######	0A 33 37 3D XX XX 2E XX XX XX XX XX XX 0D
38=#.######	0A 33 38 3D XX XX 2E XX XX XX XX XX XX 0D
39=#.######	0A 33 39 3D XX XX 2E XX XX XX XX XX XX 0D
>	0A 3E

#: 0-9, XX: 0x30-0x39

Each time a character was received, the corresponding ISR was checked to see what character was sent and then that character was retransmitted through UART1 to a PC. When a '>' character (0x3E) was received, the ISR enabled IC2 to record the time of the next falling edge, which corresponded to the subsequent start bit from the first '0' character of the next TTS data point. The start bit transition from high to low triggered IC2 to capture the local time stamp from the timer. The IC2 event also generated an

interrupt that stored the time stamp into a local variable. The time stamp was transmitted out of UART1 immediately after each TTS data point for post processing (Figure 2).

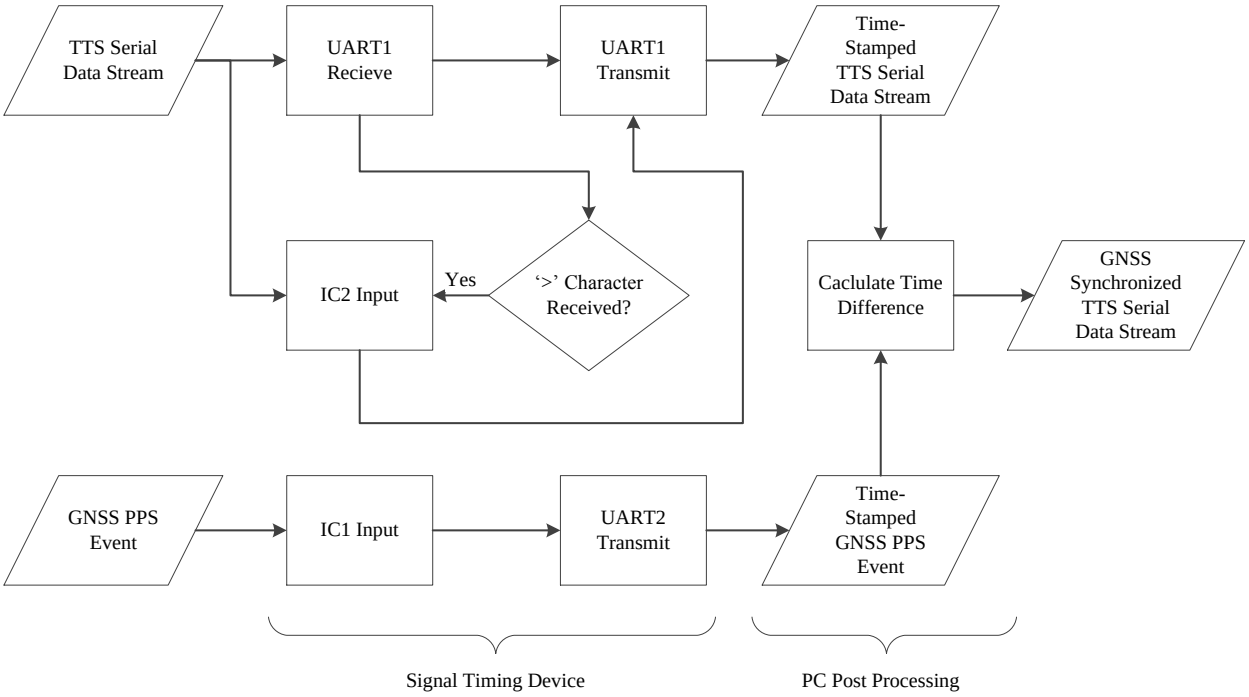


Figure 2: Signal Flow Diagram

2.2.3 Signal Timing Device Validation Procedures

A vb.net PC program (Microsoft Corporation, Redmond, WA) was written to simulate the TTS output and capture the serial data from the timing controller (Figure 3).

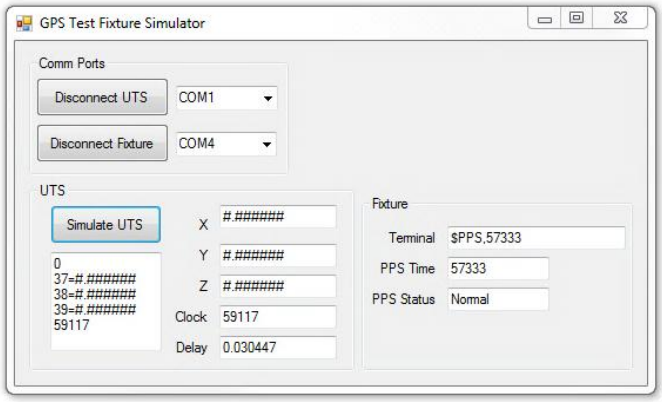


Figure 3: Interface Software

A function generator was used to simulate a PPS signal. A digital oscilloscope (DPO 4034, Tektronix, Beaverton, OR) was configured to capture the PPS and TTS data streams. The delay measurement feature was configured to record the delay between the falling edge of the PPS signal and the first falling edge of the TTS data stream, which was equivalent to the measurement made by the signal timing device (Figure 4).

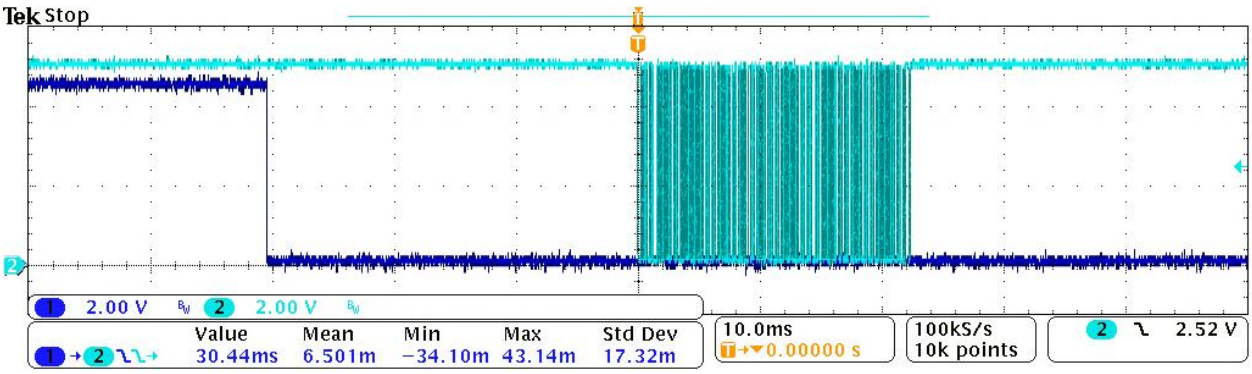


Figure 4: Time Delay between a Pulse and a Serial String using a Digital Oscilloscope

The software recorded two 16-bit numbers, t_{TTS} and t_{PPS}, corresponding to the local timer value when each signal was received. The local timer values were converted to time

delay in seconds using Equation 3 if the timer value for the PPS time stamp was less than the timer value for the TTS time stamp and Equation 4 otherwise.

$$(t_{TTS} - t_{PPS}) / 58593.75 \quad \text{Equation 3}$$

$$(2\text{^}16 - t_{PPS} + t_{TTS}) / 58593.75 \quad \text{Equation 4}$$

2.3 Results and Discussion

Eight samples were taken with delays between the PPS signal and TTS data stream ranging from 30.4 ms to 760 ms using both the digital oscilloscope and the signal timing device (Table 3).

Table 3: Validation Data

PPS Time Stamp	TTS Time Stamp	Signal Timing Device (ms)	Oscilloscope (ms)
57333	59117	30.447	30.44
35978	43548	129.19	129.1
17889	33015	258.15	258.2
51763	6118	339.47	339.4
5343	33217	475.71	475.7
61165	35014	564.42	564.4
37398	9708	645.90	645.9
12068	1970	760.11	760.1

A linear regression was performed between the digital oscilloscope and the signal timing device (Figure 5).

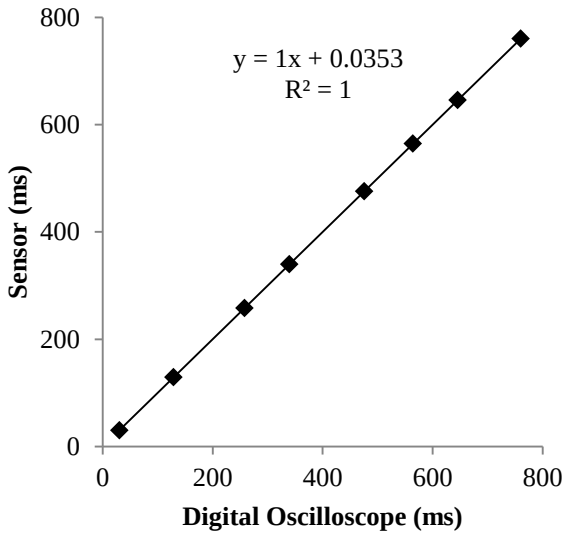


Figure 5: Linear Regression of Calibration Data

The regression resulted in a slope of 1 and an offset of 0.0353 with a coefficient of determination of 1. The standard error of the regression was 47.2 µs. In most cases, the first four significant figures in each measurement were exactly the same. The small discrepancy between the digital oscilloscope and the sensor board can be attributed to rounding or truncation. The digital oscilloscope was limited to four significant figures for all temporal measurements regardless of the time base while the signal timing device was capable of measuring a fifth significant figure with a theoretical error of no more than 17.1 µs.

2.4 Conclusions

The signal timing device developed in this study was designed to accurately time-stamp PPS signals and asynchronous serial data streams to within 17.1 µs. Validation showed that the sensor board time measurements closely matched time measurements made with a digital oscilloscope. The variability between the two measurement systems was

negligible when considered for agricultural applications. Assuming a maximum vehicle speed of 10 m/s, all results showed that position error would be less than 1 mm as a result of errors in time measurement. This system will be useful for testing GNSS devices and auto-guidance systems under the ISO-12188 standard. The precision of this system may be improved by removing the restriction that all time comparisons are made in hardware or by using a device with IC registers that are larger than 16 bits.

CHAPTER 3: TEST FIXTURE DESIGN AND ANALYSIS

3.1 Introduction

Test fixtures are used in GNSS testing to control the path of a receiver under repeatable conditions. In the case of ISO 12188-1 standardized GNSS testing (ISO, 2010) a receiver travels along a closed path with parallel straight passes. This method simulates typical agricultural applications but is limited in terms of spatial and temporal accuracy. The fixed path constrains the receiver in a vertical and single horizontal direction. The position along the orthogonal horizontal direction is unknown without the addition of a position reference. A simple way to avoid this issue is to use a rotary test fixture, where the path of the receiver is constrained to a circle. The position in the vertical direction is fixed and the horizontal directions can be calculated from an angle sensor. A disadvantage of using a rotary test fixture is that the receiver is being evaluated in a manner which is not consistent with actual use. Therefore, the performance specifications derived from circular testing may not reflect the performance observed in the field.

3.2 Materials and Methods

A rotary test fixture was designed to evaluate TTS performance and to develop a dynamic GNSS error model. It consisted of a drive train, control system, and structure.

3.2.1 Rotary Test Fixture Drive Train

Angular velocity and acceleration criteria were used to specify the fixture drive train. Velocities similar to those used in standardized GNSS testing, 0.1 m/s to 5.0 m/s, were

used as minimum and maximum criteria, respectively. A conservatively high mass for the rotation components was estimated to be 45 kg. A one meter radius circular hoop model for the moment of inertia was calculated using Equation 5.

$$I = mr^2 = 45\ kg \times (1\ m)^2 = 45\ kg\ m^2 \qquad \text{Equation 5}$$

An angular acceleration criterion of 10 seconds from 0 to 5 m/s (Equation 6) was used to determine the power required by the fixture (Equation 7) and the motor power (Equation 8).

$$\alpha = \frac{\Delta\omega}{\Delta t} = \frac{5\ rad/s}{10\ s} = 0.5\ rad/s^2 \qquad \text{Equation 6}$$

$$T = I\alpha = 45\ kg\ m^2 \times 0.5\frac{rad}{s^2} = 22.5\ Nm \qquad \text{Equation 7}$$

$$P = T\omega = 22.5\ Nm \times 5\ rad/s = 112.5\ W \qquad \text{Equation 8}$$

A margin of error of two was added to account for mechanical inefficiency as well as the reduction in torque when operating the motor as slower speeds. The resulting motor specification was determined to be 225 W (0.3 HP). To achieve the velocity criteria, a 3-phase inverter-duty AC motor was selected along with a 30:1 gear reduction. The motor

had a rated speed of 1720 RPM which was reduced to 57.3 RPM through the gear reduction. A speed of 57.3 RPM corresponded to an output angular velocity of 6.00 rad/s, or 6.00 m/s at a 1 m radius. Powering the motor with a variable frequency drive enabled the output angular velocity to be reduced to nearly 0.1 rad/s (Figure 6).

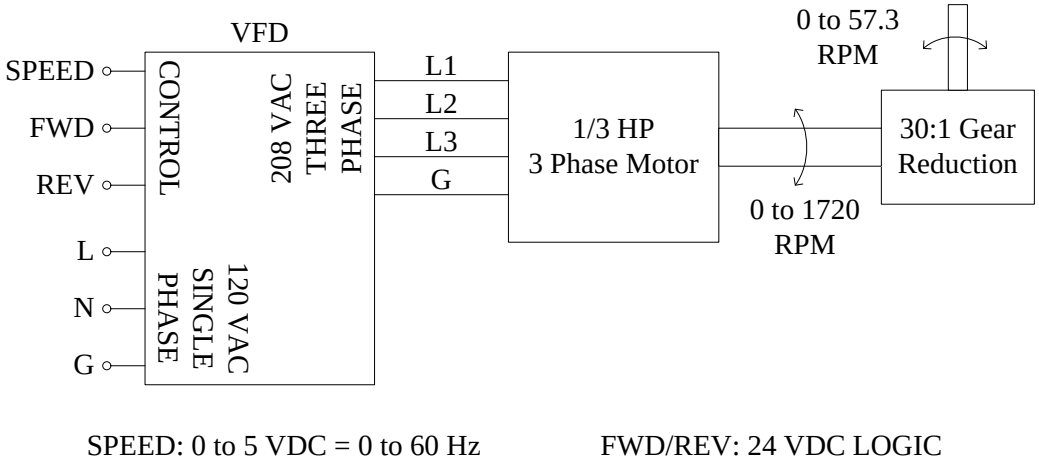


Figure 6: Fixture Drive Train and Control

3.2.2 Rotary Test Fixture Control System

The rotary test fixture was controlled using the signal timing device described in *Chapter 2.* A 3-wire RS232 interface (19,200-8-N-1) connected the signal timing device to a PC. Two commands were used to update the speed and direction of the rotary test fixture. The speed command was a fixed width string formatted as "$V,####*". The "$V" characters were used as an identifier and the #### characters were decimal numbers between 0000 and 4095, which were used to represent analog output voltages to the VFD between 0 and 5 V. The "*" character was used as a terminating character along with the non-printable characters carriage return and line feed. The direction command was a fixed width string formatted as "$M,#,#*". The "$M" characters were used as an identifier and each # character was a decimal 0 or 1 which corresponded to the FWD and

REV signals on the VFD. The "*" character was used as a terminating character along with the non-printable characters carriage return and line feed.

An optical encoder with 2,500 pulses per revolution was mounted to the bottom of the output drive shaft. When operating in 4X quadrature mode, the encoder generated 10,000 transitions between the A and B phases. This resulted in 10,000 counts per revolution or and angular resolution of 6.28×10^{-4} radians. The dsPIC30F4011 used in the signal timing device included a hardware quadrature encoder interface. The A and B phases along with I, an index pulse, were connected to the signal timing device. Any transitions or pulses automatically incremented, decremented, or reset the quadrature encoders counter without the need for a software routine.

3.2.3 Rotary Test Fixture Structure

A welded steel framework was fabricated to house the rotary test fixture drive train and control system (Figure 7). The structure consisted of 3.81 cm (1.5 in) tube with 0.303 cm (11-gauge) thick sheet metal welded to the top, middle, and bottom surfaces. The remaining sides were covered with 0.121 cm (18-gauge) sheet metal using socket head cap screws. A self-adhesive silicone gasket was adhered to all removable surfaces to minimize water infiltration. A steel stand was fabricated to mount the rotary test fixture structure to the roof of the Charles E. Barnhart Building in Lexington, KY (Figure 8). A detailed set of engineering drawings for the rotary test fixture structure can be found in Appendix 6.

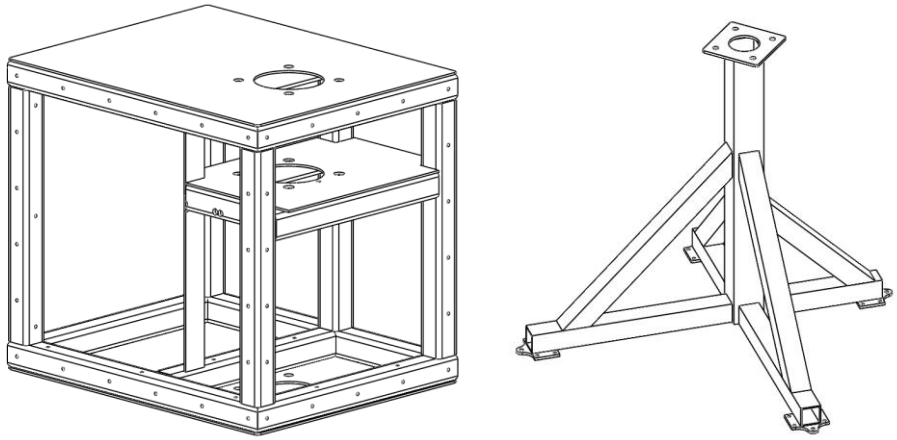


Figure 7: Dimetric View of Rotary Test Fixture Structure (left) and Trimetric View of Stand (right)

Figure 8: Rotary Test Fixture

3.2.4: Test Procedures

The ability to control the angular velocity of the rotary test fixture was evaluated at twelve control settings between 0255 and 3071 in increments of 0256. Data were sampled at 2.5 Hz for 100 seconds resulting in 250 angular velocity measurements per control input.

3.3 Results and Discussion

3.3.1 Angular Velocity Control

The angular velocity of the rotary test fixture was stable over the test interval for each speed tested. Figure 9 illustrates the angular velocity versus time for 100 seconds. The minimum and maximum speeds recorded were within 0.010 rad/s of each other, which represented $\pm 1\%$ of the mean angular velocity. An interesting trend in the data was the cyclical pattern in the angular velocity with respect to time. The period of the oscillations in angular velocity for a control input of 0255 was approximately 15 seconds. Based on and average angular velocity of 0.441 rad/s, one full revolution of the test fixture took 14.25 seconds. Therefore the angular position of the rotary test fixture may have had an influence on angular velocity.

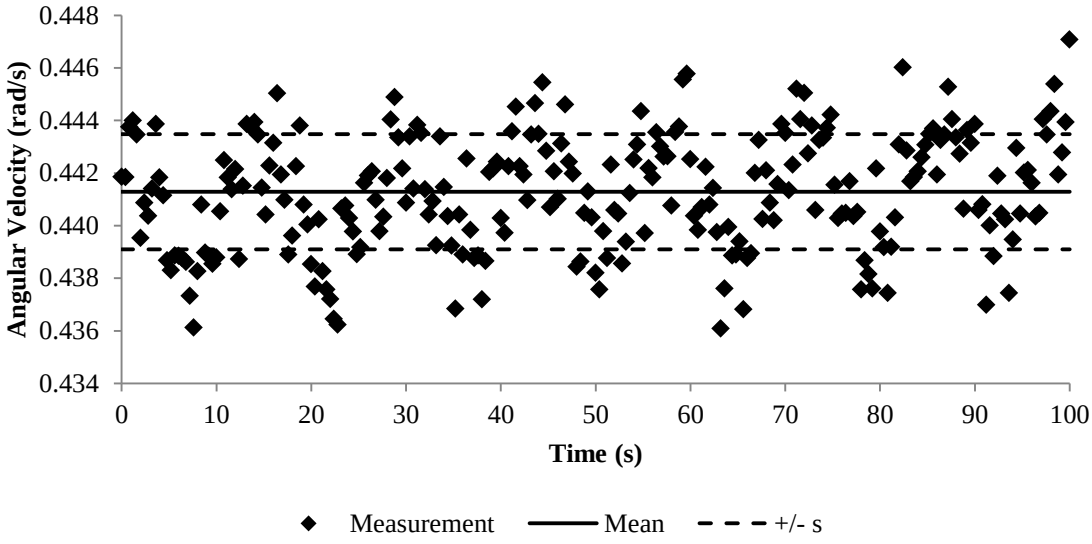


Figure 9: Angular Velocity for Control Input 0255

The amount of variability in angular velocity was small relative to the mean angular velocity for all control inputs tested (Table 4). The standard deviation of angular velocity ranged from 0.002 rad/s to 0.007 rad/s for control inputs of 0255 and 3071, respectively.

Table 4: Variability in Rotary Test Fixture Performance

Control Input	Control Voltage (V)	Mean Angular Velocity (rad/s)	Standard Deviation (rad/s)
0255	0.311	0.441	0.002
0511	0.624	0.848	0.002
0767	0.937	1.251	0.003
1023	1.249	1.660	0.003
1279	1.562	2.070	0.004
1535	1.874	2.481	0.004
1791	2.187	2.893	0.005
2047	2.499	3.304	0.005
2303	2.812	3.718	0.006
2559	3.125	4.124	0.006
2815	3.437	4.533	0.007
3071	3.750	4.946	0.007

The mean angular velocity increased linearly with respect to the control input (Figure 10). Inverting the linear regression equation provided a calibration equation for the control input to attain a desired output speed (Equation 9).

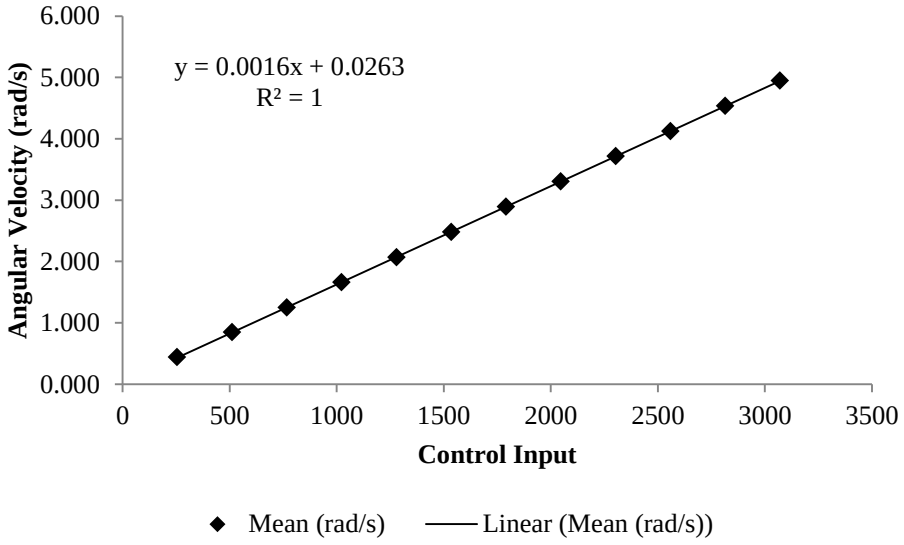


Figure 10: Linear Regression of Mean Angular Velocity versus Control Input

$$Control\ Input = round(625 \times Angular\ Velocity - 16.4375) \qquad \text{Equation 9}$$

3.4 Conclusions

The rotary test fixture was able to precisely control the angular velocity of a GNSS receiver or TTS target up to 5 rad/s with a standard deviation of 0.007 rad/s. Placing the GNSS receiver or TTS target at a 1 m radius will result in an instantaneous velocity of 5 m/s with a standard deviation of 0.007 m/s. These specifications are well within the speed requirements defined in ISO 12188-1.

CHAPTER 4: TRACKING TOTAL STATION TESTING

4.1 Introduction

The evaluation of dynamic GNSS and auto-guidance accuracy has been accomplished using several methods. One method uses a test fixture which confines a GNSS receiver to an known open track (Taylor et al., 2004) or closed track (Stombaugh et al., 2008). The other method uses a highly accurate measurement system to determine a reference path from which performance characteristics can be derived (Easterly et al., 2010).

Up to this point, dynamic GNSS and auto-guidance testing have focused on pass-to-pass accuracy and off-track error. While this may be sufficient for many field operations including harvesting and tilling, it fails to address performance characteristics in variable rate and precision planting applications.

Al-Gaadi and Ayers (1999) demonstrated a system where GIS and GPS are used to spatially prescribe application rates based on site-specific needs. They used a laptop computer as a control interface between a GPS receiver and a chemical pump to adjust the application rate based on the current position and the desired application rate.

Luck et al. (2011) estimated that off-rate errors from GNSS position data due to turning movements have resulted in up to 24% of a field receiving the wrong application rate. Zandonadi et al. (2011) developed a computation tool for estimating off-target application areas for a given field boundary. Results from nine field boundaries showed that off-target application area varied from 9% to 24% and could be reduced to less than 1% when using individual section control.

In all of these studies, the along-path error of the vehicle was ignored. This can be attributed to not having a viable method for measuring or predicting along-track error. Along-track error may significantly change the interpretation of field data or predictions of application rate in simulations. For example, GNSS latency in a section control scenario will cause the system to incorrectly apply material near boundary transitions. A measurement system is needed that can independently verify position accuracy under dynamic conditions that includes off-track and along-track error.

A tracking total station (TTS) is a survey grade instrument capable of precisely tracking a prism or other target under dynamic conditions (Krischner and Stempfhuber, 2008). Sama et al. (2009) showed that a TTS can be accurately tied in to the local coordinate systems used in standardized GNSS and vehicle guidance testing to within several millimeters. These features make a TTS a possible candidate for a position reference from which dynamic GNSS accuracy and auto-guidance performance can be evaluated against. Krischner and Stempfhuber (2008) identified that the accuracy of a TTS under dynamic conditions is limited by varying latency, lack of internal synchronization between measurement subsystems, and the quality of the target. Some of these limitations have been addressed by modern systems and the authors concluded that a TTS can perform kinematic measurements up to 50 m with an accuracy of a few millimeters. Testing was limited to a straight path and only off-track error was measured, which may not describe how a TTS will perform when the target travelling at higher velocities or along a curved path. Other issues such as the latency between the TTS measurement and when that measurement is transmitted limit the usefulness for evaluating along-track

error. Time discrepancies of a few milliseconds could result in several centimeters of additional error. The amount of latency as well as the variability in latency must be known to better understand how accurately a TTS can track a moving target.

Using a TTS to asses GNSS accuracy and auto-guidance performance requires synchronizing two independent measurement systems. A GNSS device computes position at consistent interval that is accurate to within a microsecond of universal coordinated time (UTC) (Daly et al., 1991). A TTS on the other hand operates independently of any external clock source. This creates an issue where GNSS and TTS measurements do not line up temporally. Calculating the error of a GNSS measurement requires a reference that can be sampled synchronously with GNSS time. Therefore, an interpolation method is needed to synchronize TTS measurements with GNSS time. Many GNSS devices include a pulse-per-second (PPS) output that indicates the GNSS second interval. This signal can be used to determine when a TTS measurement has been made relative to GNSS time.

4.1.1 Objective

The objectives of this study were to:

- determine the latency present in a TTS measurement.
- determine the horizontal measurement error when compensating for latency.
- develop an interpolation method to calculate the TTS target position at the GNSS time interval.

Parameters including the sight distance from the TTS to the target and angular velocity of the target were investigated to see what influence they had on accuracy and precision.

4.2 Materials and Methods

4.2.1 Test Procedures

Latency of the TTS measurement relative to the test fixture was evaluated using two different targets. The first target was a prism-based device with eight individual prisms for tracking in any direction. The second target was an active reflector based machine target which was designed for position control of construction equipment. Seven data sets were recorded for varying angular velocities. Each data set consisted of 256 measurements. Horizontal position error of the TTS measurement relative to the test fixture was evaluated using only the active reflector machine target. Thirty-three data sets were recorded for eleven varying angular velocities at three sight distances from the TTS.

4.2.2 Data Collection

The test fixture was operated via PC through an embedded controller. The controller served an interface between the PC and VFD as well as a signal timing device for time-stamping TTS position measurements and PPS events. Two RS-232 serial ports (19200-8-N-1) were used to send and receive command, timing, and position data. The first serial port was configured to receive speed and direction commands from the PC and send PPS event timestamps along with the fixture position. The second serial port was configured to receive position measurements from the TTS and retransmit each measurement to the PC along with a timestamp and fixture position. A program was

written using Microsoft Visual Studio 2010 to record data into a comma-separated-value (CSV) file and allow the speed and direction input (Figure 11).

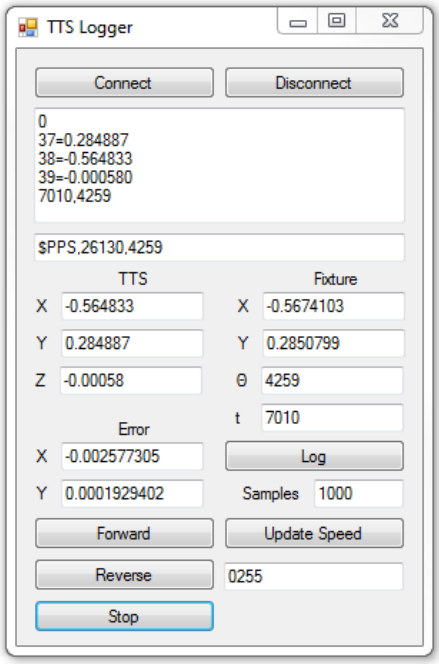


Figure 11: PC Program for Fixture Control and TTS Data Logging

4.2.3 Data Processing

The data streams from the test fixture controller were compiled and stored in a CSV file format. Each data file contained nine elements (Table 5).

Table 5: Sample Data File

n	t_{TTS}	θ_{FIX}	x_{FIX}	y_{FIX}	x_{TTS}	y_{TTS}	z_{TTS}	t_{PPS}	θ_{PPS}
1	49020	5264	-0.62628	-0.10485	-0.6321	0.002357	-0.00247	16098	4870
2	6963	5547	-0.59786	-0.21397	-0.62285	-0.10829	-0.00202	16098	4870
3	30298	5827	-0.55119	-0.31531	-0.59508	-0.21577	-0.00152	9953	5582
4	53880	6110	-0.48672	-0.40783	-0.54723	-0.3162	-0.00105	9953	5582
5	11745	6389	-0.40814	-0.48647	-0.48346	-0.40749	-0.00062	3825	6293

The t_{TTS} column contained the result of a 16-bit timer running at 58.59375 kHz. The θ_{FIX} column represented the text fixture angle measured by the encoder. The x_{FIX} and y_{FIX} columns were the actual horizontal location of the TTS target when TTS position measurement was received. The x_{TTS}, y_{TTS} and z_{TTS} columns were the 3-D location measured by the TTS. The t_{PPS} and θ_{PPS} columns were the 16-bit timer value and fixture angle at the most recent PPS event.

The average TTS measurement latency was determined using a MATLAB script (Appendix 7) that read in the CSV file. The time stamp (Equation 10) and encoder angle (Equation 11) of two subsequent measurements were used to calculate an instantaneous angular velocity (Equation 12). The test fixture angle (Equation 13) based on the TTS measurement was calculated from the *xTTS* and *yTTS* coordinates using a four-quadrant arctangent function. The latency of each measurement (Equation 14) was calculated by taking the difference between the actual fixture angle and TTS measured fixture angle, and dividing by the angular velocity.

$$\Delta t_{TTS}[n]\ (s) = \begin{cases} \dfrac{t_{TTS}[n] - t_{TTS}[n-1]}{58593.75}, & t_{TTS}[n] \geq t_{TTS}[n-1] \\ \dfrac{65536 - t_{TTS}[n-1] + t_{TTS}[n]}{58593.75}, & else \end{cases} \quad \text{Equation 10}$$

$$\Delta\theta_{FIX}[n]\ (rad) = \begin{cases} \dfrac{2\pi(\theta_{FIX}[n] - \theta_{FIX}[n-1])}{10000}, & \theta_{FIX}[n] \geq \theta_{FIX}[n-1] \\ \dfrac{2\pi(10000 - \theta_{FIX}[n-1] + \theta_{FIX}[n])}{10000}, & else \end{cases} \quad \text{Equation 11}$$

$$\omega[n]\ \left(\frac{rad}{s}\right) = \frac{\Delta\theta_{FIX}[n]}{\Delta t_{TTS}[n]} \qquad \text{Equation 12}$$

$$\theta_{TTS}[n]\ (rad) = \begin{cases} atan2(y_{TTS}[n], x_{TTS}[n]), & y_{TTS}[n] \geq 0 \\ atan2(y_{TTS}[n], x_{TTS}[n]) + 2\pi, & y_{TTS}[n] < 0 \end{cases} \qquad \text{Equation 13}$$

$$Latency[n]\ (s) = \begin{cases} \dfrac{\theta_{TTS}[n] - \theta_{PPS}[n]}{\omega[n]}, & \theta_{TTS}[n] > \theta_{PPS}[n] \\ \dfrac{2\pi - \theta_{TTS}[n] + \theta_{PPS}[n]}{\omega[n]}, & else \end{cases} \qquad \text{Equation 14}$$

TTS measurement error was evaluated using a MATLAB script that interpolated the location of the total station at a PPS event while taking TTS latency into account. First, the t_{PPS} and θ_{PPS} columns were processed for changes in content. A change in PPS values indicated that a PPS event had occurred before the current TTS measurement. There were typically 102 PPS events for every 256 TTS measurements. Four TTS points and their respective timestamps were used to calculate a third-order interpolation functions in the horizontal directions, two before the PPS event and two after (Equation 15). The time at which the PPS event occurred was then used as an input to the interpolation functions for estimating the total station position at the exact time of the PPS event (Equation 16). The process was repeated for every PPS event in the data set with at least two TTS measurements before and after the PPS event.

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} t_{TTS}[n_1]^3 & t_{TTS}[n_1]^2 & t_{TTS}[n_1] & 1 \\ t_{TTS}[n_2]^3 & t_{TTS}[n_2]^2 & t_{TTS}[n_2] & 1 \\ t_{TTS}[n_3]^3 & t_{TTS}[n_3]^2 & t_{TTS}[n_3] & 1 \\ t_{TTS}[n_4]^3 & t_{TTS}[n_4]^2 & t_{TTS}[n_4] & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_{TTS}[n_1] \\ x_{TTS}[n_2] \\ x_{TTS}[n_3] \\ x_{TTS}[n_4] \end{bmatrix} \quad \text{Equation 15}$$

$$x_{INT} = a(t_{PPS}[n]^3) + b(t_{PPS}[n]^2) + c(t_{PPS}[n]) + d \quad \text{Equation 16}$$

4.3 Results and Discussion

4.3.1 Latency Results

The average latencies were determined to not differ significantly between the prism and reflector targets for all angular velocities tested based on a single factor ANOVA ($\alpha = 0.05$). However, there was a significant difference in the average latency for each target with respect to angular velocity. The prism target had a P-value of 0.016 while the reflector target had a P-value less than 0.001. More importantly, the variability in latency measurements between the prism and reflector targets was not the same (Figure 12).

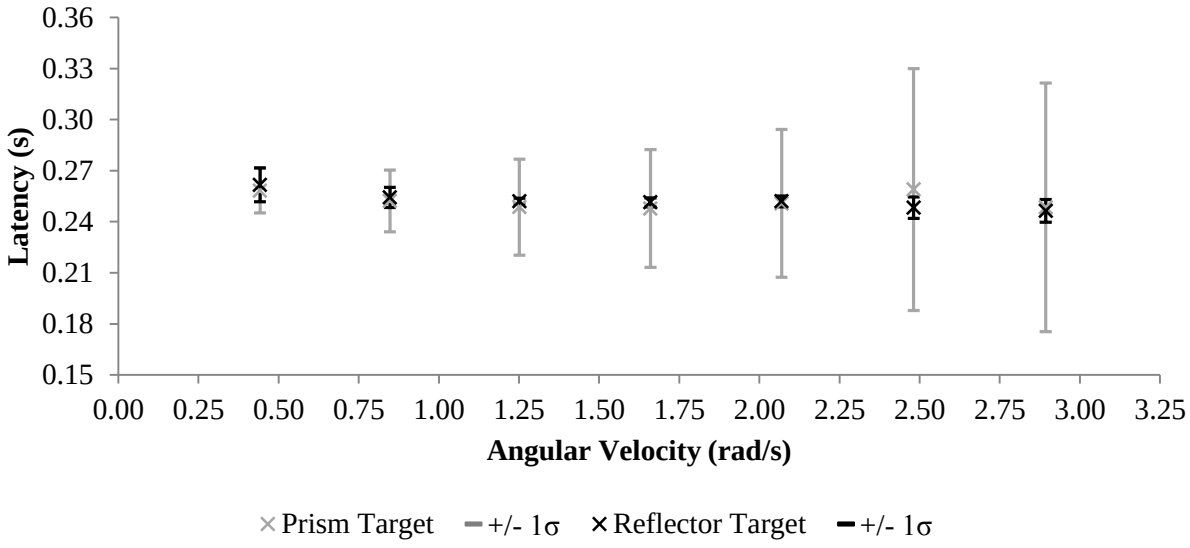


Figure 12: Latency Results for Prism and Reflector Targets

The prism target exhibited an increasing trend in the standard deviation of latency, ranging from 0.0131 to 0.0731 seconds. The reflector target standard deviation of latency measurements were consistently smaller and had a range from 0.0017 to 0.0100 seconds (Table 6).

Table 6: Summary of Latency Results

	Prism Target Latency		Reflector Target Latency	
Angular Velocity (rad/s)	**Mean (s)**	**Standard Deviation (s)**	**Mean (s)**	**Standard Deviation (s)**
0.442	0.2583	0.0131	0.2617	0.0100
0.847	0.2522	0.0182	0.2543	0.0060
1.251	0.2486	0.0282	0.2521	0.0017
1.660	0.2478	0.0346	0.2514	0.0026
2.069	0.2508	0.0435	0.2520	0.0032
2.482	0.2589	0.0710	0.2483	0.0063
2.894	0.2485	0.0731	0.2464	0.0066
Mean	**0.2522**	**0.0402**	**0.2523**	**0.0052**

4.3.2 Interpolation Results

The average measurement errors in both horizontal directions and at varying speeds were highly correlated (Table 7).

Table 7: Correlation of TTS Measurement Error for Direction and Distance

		4.255 m		**14.689 m**		**30.184 m**	
		x_{error}	y_{error}	x_{error}	y_{error}	x_{error}	y_{error}
4.255 m	x_{error}	1.000	0.997	0.998	0.999	0.998	0.999
	y_{error}	0.997	1.000	0.998	0.999	0.997	0.999
14.689 m	x_{error}	0.998	0.998	1.000	0.999	0.999	0.999
	y_{error}	0.999	0.999	0.999	1.000	0.999	0.999
30.184 m	x_{error}	0.998	0.997	0.999	0.999	1.000	0.998
	y_{error}	0.999	0.999	0.999	0.999	0.998	1.000

Single factor ANOVA ($\alpha = 0.05$) was used to test for significant differences in the magnitude of position error between differences in distance from the TTS at varying angular velocities. There was a statistically significant difference in the average error for angular velocities for 0.000, 0.441 and 0.847 rad/s, but the actual average amount of difference was less than 1.5 mm. There was no significant difference in the average error for all other angular velocities (Table 8).

Table 8: Mean and Standard Deviation of Position Error Magnitude for varying Sight Distances and Angular Velocities

Angular Velocity (m/s)	**4.255 m**	**14.689 m**	**30.184 m**	**P-value**
0.000	0.0030 *0.0005*	0.0029 *0.0002*	0.0015 *0.0003*	< 0.001
0.441	0.0041 *0.0018*	0.0038 *0.0013*	0.0046 *0.0020*	0.004
0.847	0.0069 *0.0021*	0.0079 *0.0029*	0.0081 *0.0029*	0.006
1.251	0.0104 *0.0024*	0.0107 *0.0027*	0.0111 *0.0023*	0.114

1.660	0.0164	0.0170	0.0164	0.205
	0.0033	*0.0024*	*0.0025*	
2.069	0.0243	0.0245	0.0242	0.868
	0.0054	*0.0033*	*0.0035*	
2.481	0.0343	0.0351	0.0353	0.377
	0.0062	*0.0052*	*0.0046*	
2.893	0.0477	0.0490	0.0500	0.163
	0.0094	*0.0091*	*0.0060*	
3.303	0.0651	0.0663	0.0680	0.335
	0.0159	*0.0141*	*0.0098*	
3.718	0.0880	0.0881	0.0900	0.708
	0.0195	*0.0184*	*0.0176*	

Mean (m) *Stadard Deviation (m)*

Both the mean and standard deviation of error tended to increase as angular velocity increased. The larger amount of variability in error at higher angular velocities was expected as any variation in latency is directly reflected to position error. There is evidence that filtering in the TTS may have introduced additional position error. The horizontal position measurement tended to shift towards the point of rotation as angular velocity increased (Figure 13). This may be due to low-pass filtering inside the TTS for noise reduction.

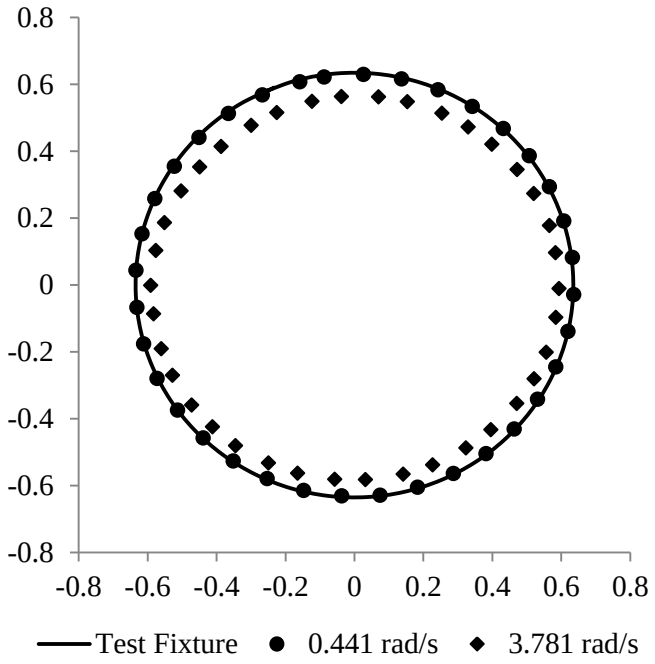


Figure 13: Horizontal Position Error versus Angular Velocity

4.4 Conclusions

Testing has shown that the prism and reflector targets have a similar average latency. The variability in latency for the prism target was several times greater than the reflector target. Since averages cannot be used for position measurements made on-the-fly, it is recommended that the prism target not be used for dynamic applications where millimeter resolution is required. There is a distinct and significant trend in average latency for the reflector target. However, this may have resulted from TTS measurement error and not actual latency. As angular velocity increased, position error increased, which may have had an effect on the calculated phase shift between the TTS and test fixture.

Distance from the TTS to the reflector target was shown to not have a significant effect on measurement error for most angular velocities tested. In the cases where there was a significant difference, that difference was less than 1.5 mm, which fulfills the order-of-magnitude accuracy requirement prescribed by the ISO 12188-1 standard. At higher angular velocities, the accuracy of the TTS is at a similar level to the static accuracy specified for most RTK GNSS devices. It is not known whether or not this level of accuracy will suffice for dynamic GNSS at angular velocities because no data on dynamic GNSS accuracy at high angular velocities has been published. Furthermore, no comparison has been made between angular velocity and actual speed. It has been assumed that constant change in direction is one of the worst case scenarios for the TTS because the system is marginally stable. Both the TTS, and the interpolation method used to calculate TTS position at PPS events are expected to perform better when travelling in straight paths.

CHAPTER 5: DYNAMIC GNSS ERROR MODELING

5.1 Introduction

Much of the research on dynamic GNSS receiver accuracy focuses on testing GNSS receivers under simulated agricultural operations. This includes features such as long parallel passes connected by curved sections. While this technique has proven useful for determining pass-to-pass accuracy and cross-track errors, it cannot be used directly to determine along-track error.

Along-track error is an important component in variable rate and precision planting applications where the position along a path is required for precise placement of a material. Determining along-track error from test fixtures prescribed by ISO 12188-1 can be difficult due to error in the measurement system of a test fixture.

Highly accurate RTK GNSS receivers are also difficult to test using the ISO 12188-1 standard because their accuracy exceeds the accuracy of the measurement systems used in the test fixture. The standard requires a position reference one order of magnitude more accurate than the device being tested. If an RTK GNSS receiver has an expected dynamic accuracy of a few centimeters, than the accuracy of the position reference must be a few millimeters.

Test fixtures such as the University of Kentucky GPS Test Track (Cole et al., 2004) were designed to primarily quantify off-track error. However, a well-defined path cannot be used to determine along-track error or exactly determine off-track error. Instead, off-track error is calculated as the shortest distance between a GNSS measurement and the

predetermined path and may include a vector component of along-track error. Incorporating a real-time position measurement system for determining position along a predetermined path has not been published in literature. This is likely due to the lack of viable position measurement systems that function at millimeter level accuracies over scales required by ISO 12188-1. More instrumentation is needed than just a physical test fixture to calculate along-track error and off-track error in curved segments.

In order to develop dynamic GNSS error models, a rotary test fixture can be used to get an indication of how the mean and standard deviation of error will change with respect to angular velocity as well as any internal settings of the GNSS receiver. A small rotary test fixture has the additional advantage of being physically rigid which improves the accuracy of the overall measurements system.

5.1.1 Objective

The objective of this study is to develop dynamic error models for a GNSS receiver for varying angular velocity and receiver settings, with an emphasis on the distribution of error.

5.2 Materials and Methods

5.2.1 Test Procedures

GNSS and rotary fixture data were recorded for angular velocities of 0.824, 1.423, 2.018, 2.618, and 3.222 rad/s at a 1 m radius. Each data set contained 4 hours of continuous measurements and was replicated three times. Trimble AgGPS 252 (V3.70) and Trimble MS990 GNSS (V4.40) receivers were configured to transmit the NMEA GPGGA (38400-8-N-1) string at a frequency of 1 Hz. The AgGPS 252 was mounted on the test

fixture and the MS990 was used as a static reference for identifying uncontrollable GNSS accuracy issues that might affect the experiment. Both receivers were configured to their factory default settings and received RTK corrections from a Trimble NetR8 GNSS Reference Receiver (V4.70) base station. The position of the base station was established by recording 4 hours of raw GNSS measurements. The raw measurements were post-processed using the Online Positioning User Service (OPUS). OPUS used the raw measurements in conjunction with the Kentucky Transportation Cabinet's (KTC) Continuously Operating Reference Station (CORS) network to determine the average position of the base station.

The PPS signal from the AgGPS 252 receiver was connected to the signal timing device and used to time stamp GNSS measurements with the relative fixture time and position. The AgGPS 252 included the ability to adjust the amount of position filtering between four levels; none, normal, high, and max. Each level was tested for this receiver to better understand how position filtering affects dynamic position measurement accuracy.

A PC program was written to interface with the test fixture and provide data logging capabilities (Figure 14, Appendix 3). GNSS measurements and corresponding test fixture time and position were stored in a CSV file for post processing. The date, time, and fixture setting for each test were used to construct the filename for each test (Figure 15).

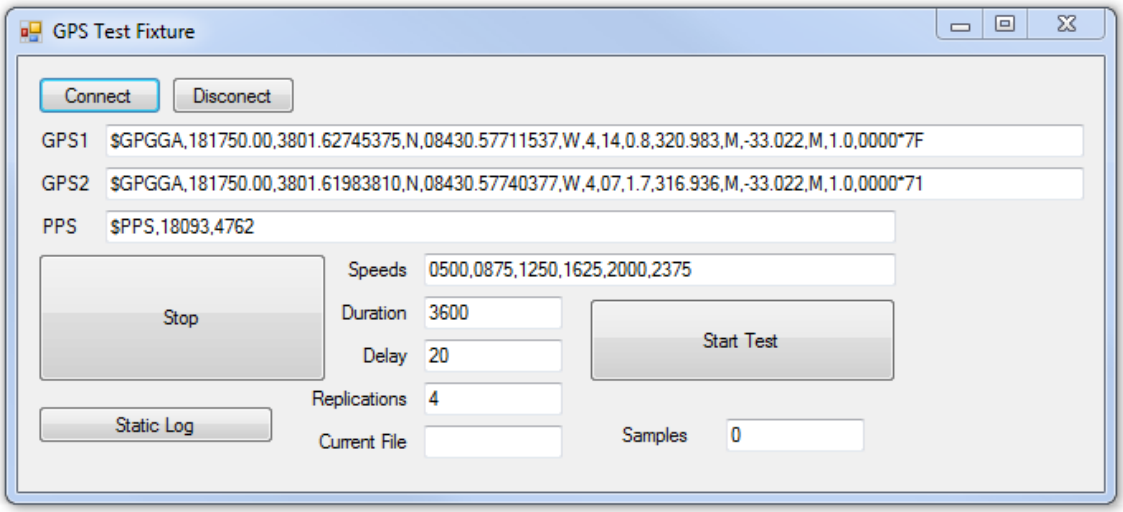


Figure 14: PC Program for Fixture Control and GNSS Data Logging

```
"mm-DD-YYYY_HH-MM-SS_FFFF.csv"

mm = Month, DD = Day, YYYY = Year,
HH = Hour,  MM = Minute, SS = Second,
FFFF = Fixture Setting
```

Figure 15: GNSS Data Filename Schema

5.2.2 Importing GNSS and Fixture Data into MATLAB for Processing

Data sets were imported into MATLAB for post processing. A script, *GNSSanaylis.m* (Appendix 8.1), was written to automate data importation and processing. The script used two strings to identify files that corresponded to a single test. The first was a folder string that contained the full path of the folder where all data files from test were stored. The second was a filename string which identified a file containing the names of the data

files to be processed (Figure 16). The filename string was always set to "FileList.txt" and the corresponding file was placed in the same folder as the data to be processed.

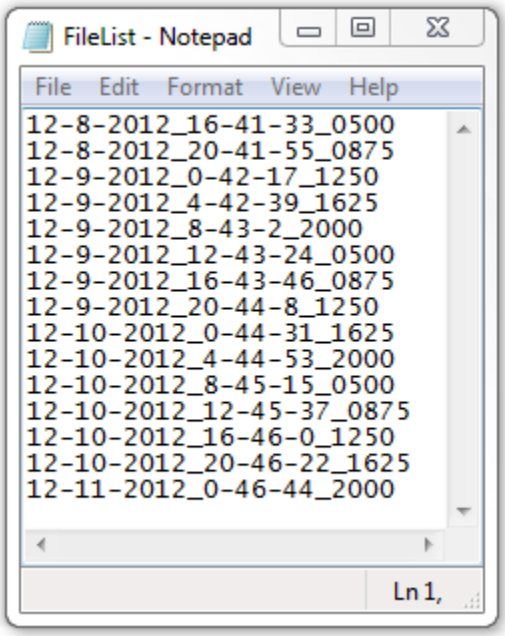


Figure 16: File List used for Importing GNSS and Fixture Data into MATLAB

The script iterated through each line in "FileList.txt" and concatenated the filenames with the folder string which resulted in a list of filenames with their full paths. Each data file was then imported using the built in *xlsread()* function and stored as an element in a cell-structure array.

5.2.3 Calculating X/Y GNSS Position Error

GNSS measurements were converted from the NMEA format to decimal degrees and then transformed into a local coordinate system centered about the test fixture using the ellipsoid transformation prescribed by ISO 12811-1 (ISO, 2010). A reference latitude of

38.02700334 degrees North and reference longitude of -84.50963172 degrees East was defined as the origin of the test fixture and determined by averaging the output of a RTK GPS receiver for 15 minutes mounted atop the center of rotation of the fixture. The transformation between radians and meters was 110996.411 m/rad and 87799.791 m/rad for latitude and longitude, respectively. Local coordinates were calculated by taking the difference between the GNSS receiver and the test fixture origin in radians, and multiplying by the corresponding transformation.

The optical encoder on the test fixture was used to determine the local X/Y coordinates of the test fixture. The encoder position A, which varied between 0 and 10,000, and ϕ, a constant for rotating the text fixture polar coordinate system to be in phase with the local coordinate system used for GNSS measurements were used to calculate the test fixture X/Y coordinates (Equation 17).

$$f_X = \cos\left(\frac{2\pi A}{10000} - \phi\right)$$
$$f_Y = \sin\left(\frac{2\pi A}{10000} - \phi\right) \qquad \text{Equation 17}$$

The X/Y coordinates for both the GNSS measurements and text fixture were plotted for each data set to visualize the application of the local transformation (Figure 17).

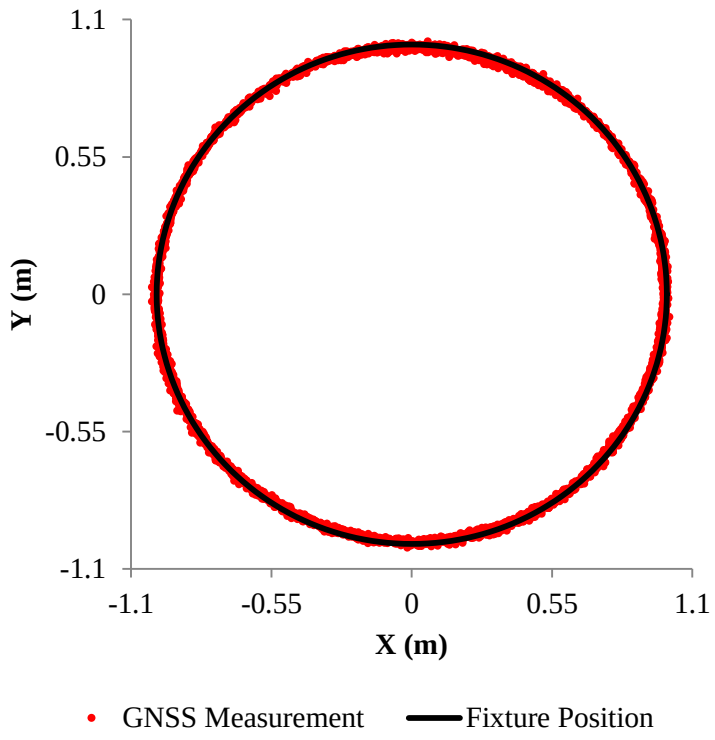


Figure 17: X/Y GNSS Measurements and Test Fixture Position Plot at 2.618 rad/s

Error in the X/Y cardinal directions were calculated by taking the difference between a GNSS measurement and the test fixture measurement for each sample (Figure 18). In some cases the X/Y error exhibited a directional dependence which was a result of the individual X and Y error distributions having a different mean and standard deviation of error.

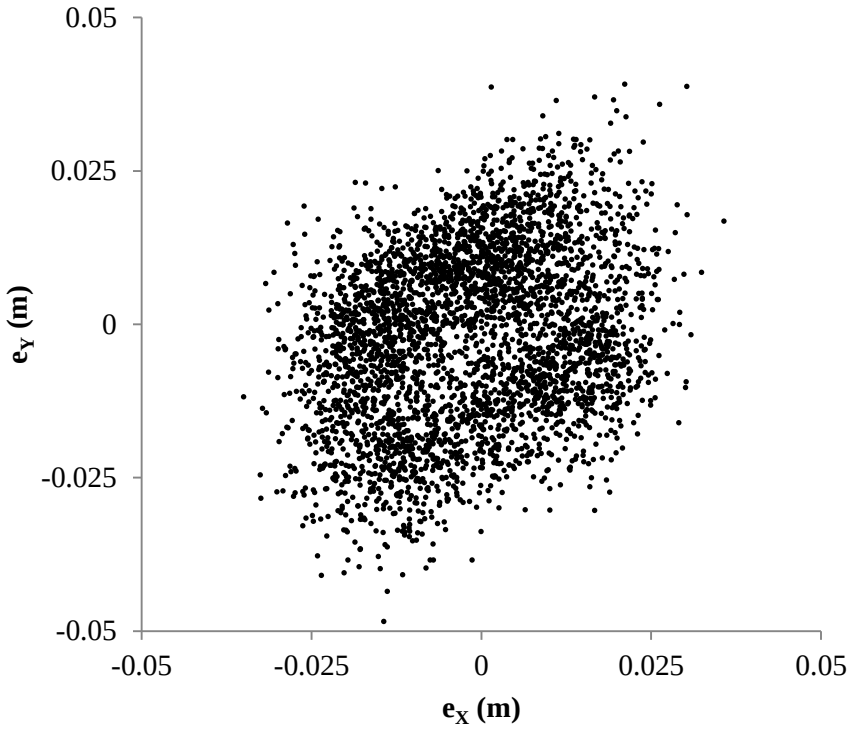


Figure 18: X/Y Error Plot at 2.62 rad/s

5.2.5 Calculating Along- and Off-Track Error

Off- and along-track errors relate position error with the direction of a vehicle in terms of the ability to track a predetermined path and the ability to perform and operation along the path, respectively. Along- and off-track error can be calculated directly from the X/Y errors in the cardinal directions by rotating the error about the reference point (Figure 19).

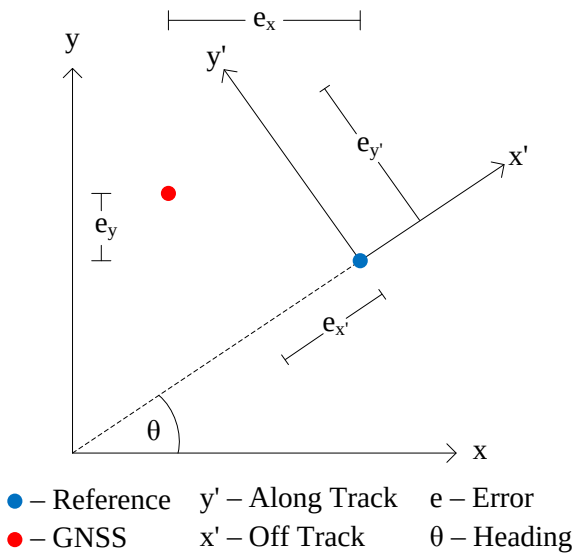


Figure 19: Cardinal and Reference Coordinate Systems

The amount of rotation is determined by the heading of the reference point relative to the original coordinate system. Because the rotary test fixture was centered at the coordinate (0,0), the heading at any reference measurement is the four-quadrant arctangent of the ratio of the y and x coordinates (Equation 18). The along- and off-track errors are calculated using a rotation transformation (Equation 19) and an example of the results are show in Figure 20.

$$\theta[n] = \tan^{-1}\left(\frac{y[n]}{x[n]}\right) \qquad \text{Equation 18}$$

$$\begin{bmatrix} e_{x'}[n] \\ e_{y'}[n] \end{bmatrix} = \begin{bmatrix} -\sin(-\theta[n]) & cos(-\theta[n]) \\ cos(-\theta[n]) & \sin(-\theta[n]) \end{bmatrix} \begin{bmatrix} e_x[n] \\ e_y[n] \end{bmatrix} \qquad \text{Equation 19}$$

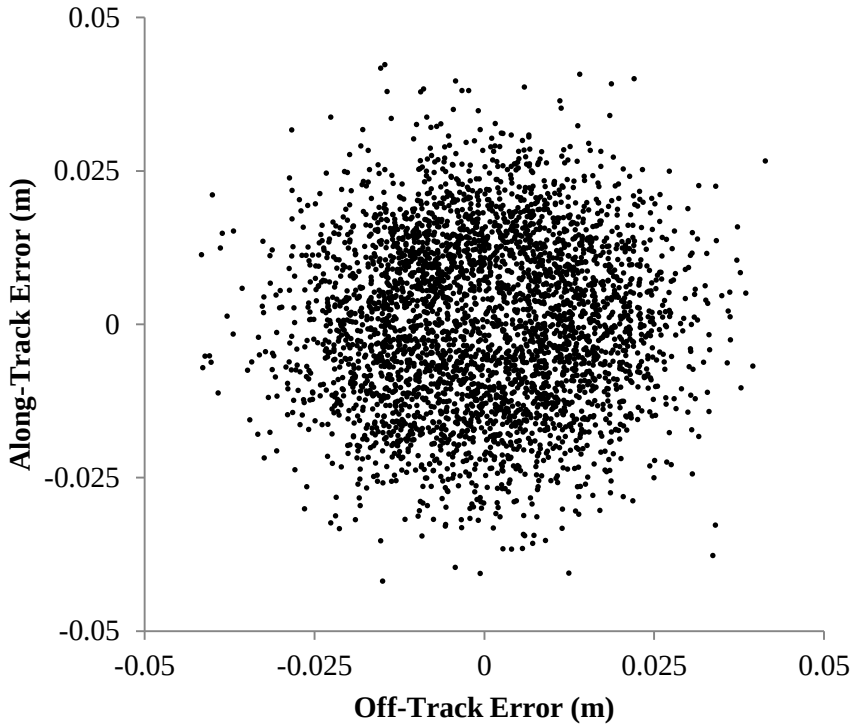


Figure 20: Along-/Off-Track Error Plot at 2.62 rad/s

Note that the direction dependence for the data in Figure 18 is no longer present in the same data that has been rotated about the receiver in Figure 20.

5.3 Results and Discussion

5.3.1 X/Y Position Error Results

The standard deviation of error in the X and Y cardinal directions increased with respect to angular velocity (

Table 9 through Table 12). The amount of filtering had a direct effect on the the standard deviation of error. The standard deviation of error steadily increased between no filter

and max filter settings on the receiver. The mean error was close to zero in every instance.

Table 9: X/Y Mean Error and Standard Deviation of Error – No Filter

rad/s	0.82		1.42		2.02		2.62		3.22	
Error	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
Mean	-0.002	-0.002	0.000	-0.001	0.000	-0.001	0.000	-0.001	0.000	-0.001
StDev	0.014	0.016	0.014	0.014	0.014	0.014	0.015	0.015	0.019	0.020

Table 10: X/Y Mean Error and Standard Deviation of Error – Normal Filter

rad/s	0.82		1.42		2.02		2.62		3.22	
Error	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
Mean	0.000	-0.003	-0.001	-0.001	0.000	-0.001	0.000	-0.001	0.000	-0.001
StDev	0.016	0.017	0.016	0.015	0.016	0.016	0.019	0.019	0.022	0.023

Table 11: X/Y Mean Error and Standard Deviation of Error – High Filter

rad/s	0.82		1.42		2.02		2.62		3.22	
Error	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
Mean	-0.001	-0.001	0.000	-0.001	0.000	-0.001	0.000	-0.001	0.000	-0.001
StDev	0.023	0.025	0.025	0.024	0.021	0.020	0.023	0.023	0.026	0.027

Table 12: X/Y Mean Error and Standard Deviation of Error – Max Filter

rad/s	0.82		1.42		2.02		2.62		3.22	
Error	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
Mean	-0.001	-0.002	0.000	-0.001	0.000	-0.001	0.000	-0.001	0.000	-0.001
StDev	0.024	0.026	0.024	0.022	0.024	0.023	0.025	0.025	0.028	0.029

The level of filtering changed the shape of the distribution of error. Discrete probability density functions were computed for each angular velocity and filter level. Lower filter levels generally produced normal distributions (Figure 21) for all angular velocities while

higher filter levels produced uniform distributions (Figure 22). A complete set of the discrete probability density functions in the X/Y directions can be found in Appendices 4.1 through 4.4.

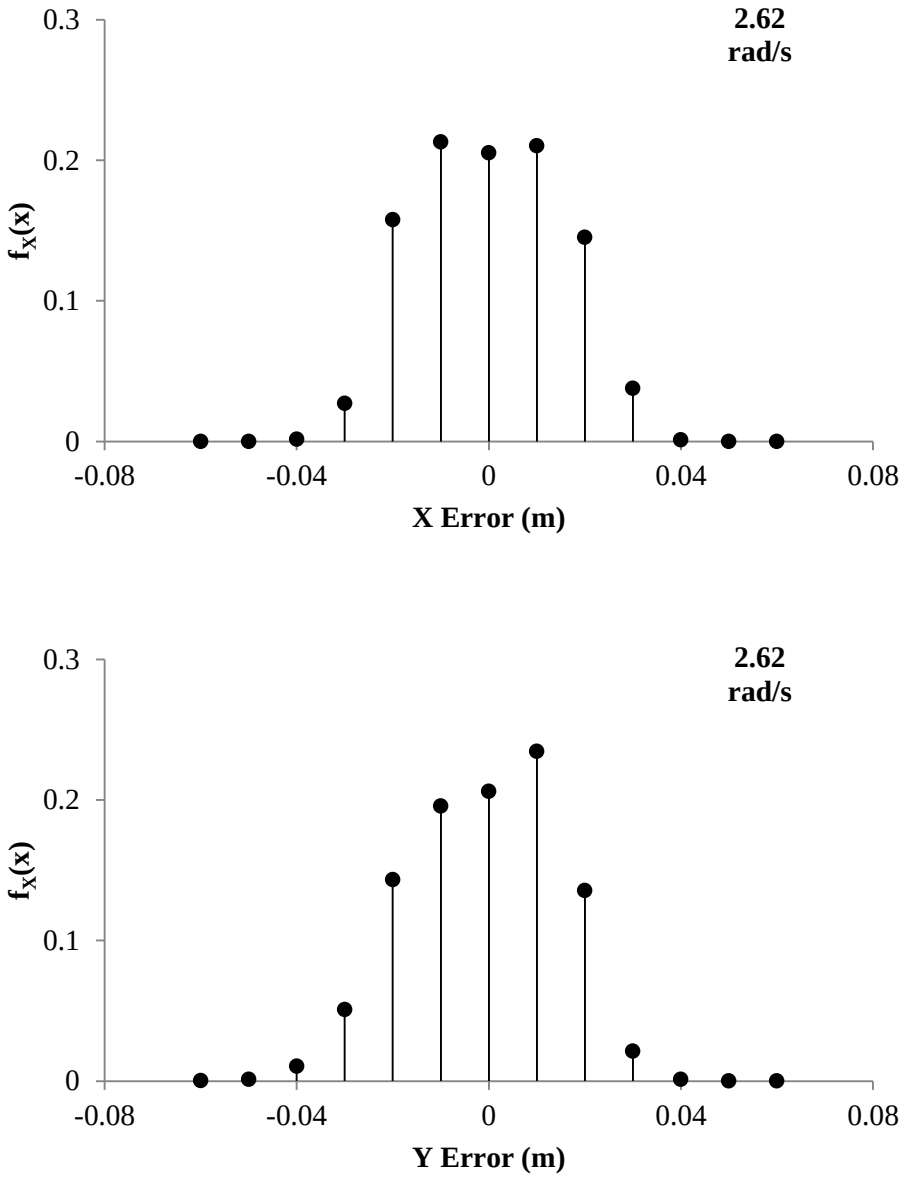


Figure 21: X/Y Error Discrete Probability Density Functions at 2.62 rad/s – No Filter

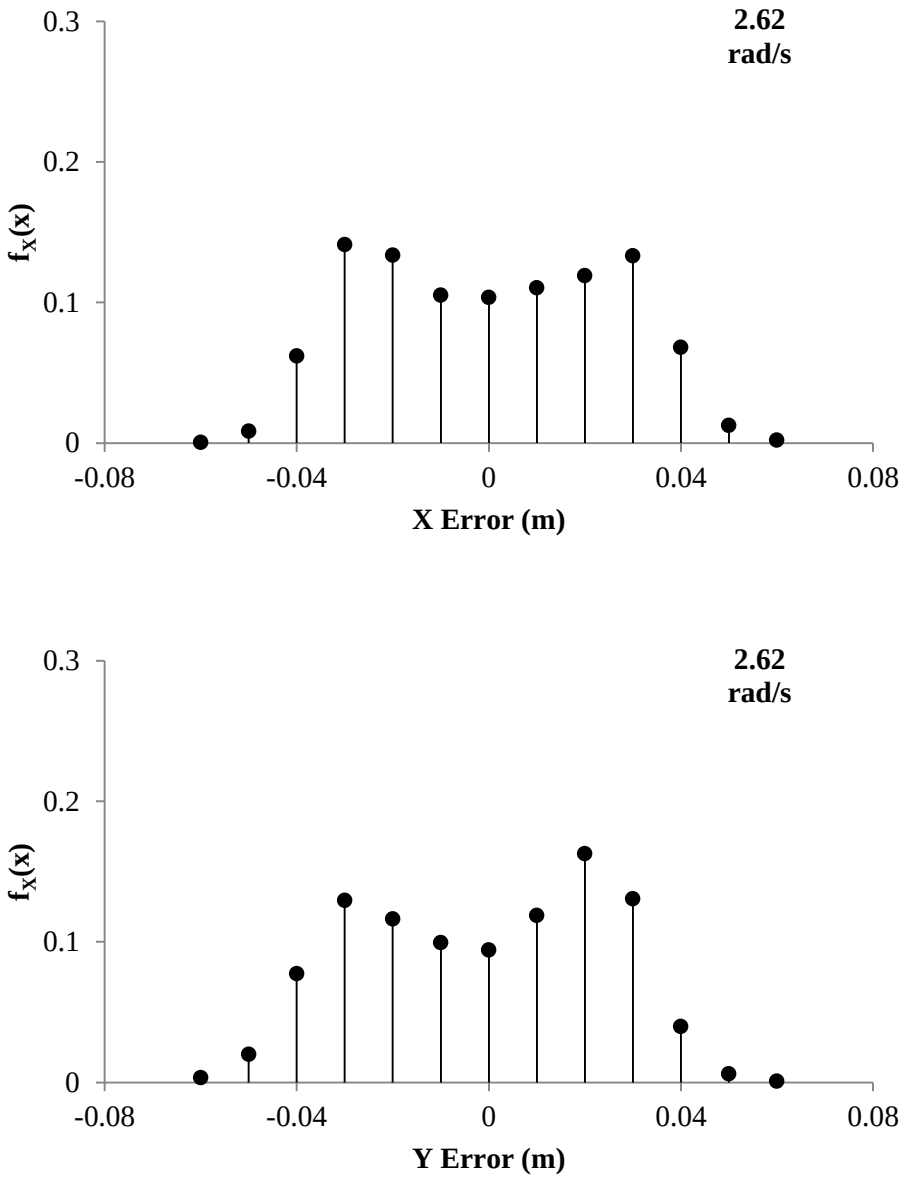


Figure 22: X/Y Error Discrete Probability Density Functions at 2.62 rad/s – Max Filter

5.3.2 Along- and Off-Track Position Error Results

The trends in along- and off-track error were similar to what was observed in the X and Y cardinal directions. The standard deviation of error in the along- and off-track directions increased with respect to angular velocity (Table 13 through Table 16). The amount of

filtering had a direct effect on the the standard deviation of error. The standard deviation of error steadily increased between no filter and max filter settings on the receiver. Additionally, there was no significant difference between the mean error (t-test, $p = 0.05$) and standard deviation of error (F-test, $p = 0.05$) in the along- and off-track directions with the exception of a single test at 1.42 rad/s and high filtering (Figure 23 and Figure 24). The mean error was less than 1 mm in every instance.

Table 13: Along-/Off-Track Mean Error and Standard Deviation of Error – No Filter

rad/s	**0.82**		**1.42**		**2.02**		**2.62**		**3.22**	
Error	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**
Mean	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
StDev	0.015	0.015	0.014	0.014	0.014	0.014	0.015	0.015	0.019	0.029

Table 14: Along-/Off-Track Mean Error and Standard Deviation of Error – Normal Filter

rad/s	**0.82**		**1.42**		**2.02**		**2.62**		**3.22**	
Error	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**
Mean	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
StDev	0.016	0.016	0.015	0.015	0.016	0.016	0.019	0.019	0.022	0.022

Table 15: Along-/Off-Track Mean Error and Standard Deviation of Error – High Filter

rad/s	**0.82**		**1.42**		**2.02**		**2.62**		**3.22**	
Error	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**
Mean	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
StDev	0.024	0.024	0.024	0.026	0.021	0.021	0.023	0.023	0.026	0.026

Table 16: Along-/Off-Track Mean Error and Standard Deviation of Error – Max Filter

rad/s	**0.82**		**1.42**		**2.02**		**2.62**		**3.22**	
Error	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**	**Off**	**Along**
Mean	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
StDev	0.025	0.025	0.023	0.023	0.023	0.023	0.025	0.025	0.028	0.029

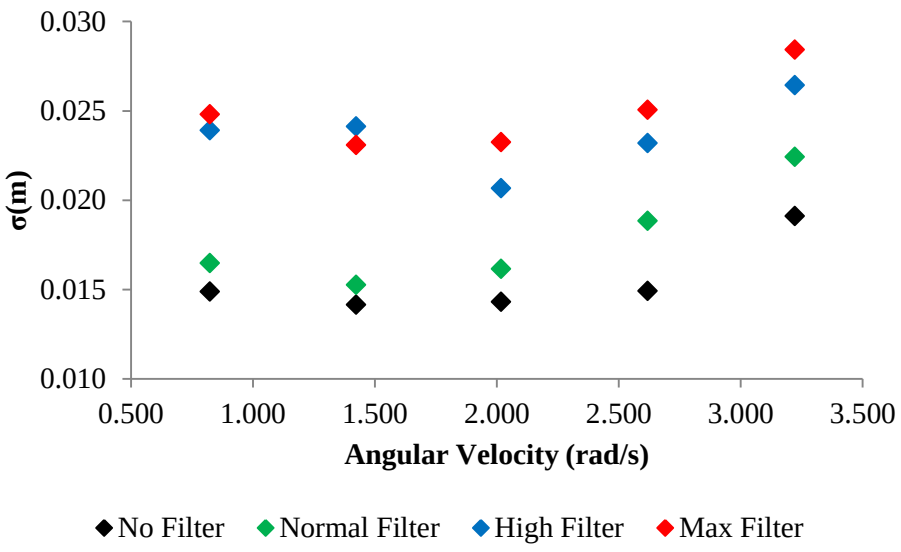


Figure 23: Off-Track Standard Deviation of Error

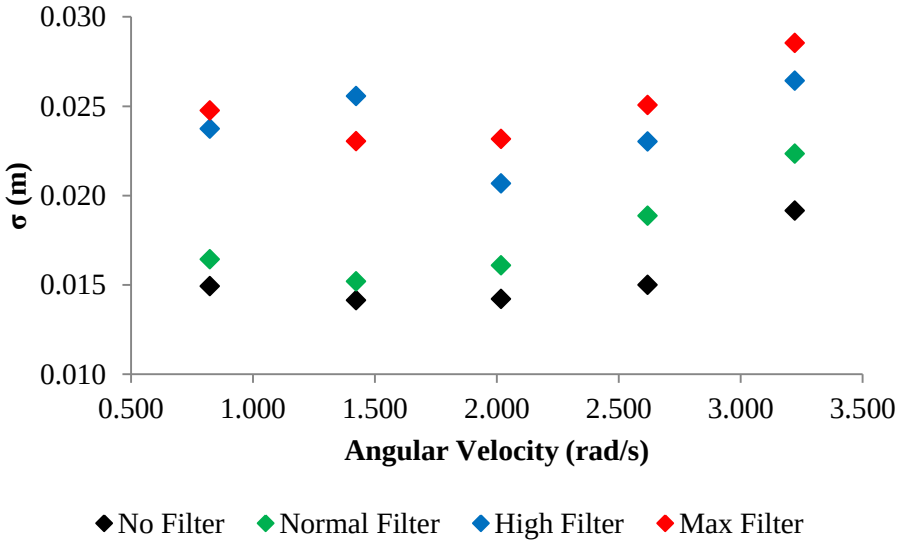


Figure 24: Along-Track Standard Deviation of Error

Due to the similarities in mean error and standard deviation of error, the along- and off-track errors are presented together as a single discrete probility distribution function for each angular velocity and filter level. Lower filter levels generally produced normal distributions (Figure 25) for all angular velocities while higher filter levels produced uniform distributions (Figure 26). A complete set of the discrete probability density functions in the along- and off-track directions can be found in Appendices 5.1 through 5.4.

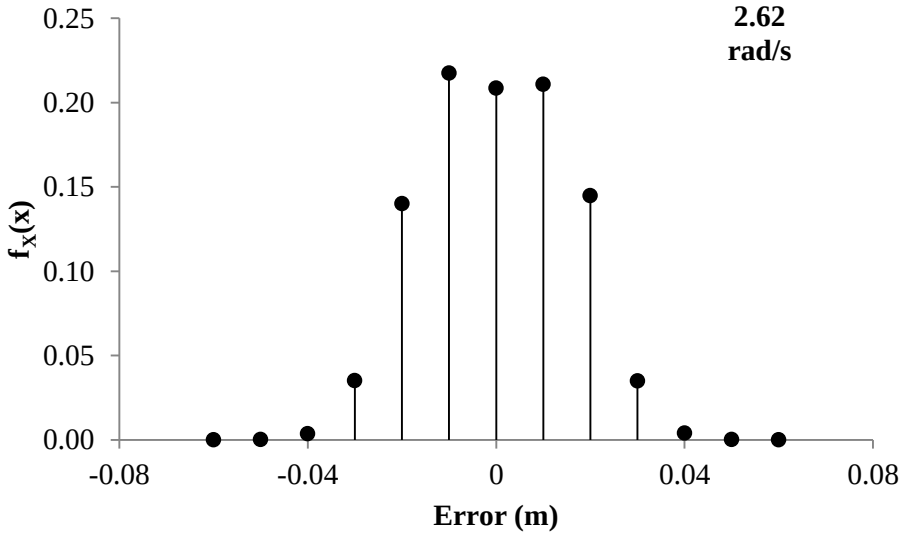


Figure 25: Along/Off-Track Error Discrete Probability Density Functions at 2.62 rad/s – No Filter

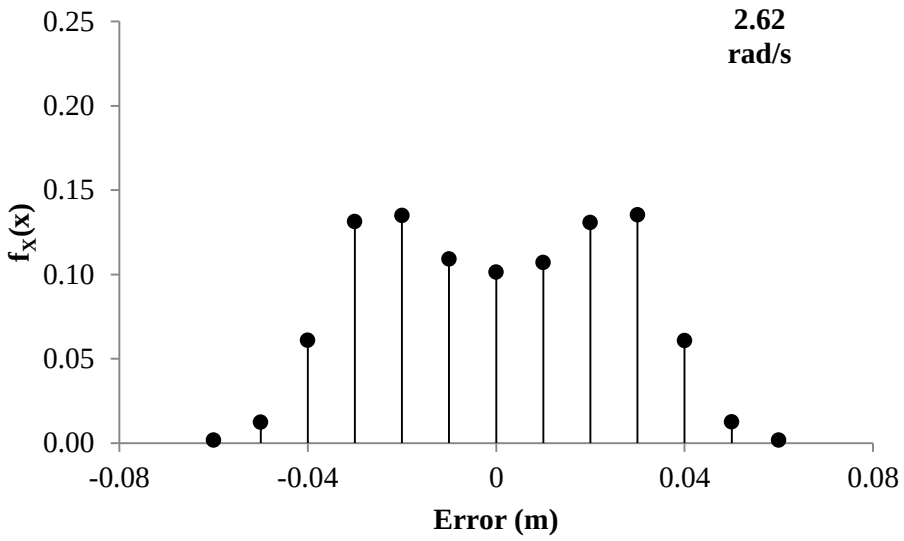


Figure 26: Along/Off-Track Error Discrete Probability Density Functions at 2.62 rad/s – Max Filter

5.4 Conclusions

The angular velocity and filter level of an AgGPS 252 GPS receiver was shown to have no significant effect on the average error over a 20 hour period. However, the standard deviation of error tended to increase at higher angular velocities. Increases in the filter level also resulted in larger standard deviations in error. The difference in standard deviation between the best and worst tests at each filter level was nearly two-fold. This indicates that GNSS receiver dynamics may play an important role in estimating and measuring field performance of agricultural machinery that utilize satellite based positioning systems.

Along- and off-track errors were shown to be remarkably similar when testing under steady-state sinusoidal conditions. This trend was not expected, as increasing angular

velocities and filter levels were hypothesized to result in a phase shift and/or a change in the magnitude of error. Neither was observed in any test. The only consistent significant difference in results between testing parameters was the standard deviation of error.

Given the wide range of distributions, and their shapes, it can be concluded that a mean and standard deviation alone is not enough to fully define dynamic GNSS error. Distributions tended to fall somewhere between normal and uniform. A discrete probability density function for each scenario is one way to get around the issue of the non-stationary distributions between testing parameters. However, the changes in the distributions of error are small when compared to the scale of field operations. Differences in standard deviations on the sub-millimeter level alone are not likely to result in any measureable performance difference in field operations. More research is needed to understand how the variability in GNSS measurements translates into overall machine performance and ultimately field performance.

CHAPTER 6: APPLYING A DYNAMIC GNSS ERROR MODEL

6.1 Introduction

6.1.1 Background

Precision agriculture techniques are adopted based on the potential economic benefit of reducing inputs or increasing production. Numerous studies have shown the economic benefit of adopting these practices (Watkins et al., 1998); (Silva et al., 2007); (Thrikawala et al., 1998); and (Robertson et al., 2009). Some of these techniques rely on GNSS receivers for navigation while others do not. Technology including variable-rate spraying, section control spraying, strip till fertilizing, and precision planting all use GNSS receivers for position, speed, and orientation measurements. Broadcast fertilizing and harvesting commonly use GNSS receivers for navigation and data collection but the error in position measurement is relatively small when compared to the varibilitly of the process. The ammount of potential economic benefit from knowing how much error is present in position measurement will depend on the application. Predictions of overlaps and skips and application rate can be refined to include position error.

One way to calculate the direct economic benefit of choosing between GNSS systems of varying accuracies would be to determine if there is any measureable difference in performance at the scale that is required to complete the management operation. If there is no advantage to using RTK level corrections over a cheaper method for a given application, then the additional capital cost can be viewed as directly reducing profit. Setting up a local base station, data radios, and unlocking the RTK capability of a GNSS

can be cost prohibitive. This may make sense economically for a large farm if multiple machines are running simultaneously because sattelite-based corrections require annual suscriptions for each device that can quicky add up to more than a ground based solution. Smaller farms are more likely to have less capital available for investing in top-of-the-line navigation equipment, and yet the performance may not differ significantly when using lower cost sattelite based corrections.

6.1.2 Visualizing GNSS Accuracy

Visualizing the standard deviation of error as a probability density function illustrates the difference between the multiple accuracy specifications when using different DGPS methods (Figure 27). A "good" estimate of the standard deviation of error is 2.54 cm, an "average" estimate of the standard deviation of error is 10.2 cm, and a "poor" estimate of the standard deviation of error is 1 m. These specifications were derived from a single Trimble AgGPS 262 receiver under RTK, Omnistar HP, and Omnistar VBS corrections, respectively. The distribution of error is assumed to be Gaussian with zero mean. Realistically, there would be an offset in the mean error when switching sources of DGPS correction. Note that the distributions shown in Figure 27 are continuous for illustration purposes. The actual distributions used in the dynamic GNSS error model are discrete probability density functions.

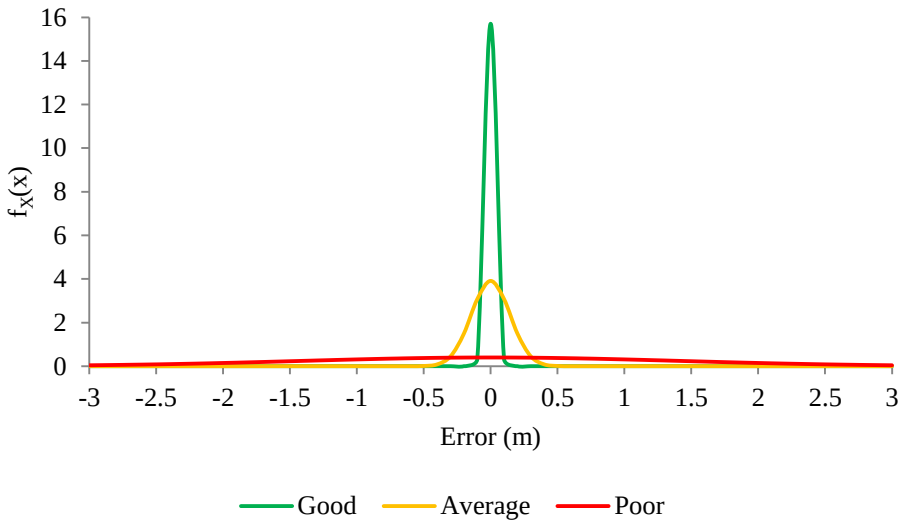


Figure 27: Good, Average, and Poor Standard Deviation of Error for GNSS Receivers

Preliminary data has shown that there is change in the distribution of error with respect to the mean and standard deviation when comparing the correction method and between static and dynamic tests (Table 17) (Sama et al., 2009). One way to interpret the data is using the criteria shown in Figure 27. The GNSS receiver had a "good" standard deviation of error under static RTK conditions but became "average" once the receiver was placed in motion.

Table 17: Error Distribution for a Static and Dynamic GNSS Receiver

	Static RTK	**Dynamic RTK**	**Static WAAS**	**Dynamic WAAS**
|x| (m)	0.016	0.065	0.776	0.258
σ (m)	0.003	0.131	0.091	0.070

6.1.3 Resolution of Computation

To emphasize the potential for computational error when applying GNSS error models, a hypothetical example will be explored. This example illustrates that the effect of spatial

resolution on off-rate and off-target application error estimations due to overlaps and skips in a field will change when the error distribution of the GNSS receiver is taken into account. Off-rate is commonly referred to as an application rate that is outside +/- 10% of the target rate (Luck et al., 2011). Figure 28 shows how a 25.4 cm (10 in) skipped or overlapped area looks for each estimate standard deviation of error when compared to theoretical skipped area of 25.4 cm. An application rate of 1 corresponds to 100% and 2 corresponds to 200%.

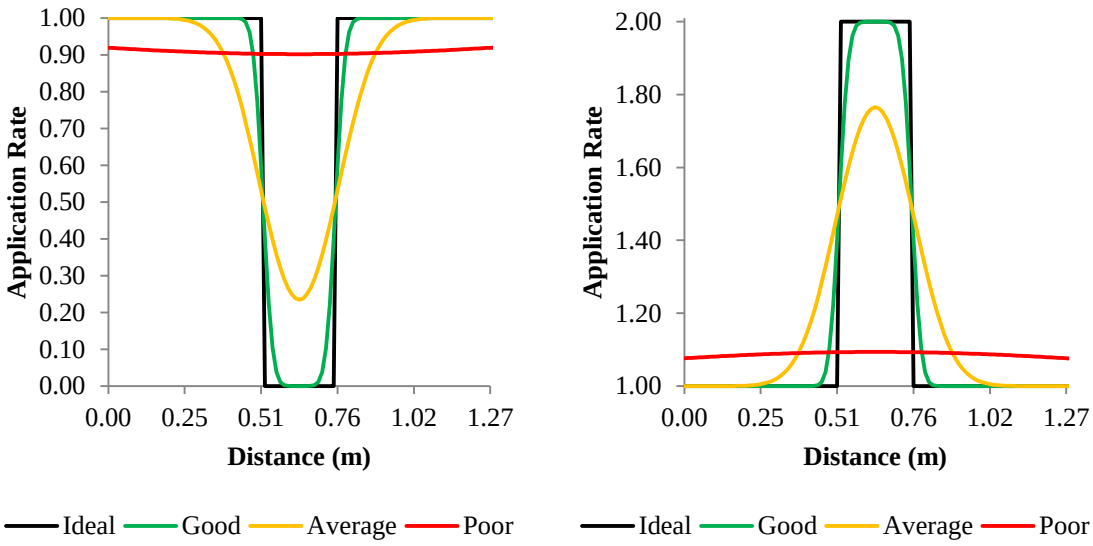


Figure 28: Application Error for Skipped (left) and Overlapped (right) Coverage

Under the ideal scenario, there are 9 samples out of 50 where the application rate was off-rate when the spatial resolution is 2.45 cm and 19 samples out of 100 when the spatial resolution is 1.27 cm. When a “good”, “average”, and “poor” estimate of the standard devation of error was included, the number of off-rate samples inscreased to 11, 19, and 0

for a spatial resolution of 2.54 cm. The results changed to 23, 39, and 0 samples that were off-rate when the spatial resolution was doubled. A summary of the percent difference from the ideal off-rate calculations is shown in Table 18. An imporant observation is that theoretical off-rate error was underestimated when the error due to the GNSS receiver was ignored. The spatial resolution at which off-rate error was calculated also had an effect on the results. More investigation is needed to determine how the spatial resolution should relate to the precision of the measurement being made. In the case when a "Poor" quality estimate for the standard deviation of error is used, the position error is so large with respect to the process that an off-rate calculation should not be made at either spatial resolution.

Table 18: Percent Change from an Ideal Off-Rate Calculation

Spatial Resolution (cm)	"Good"	"Average"	"Poor"
2.54	22.2%	111%	-
1.27	21.1%	105%	-

Another way to interpret the results is to use a binary threshold for deciding whether or not material was applied in an area. This type of off-target comparison is more in line with what is commonly considered in terms of overlap in a field (Luck et al., 2010). If the decision boundary is set to 0.5 then both the "good" and "average" standard deviations of error do not change the area at either spatial resolution while the "poor" standard deviation of error inteferes with the ability to determine where a skipped or overlapped area occurred. In other words, a GNSS receiver with and specified accuracy

of +/- 1 m is not adequate for determing skipped and overlapped areas 25.4 cm in width. This kind of analysis might prove to be useful in determining what the minimum level of GNSS accuracy needs to be in order to calculate overlapped and skipped areas at a defined level of certainty.

6.1.4 How Much Accuracy is Needed?

The level of accuracy required for a given agricultural operation has not been presented in very consistent manner throughout literature. One problem is that there historically hasn't been a standard method for expressing dynamic GNSS accuracy, and then relating that accuracy to the expected performance of an agricultural operation. Manufactures sometimes express relevant terms such as pass-to-pass accuracy of the GNSS receiver but the specifications should only be used as a loose guideline unless they indicate the method used to determine the results and whether or not they respresnet absolute or relative error. It would be extremely useful for a farmer to be able to look up what type of system is required to achieve a required level of performace for a specific application. Refering to the hypothetical example in *6.1.3*, if the goal is to use a GNSS based navigation system to minimize off-target applications due to overlaps and skips greater than 25.4 cm (10 in), then there is no measurable advantage to using RTK over Omnistar HP. However, if the goal is to minimize off-rate applications to within specifications between passes, RTK outperforms Omnistar HP by approximately 80%. These calculations are based solely on static accuracy specifications, which as previously stated are better than dynamic accuracy specifications.

At low spatial resolution processes, such as yield monitoring or broadcast fertilizing, the ammount of averaging or filtering throughout the entire process is so large that

estimating the additional error due to dynamic GNSS accuracy is not practical. Therefore, the primary focus on feasibility in this chapter will be with section control and variable-rate technologies where precise placement of material inside a boundary is critical to efficiency. As more and more manufacturers move to individual nozzle control in their sprayers, the spatial resolution of actuation approaches the nozzle spacing across the boom and depends on the vehicle speed and control update rate in the direction of travel.

6.1.5 Applying a GNSS Error Model using Convolution

Assuming GNSS receivers exhibit random fluctuations in position output with respect to time, the two dimensional position can be modeled as a multivariate random process. Determining the appropriate model to use requires testing the receiver in a repetitive method to better understand the distribution of error with respect to parameters which have an effect on error. The simplest method assumes that the random process is stationary, or has a constant distribution regardless of varying dynamics and latency. As such, the direction and speed of an agricultural process will not change the estimated position error. Incorporating this error into an input map can be performed by applying a convolution summation between the recorded input map and estimated position error.

If, for example, the position error of a typical GNSS receiver happens to exhibit a bivariate normal distribution with no cross-correlation between the *x* and *y* directions, then the model for error would be as follows.

$$f_{X,Y}[x,y] = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{1}{2}\left(\frac{x-\mu_x}{\sigma_x}\right)^2\left(\frac{y-\mu_y}{\sigma_y}\right)^2}$$

where,

$\mu_x =$ mean error in the x-direction

$\mu_y =$ mean error in the y-direction

$\sigma_x =$ standard deviation of error in the x-direction

$\sigma_y =$ standard deviation of error in the y-direction

The random error can be visualized in either two or three dimensions (Figure 29). As with all discrete probability density functions, the sum off all of the "pixels" in the 2D model, or the area underneath the surface of the 3D model is exactly one as long as the extent of the model is large enough to encompass all values. Since a Gaussian distribution is defined for all values, it must be clipped and normalized to ensure a proper summation.

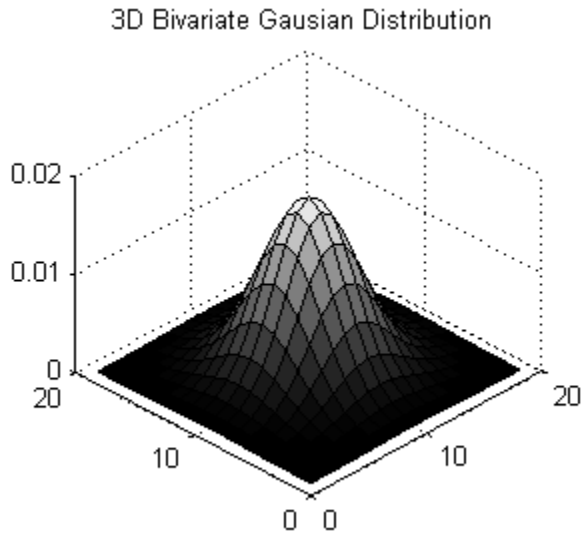


Figure 29: Bivariate Gaussian Distribution

If the cumulative distribution did not add up to one for a certain extent, then incorporating position error into an input map would produce a difference in the total amount of material input to a field. Ensuring that the sum of the pixels adds to one requires controlling the extent of the error distribution to a reasonable amount if the dynamic GNSS error model is continuous or discretizing the dynamic GNSS error model prior to application. Larger extents ensure less computational error but increase the time required to perform the computation. The greater the standard deviation in each direction, the larger the extent required. Taking these requirements into consideration, convolving the prescribed input map with the GNSS receiver error distribution will simply redistribute the location of inputs to reflect the uncertainty in position (Figure 30).

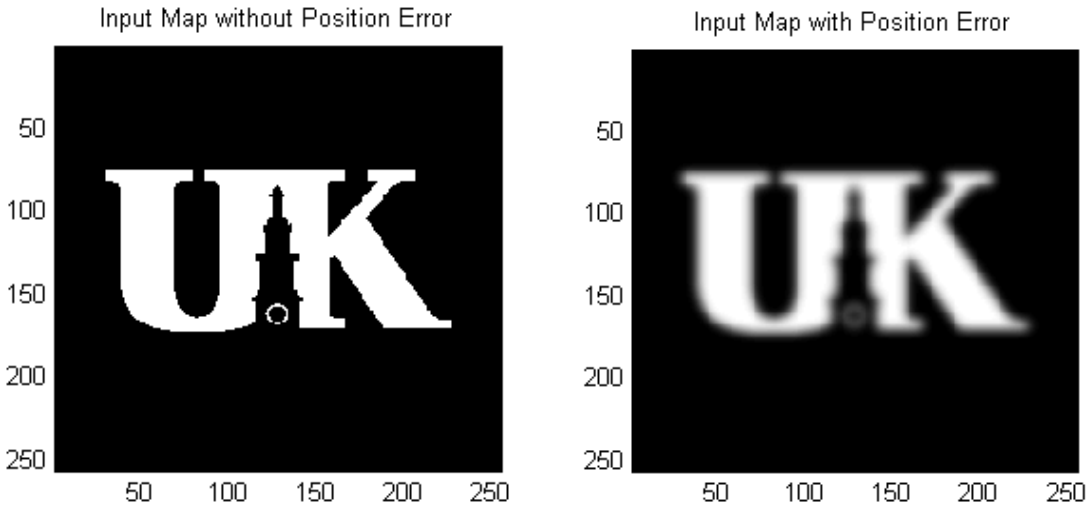


Figure 30: Convolution of an Input Map with a Position Error Distribution

The most noticeable effect in the output is the loss of sharp transitions. Binary field inputs such as turning sprayer nozzles on and off no longer produce binary transitions as the exact location where they occur is no longer assumed to be precisely known. These

transitions are of great importance when calculating areas where application rates deviate. Note that the predicted field inputs only change around the boarders. This property may prove useful to reduce the complexity of estimating application error when incorporating a dynamic GNSS receiver error model.

Assuming the position error distribution of a GNSS receiver to be a stationary process is convenient for calculation purposes but may not correctly reflect the true nature of the error. More complicated models will include parameters such as latency, which is manifested as a shift in the mean error depending on the direction of travel. Allowing the error distribution model to change with respect to direction required the convolution between an input map and an error distribution to be divided into separate intervals for each unique error distribution. To achieve this using a discrete method, the convolution sum must be manually compiled point by point, taking into consideration the error distribution at that location.

For example, if the position error of a typical GNSS receiver exhibited a bivariate normal distribution with no cross-correlation between the x and y directions and a shifted mean based upon the speed and direction of travel due to GNSS receiver latency, then the input map corrected for position error will exhibit a similar shift (Figure 31).

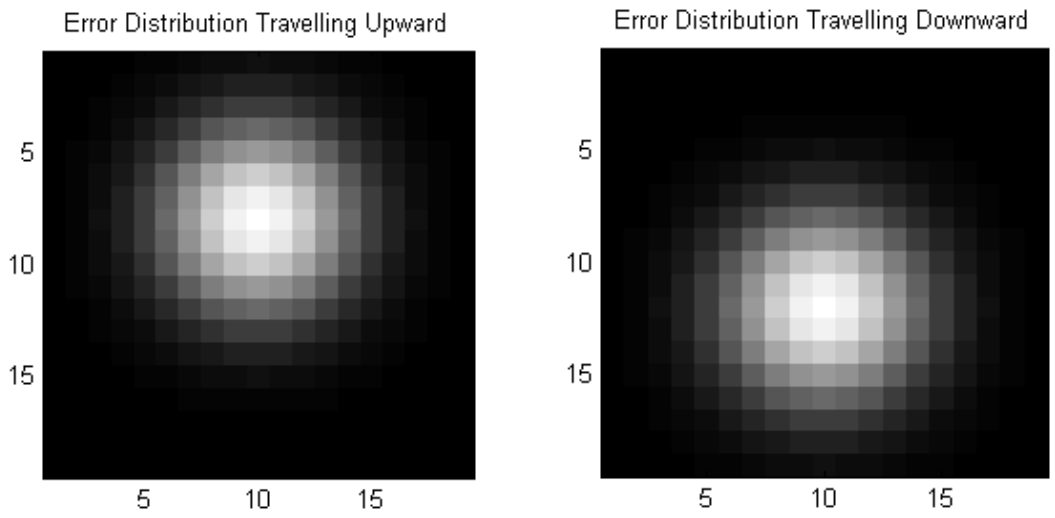


Figure 31: Directional Error Distributions

Applying the directional filter to the original input map redistributed the location of inputs with respect to the direction of travel (Figure 32).

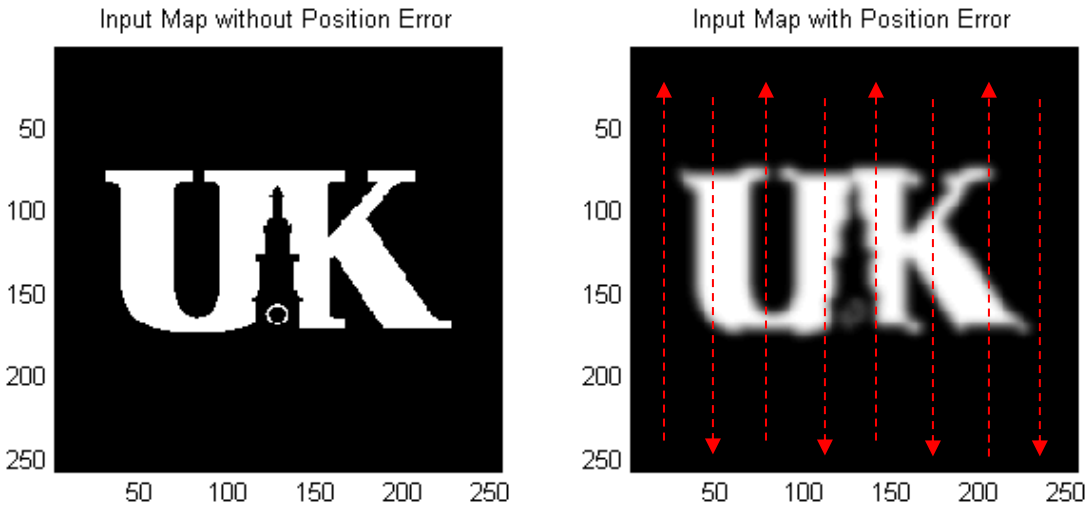


Figure 32: Visualizing Directional Error

Once again, the predicted field inputs only change around the borders. In fact, the amount of material that is applied outside of the field boundary due to latency in the GNSS error model approaches the same value as the amount of material applied inside the field boundary when the size of the field increases or the width of each pass decreases. Therefore, the focus on applying a dynamic GNSS error model in actual applications should be placed on the boundary or perimeter of the field.

6.2 Materials and Methods

Square and circle field shapes were used to predict the amount of off-rate error due to section control around the borders of a field when using three different levels of GNSS accuracy. A calibration curve was generated for each level of GNSS accuracy that took the perimeter to area (P/A) ratio as an input and output the expected amount of off-rate error around the border. GNSS accuracy levels of 2.54 cm, 10.2 cm, and 100 cm with bivariate normal distributions were used to represent "good", "average", and "poor" error distributions, respectively (Figure 33).

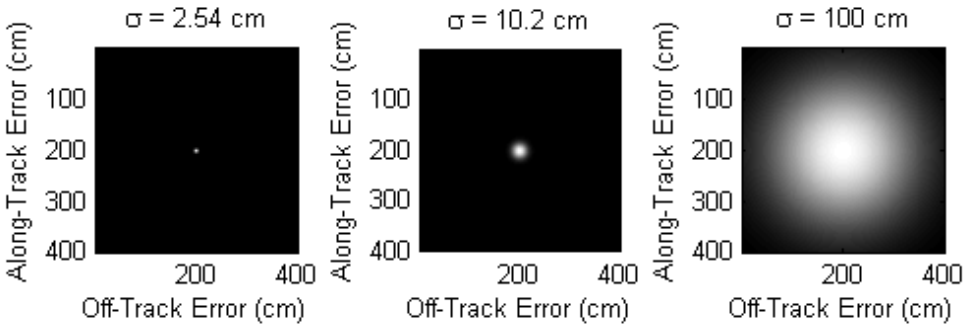


Figure 33: GNSS Error Distributions

The off-rate error for ten square and ten circle field shapes with lengths and diameters ranging from 1 to 10 m was determined at a resolution of 1 cm by convolving the error distribution with the field shape (Figure 34). The application maps with GNSS error were clipped to remove applications outside of the field boundary. This resulted in the expected application rate inside the field boundary due to GNSS error (Figure 35).

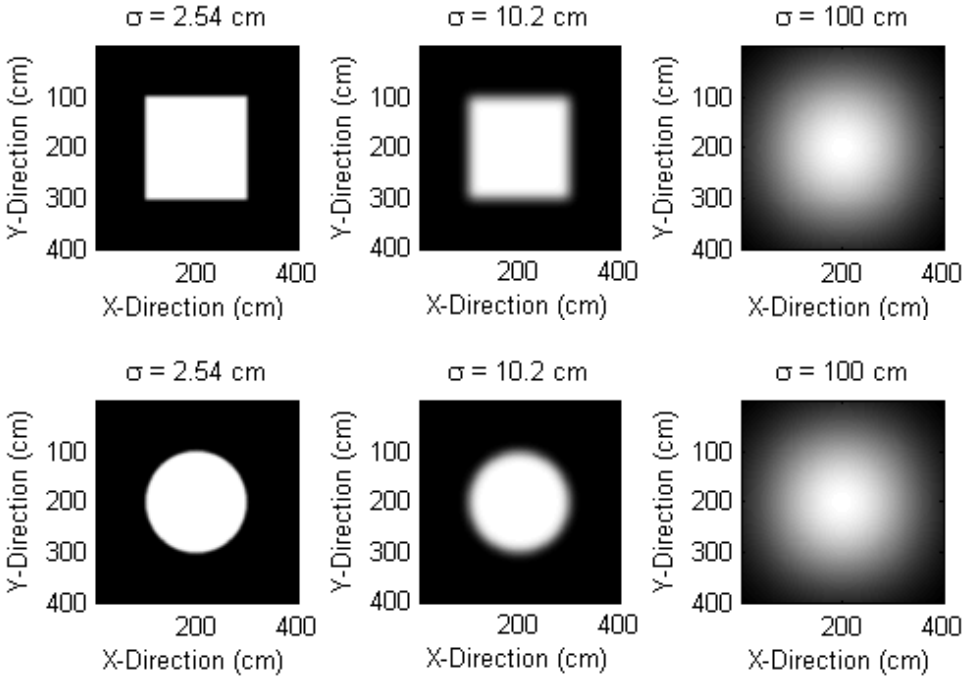


Figure 34: Application Error in Square and Circle Fields

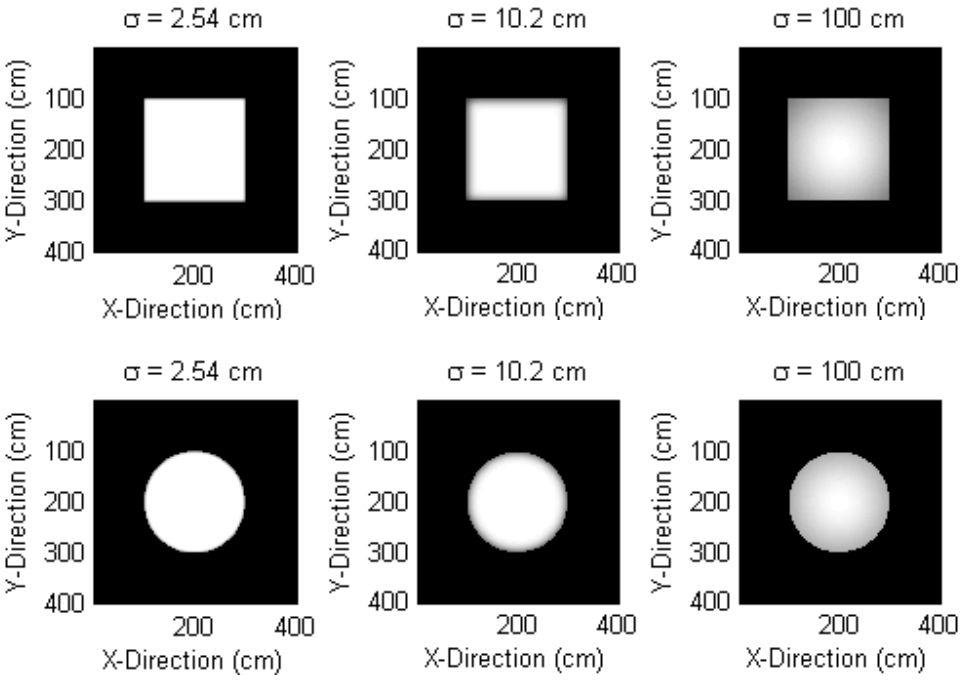


Figure 35: Application Error Inside Square and Circle Field Boundaries

Off-rate error was calculated by counting the number of square centimeter pixels that deviated from the target rate by more than 10%. Off-rate errors were tabulated along with field area, perimeter, and P/A ratio to determine the relationship between the P/A ratio and the expected off-rate error. Appendix 9.1 contains the script used to estimate off-rate application errors at the field boundary based on the P/A ratio.

Boundaries from nine fields were used to represent typical farms in Kentucky. The fields were previously used by Zandonadi et al. (2011) to determine off-target application areas. They ranged in area between 9.4 ha and 39.6 ha (Figure 36).

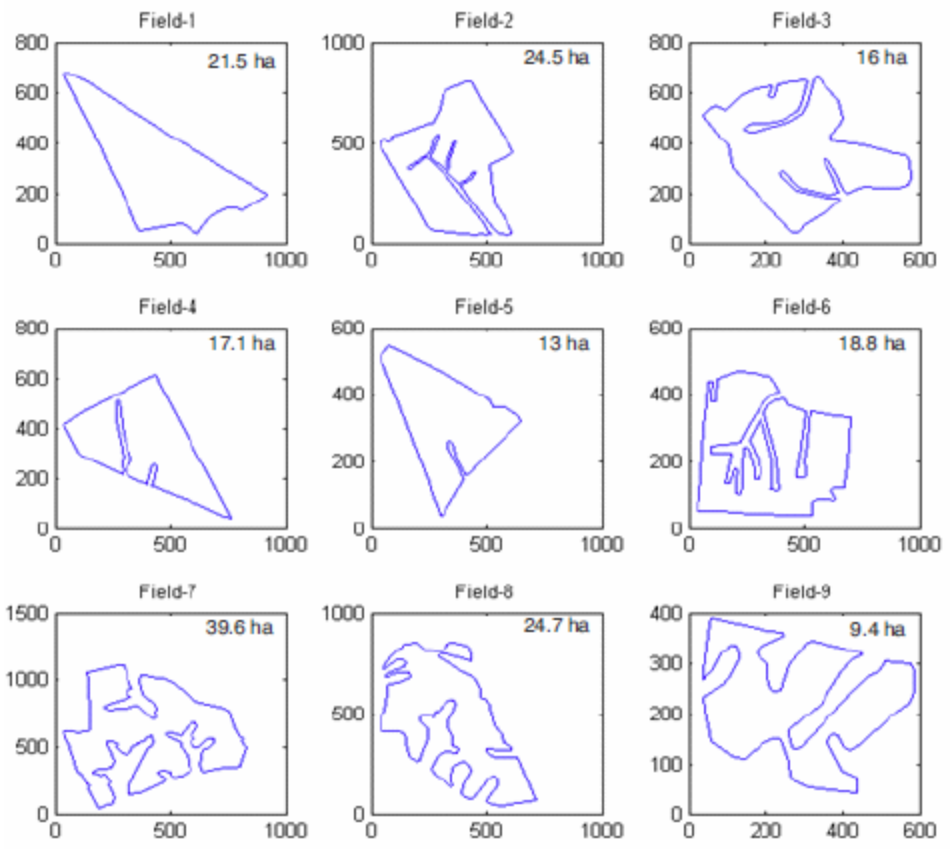


Figure 36: Field Boundaries used for Estimated Off-Rate Application Errors
Source: Zandonadi et al. (2011)

The field boundaries were imported into MATLAB at a resolution of 1 pixel per meter. Converting a field boundary to an application map consisted of converting the boundary to a binary image where pixels inside the border were assigned a value of 1 (white) and 0 (black) otherwise (Figure 37). The *regionprops()* function in MATLAB was used to determine the area and perimeter of the field and the *bwboundaries()* function was used to highlight the field boundary as a red line. Appendix 9.2 contains the script used to estimate off-rate application errors at field boundaries from typical fields found in Kentucky.

Figure 37: Binary Field Application Map with Highlighted Boundary

6.3 Results and Discussion

For square and circle field shapes, off-rate error as a percentage of field area decreased as the field area increased (Table 19 and Table 20). Differences in off-rate error estimates varied by as little as 0.1% and as much as 3.8% with the higher deviations occurring in smaller field areas.

Table 19: Square Field Off-Rate Error Estimates

L/W (m)	Perimeter (m)	Area (m^2)	P/A	2.54 cm	10.2 cm	100 cm
1	4	1	3.92	11.7%	46.4%	100%
2	8	4	1.98	5.9%	24.6%	100%
3	12	9	1.33	4.0%	16.7%	100%
4	16	16	1.00	3.0%	12.7%	99.5%
5	20	25	0.80	2.4%	10.2%	92.8%
6	24	36	0.67	2.0%	8.5%	84.6%
7	28	49	0.57	1.7%	7.3%	77.0%
8	32	64	0.50	1.5%	6.4%	70.3%
9	36	81	0.44	1.3%	5.7%	64.6%
10	40	100	0.40	1.2%	5.1%	59.6%

Table 20: Circle Field Off-Rate Error Estimates

Diameter (m)	Perimeter (m)	Area (m^2)	P/A	2.54 cm	10.2 cm	100 cm
1	3.1	0.8	4.00	12.8%	49.1%	100%
2	6.3	3.1	2.00	6.5%	25.3%	100%
3	9.4	7.1	1.33	4.3%	17.1%	100%
4	12.6	12.6	1.00	3.2%	12.9%	100%
5	15.7	19.6	0.80	2.6%	10.3%	96.5%
6	18.8	28.3	0.67	2.2%	8.6%	88.3%
7	22.0	38.5	0.57	1.9%	7.4%	80.4%
8	25.1	50.3	0.50	1.6%	6.5%	73.4%
9	28.3	63.6	0.44	1.4%	5.8%	67.3%
10	31.4	78.5	0.40	1.3%	5.2%	62.0%

Linear regressions were generated for the P/A ratios versus off-rate error estimates. The offsets of the linear regressions were driven to zero to maintain the property that percentage of a field containing off-rate errors due to GNSS accuracy approached zero as the field area become very large. Square fields (Figure 38) exhibited a slightly smaller slope than circle fields (Figure 39). This illustrated that the larger perimeter in circle fields relative to area caused an increase in the estimated off-rate error due to GNSS accuracy.

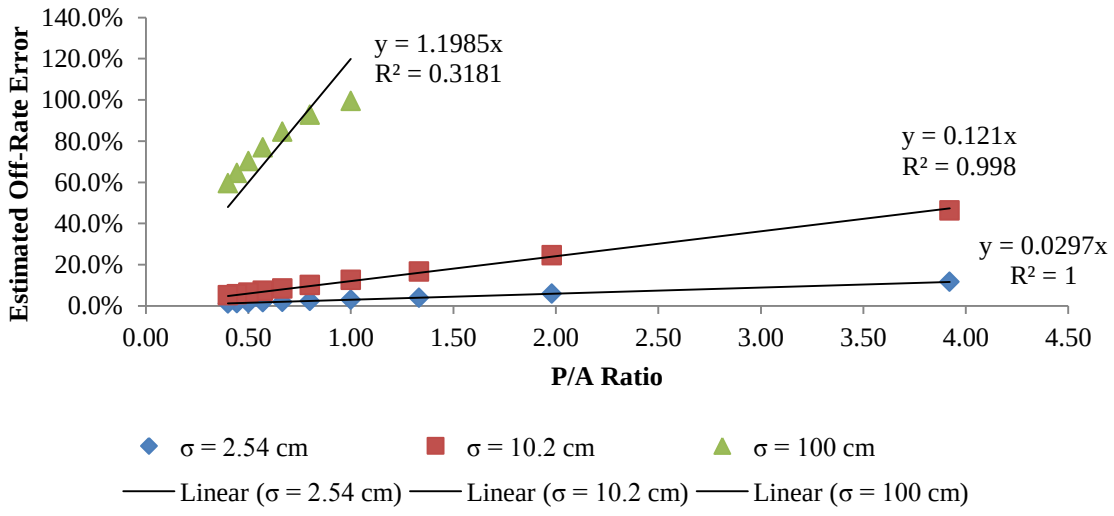


Figure 38: Square Field P/A Ratio versus Estimated Off-Rate Error

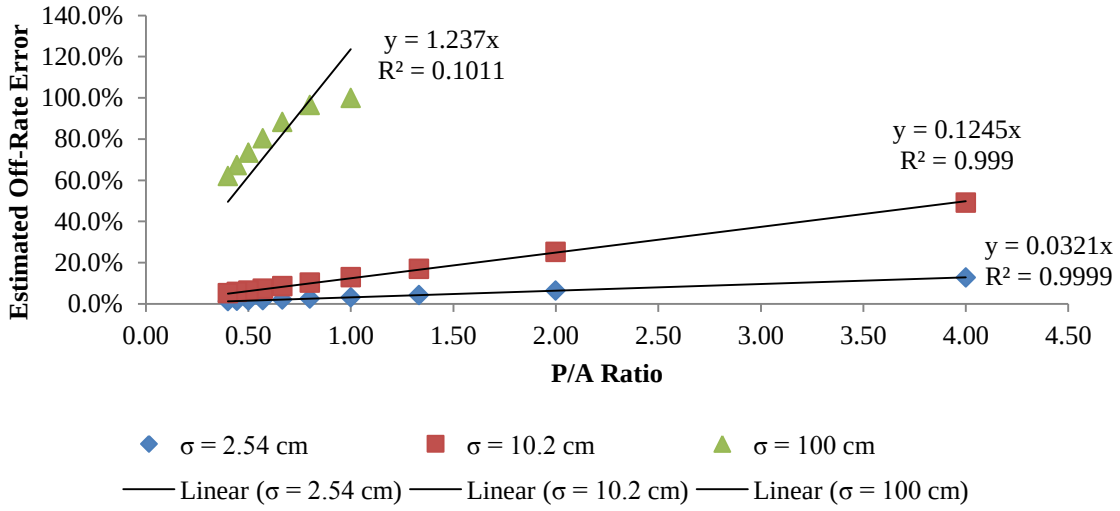


Figure 39: Circle Field P/A Ratio versus Estimated Off-Rate Error

Estimated off-rate error due to GNSS accuracy was found to be small for the nine test fields for all three GNSS accuracy levels tested (Table 21). Estimated off-track error at 2.54 cm and 10.2 cm levels of accuracy were less than 0.08% and 0.3%, respectively. At a 100 cm level of accuracy, off-track error was never greater than 2.06%.

Table 21: Estimated Off-Track Error for Nine Typical Fields in Kentucky

Field	P/A Ratio	σ = 2.54 cm	σ = 10.2 cm	σ = 100 cm
1	0.012	0.04%	0.14%	0.77%
2	0.018	0.06%	0.22%	1.22%
3	0.021	0.07%	0.26%	1.42%
4	0.017	0.05%	0.21%	1.14%
5	0.016	0.05%	0.19%	1.04%
6	0.025	0.08%	0.30%	1.65%
7	0.017	0.05%	0.21%	1.15%
8	0.020	0.06%	0.24%	1.32%
9	0.031	0.03%	0.10%	2.06%

6.4 Conclusions

A method for estimating off-rate error from an application map and a GNSS error distribution was discussed. The estimated off-rate error was limited to the boundary of the field, and therefore is likely to represent a lower limit on off-rate error due to dynamic GNSS accuracy. In other words, the off-rate errors due to parallel passes were ignored in this study. As a result, the estimated off-rate errors as a percentage of the field area were small. One limitation of this type of analysis was that it was done in a raster format. This limited both the size of the maps and the resolution to which off-rate error could be calculated. Converting the analysis to a vector format may allow more precise estimates of off-rate errors at field boundaries, as well as the ability to estimate off-rate and off-target errors throughout the entire field.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

7.1.1 Synchronizing Serial Data Streams with GNSS Time

The timing device developed in this study was designed to accurately time-stamp PPS signals and asynchronous serial data streams to within 17.1 µs. Validation showed that the sensor board time measurements closely matched time measurements made with a digital oscilloscope. The variability between the two measurement systems was negligible when considered for agricultural applications. Assuming a maximum vehicle speed of 10 m/s, all results showed that position error would be less than 1 mm as a result of errors in time measurement. This system will be useful for testing GNSS devices and auto-guidance systems under the ISO-12188 standard. The precision of this system may be improved by removing the restriction that all time comparisons are made in hardware or by using a device with IC registers that are larger than 16 bits.

7.1.2 Test Fixture Design and Analysis

The rotary test fixture was able to precisely control the angular velocity of a GNSS receiver or TTS target up to 5 rad/s with a standard deviation of 0.007 rad/s. Placing the GNSS receiver or TTS target at a 1 m radius will result in an instantaneous velocity of 5 m/s with a standard deviation of 0.007 m/s. These specifications are well within the speed requirements defined in ISO 12188-1.

7.1.3 Tracking Total Station Testing

Testing has shown that the prism and reflector targets have a similar average latency. The variability in latency for the prism target was several times greater than the reflector target. Since averages cannot be used for position measurements made on-the-fly, it is recommended that the prism target not be used for dynamic applications where millimeter resolution is required. There is a distinct and significant trend in average latency for the reflector target. However, this may have resulted from TTS measurement error and not actual latency. As angular velocity increased, position error increased, which may have had an effect on the calculated phase shift between the TTS and test fixture.

Distance from the TTS to the reflector target was shown to not have a significant effect on measurement error for most angular velocities tested. In the cases where there was a significant difference, that difference was less than 1.5 mm, which fulfills the order-of-magnitude accuracy requirement prescribed by the ISO 12188-1 standard. At higher angular velocities, the accuracy of the TTS is at a similar level to the static accuracy specified for most RTK GNSS devices. It is not known whether or not this level of accuracy will suffice for dynamic GNSS at angular velocities because no data on dynamic GNSS accuracy at high angular velocities has been published. Furthermore, no comparison has been made between angular velocity and actual speed. It has been assumed that constant change in direction is one of the worst case scenarios for the TTS because the system is marginally stable. Both the TTS, and the interpolation method used to calculate TTS position at PPS events are expected to perform better when travelling in straight paths.

7.1.4 Dynamic GNSS Error Modeling

The angular velocity and filter level of an AgGPS 252 GPS receiver was shown to have no significant effect on the average error over a 20 hour period. However, the standard deviation of error tended to increase at higher angular velocities. Increases in the filter level also resulted in larger standard deviations in error. The difference in standard deviation between the best and worst tests at each filter level was nearly two-fold. This indicates that GNSS receiver dynamics may play an important role in estimating and measuring field performance of agricultural machinery using satellite based positioning systems.

Along- and off-track errors were shown to be remarkably similar when testing under steady-state sinusoidal conditions. This trend was not expected, as increasing angular velocities and filter levels were hypothesized to result in a phase shift and/or a change in the magnitude of error. Neither was observed in any test. The only consistent significant difference in results between testing parameters was the standard deviation of error.

Given the wide range of distributions, and their shapes, it can be concluded that a mean and standard deviation alone is not enough to fully define dynamic GNSS error. Distributions tended to fall somewhere between normal and uniform. A discrete probability density function for each scenario is one way to get around the issue of the non-stationary distributions between testing parameters. However, the changes in the distributions of error are small when compared to the scale of field operations. Differences in standard deviations on the sub-millimeter level alone are not likely to result in any measureable performance difference in field operations. More research is

needed to understand how the variability in GNSS measurements translates into overall machine performance and ultimately field performance.

7.1.5 Applying a Dynamic GNSS Error Model

A method for estimating off-rate error from an application map and a GNSS error distribution was discussed. The estimated off-track error was limited to the boundary of the field, and therefore is likely to represent a lower limit on off-rate error due to dynamic GNSS accuracy. In other words, the off-rate errors due to parallel passes were ignored. As a result, the estimated off-rate errors as a percentage of the field area were small. One limitation of this type of analysis was that it was done in a raster format. This limited both the size of the maps and the resolution to which off-rate error could be calculated. Converting the analysis to a vector format may allow more precise estimates of off-rate errors at field boundaries, as well as the ability to estimate off-rate and off-target errors throughout the entire field.

7.2 Future Work

The implication of dynamic GNSS error in an agricultural process depends on several parameters including accuracy requirements, spatial resolution, and temporal resolution. Dynamic GNSS error models require these parameters to be well defined in a consistent manner such that varying GNSS technologies can be directly compared to one another in terms of expected field performance. Standardized guidelines for expressing accuracy requirements and spatial/temporal resolution would be beneficial for producers when selecting GNSS equipment as well as for predicting expected performance at a meaningful level. More research is needed to better understand what GNSS error contributes to the overall error of a system.

Determining the dynamic error of the highest accuracy GNSS receivers under ISO 12188 remains a difficult task due to the accuracy of position reference. Without the use of a mechanical test fixture that forces a receiver to follow an exact path, a position reference is required. The TTS evaluated in this dissertation showed promise as a possible candidate for a ground-based position reference to RTK GNSS receivers. However, the error measured from the TTS was not an order of magnitude less than the RTK GNSS receiver at higher angular velocities. It may be possible to compensate for the large change in magnitude observed in the TTS position output through post-processing which would make the system a viable option for a position reference in standardize GNSS testing. Better understand of the internal filtering process of the TTS is needed so that its output can properly compensated for dynamics.

APPENDICIES

Appendix 1: Signal Timing Device Software

1.1 Main Program

```
////////////////////////////////////////////////////
//    Title: Rotary Test Fixture Program (c) 2012   //
//  Filename: main.c                                //
//    Author: Michael P. Sama                       //
//    Date: 1/28/11 - 5/18/12                       //
////////////////////////////////////////////////////

#define SYSCLK    15000000UL //Define the system clock speed as 15 MHz
#define FCY 3750000UL //Define the instruction clock speed as 3.75 MHz

//Pin aliases for input capture modules
#define IC1_OV   IC1CONbits.ICOV

//Pin aliases for on board push buttons
#define     B1    PORTEbits.RE0 //Button 1
#define     B2    PORTEbits.RE1 //Button 2
#define     B3    PORTEbits.RE2 //Button 3
#define     B4    PORTEbits.RE3 //Button 4

#include <p30fxxxx.h> //base library for the dsPIC30F4011
#include <libpic30.h> //general c30 Functions (delays, etc.)
#include <uart.h> //universal asynchronous receiver/transmiter
#include <string.h> //string manipulation
#include <stdio.h> //standard input/output
#include <InCap.h> //input capture
#include "AOUT.h" //custom analog output
#include "RS232.h" //custom RS232

_FOSC(HS) //set the oscillator to external high speed crystal
_FWDT(WDT_OFF) //turn off the watch dog timer

char TXdata[128]; //data transmit string for RS-232
char RXdata[128]; //data receive string for RS-232

unsigned int PPS = 0;   //global PPS event variable
unsigned int PPStime = 0;     //global PPS time variable
unsigned int TRG = 0; //global TTS event variable
unsigned int TRGtime = 0; //global TTS time variable

void __attribute__((__interrupt__)) _U1RXInterrupt(void); //declare the
interrupt handler for the UART1 Reciever
void __attribute__((__interrupt__)) _U2RXInterrupt(void); //declare the
interrupt handler for the UART2 Reciever
```

```
void __attribute__((__interrupt__)) _IC1Interrupt(void);  //declare the
interrupt handler for the IC1
void __attribute__((__interrupt__)) _IC2Interrupt(void);  //declare the
interrupt handler for the IC2

//UART1 Receive Interrupt Handler
void __attribute__((interrupt, no_auto_psv)) _U1RXInterrupt(void)
{
      IFS0bits.U1RXIF = 0;    //reset the UART1 receive interrupt flag
      char Character = ReadUART1();  //read a character from UART1
      if (Character == 0x3E) //if the character is the end of a TTS
message
      {
            IC2CONbits.ICM = 2;  //then enable IC2 to capture every
falling edge
            if (TRG)  //if a TTS event has occured
            {
                  TRG = 0;  //reset the TTS event trigger
                  sprintf(TXdata,"%u\r\n><",TRGtime);  //form a TTS
event string
                  putsUART1((unsigned int *) TXdata);  //output the TTS
event string
            }
      }
      else
      {
            putcUART1(Character); //otherwise just pass the character
along
      }
}

//UART2 Receive Interrupt Handler
void __attribute__((interrupt, no_auto_psv)) _U2RXInterrupt(void)
{
      IFS1bits.U2RXIF = 0; //reset the UART2 receive interrupt flag
      char Character = ReadUART2(); //read a character from UART2
      CharacterBuffer(Character);  //place the character in a buffer
for processing
}

//IC1 Receive Interrupt Handler
void __attribute__((interrupt, no_auto_psv)) _IC1Interrupt(void)
{
      IFS0bits.IC1IF = 0; //reset the IC1 interrupt flag
      PPStime = IC1BUF;  //store the PPS event time
      PPS = 1;  //indicate that a PPS event has occured
}

//IC2 Receive Interrupt Handler
void __attribute__((interrupt, no_auto_psv)) _IC2Interrupt(void)
{
      IC2CONbits.ICM = 0; //disable the IC2 capture after 1 event
      IFS0bits.IC2IF = 0; //reset the IC2 interrupt flag
      TRGtime = IC2BUF; //store the TTS event time
      TRG = 1; //indicate that a TTS event has occured
}
```

```
//Main function
int main (void)
{
      TRISC = 0b101111111111111;  //PORTC digital I/O directions
      TRISD = 0b0011;   //PORTD digital I/O directions
      TRISE = 0b11111111; //PORTE digital I/O directions
      TRISF = 0b010001; //PORTF digital I/O directions
      ADPCFG = 0b0000000000111000; //analog input pins

      IEC0bits.U1RXIE = 1; //enable RX1 receive interrupt
      SetPriorityIntU1RX(5); //and set interrupt priority to 5
      IEC1bits.U2RXIE = 1; //enable RX2 receive interrupt
      SetPriorityIntU2RX(6); //and set interrupt priority to 6
      IEC0bits.IC1IE = 1; //enable IC1 interrupt
      SetPriorityIntIC1(7); //and set interrupt priority to 7
      IEC0bits.IC2IE = 1; //enable IC2 interrupt
      SetPriorityIntIC2(7); //and set interrupt priority to 7

      //Open UART1 19200 8-N-1
      OpenUART1 ( UART_EN &
                        UART_IDLE_CON &
                        UART_DIS_WAKE &
                        UART_DIS_LOOPBACK &
                        UART_DIS_ABAUD &
                        UART_NO_PAR_8BIT &
                        UART_1STOPBIT,
                        UART_INT_TX_BUF_EMPTY &
                        UART_TX_PIN_NORMAL &
                        UART_TX_ENABLE &
                        UART_INT_RX_CHAR &
                        UART_ADR_DETECT_DIS &
                        UART_RX_OVERRUN_CLEAR,
                        11);
      U1MODEbits.ALTIO = 1;   //set UART1 to the default pins

      //Open UART2 19200 8-N-1
      OpenUART2 ( UART_EN &
                        UART_IDLE_CON &
                        UART_DIS_WAKE &
                        UART_DIS_LOOPBACK &
                        UART_DIS_ABAUD &
                        UART_NO_PAR_8BIT &
                        UART_1STOPBIT,
                        UART_INT_TX_BUF_EMPTY &
                        UART_TX_PIN_NORMAL &
                        UART_TX_ENABLE &
                        UART_INT_RX_CHAR &
                        UART_ADR_DETECT_DIS &
                        UART_RX_OVERRUN_CLEAR,
                        11);

      //Configure Timer 2
      T2CONbits.TSIDL = 0;    //continue timer operation in idle mode
      T2CONbits.TGATE = 0;    //timer gated time accumulation disabled
      T2CONbits.TCKPS = 2;    //timer input clock prescale bits set to
1:64
```

```
      T2CONbits.T32 = 0;            //timer2 and Timer3 form seperate
16-bit timers
      T2CONbits.TCS = 0;            //internal timer clock (FOSC/4)
      T2CONbits.TON = 1;            //start Timer2

      //Configure Input Capture 1
      IC1CONbits.ICM = 0;           //turn off IC1 module while
configuring
      IC1CONbits.ICSIDL = 0;  //input capture module will continute to
operate in CPU idle mode
      IC1CONbits.ICTMR = 1;   //Timer2 contents are captured on capture
event
      IC1CONbits.ICI = 0;           //interrupt on every capture event
      IC1CONbits.ICM = 2;           //capture every falling edge

      //Configure Input Capture 2
      IC2CONbits.ICM = 0;           //turn off IC1 module while
configuring
      IC2CONbits.ICSIDL = 0;  //input capture module will continute to
operate in CPU Idle Mode
      IC2CONbits.ICTMR = 1;   //Timer2 contents are captured on capture
event
      IC2CONbits.ICI = 0;           //interrupt on every capture event
      //IC2CONbits.ICM = 2;   //capture remains disabled, to be enabled
by serial character

      InitializeI2C(); //turn on the I2C module
      SetVoltage(0); //set analog outputs to zero volts

      //main loop
      while(1)
      {
            if (IC1_OV) //if IC1 overflows
            {
                  IC1CONbits.ICM = 0;     //reset module
                  IC1CONbits.ICM = 2;     //capture every falling edge
            }
            if (PPS) //if a PPS event has occured
            {
                  PPS = 0; //reset the PPS event
                  sprintf(TXdata,"$PPS,%u\r\n",PPStime); //form a PPS
time string
                  putsUART2((unsigned int *) TXdata); //output the PPS
time string
            }
            if ((B3 == 0) || (B4 == 0)) //if either button 3 or 4 is
pressed
            {
                  if (B3 == 0) //if button 3 is pressed
                  {
                        SetVoltage(1024);  //set the analog output to
5*1024/4095 volts
                  }
                  else if (B4 == 0)  //if button 4 is pressed
                  {
                        SetVoltage(512); //set the analog output to
5*512/4095 volts
```

```
                  }
                  __delay32(100000); //pause for 100,000 clock cycles
            }
            if (NewMessage())  //if a new serial message has been
received
            {
                  strcpy(RXdata,GetMessage());  //retrieve the message
from the buffer

                  if (RXdata[0] == '\0')  //if nothign was actually
received
                  {
                        putsUART2((unsigned int *)"$RS*01\r\n");
//request a resend
                        strcpy(RXdata,GetMessage()); //retrieve the
message from the buffer
                  }
                  putsUART2((unsigned int *)RXdata);  //loop-back the
same incoming message
                  if ((RXdata[0] = '$') &&  //if a voltage update
message was received
                        (RXdata[1] = 'V') &&
                        (RXdata[2] = ',') &&
                        (RXdata[7] = '*'))
                  {
                        //update the output voltage
                        unsigned int Vout = ((unsigned int) RXdata[3] -
48) * 1000 + ((unsigned int) RXdata[4] - 48) * 100 + ((unsigned int)
RXdata[5] - 48) * 10 + ((unsigned int) RXdata[6] - 48);
                        SetVoltage(Vout);
                  }
            }

      }
      return 0;
}
```

1.2 Analog Output Header File

```
////////////////////////////////////////////////////
//    Title: Analog Output Header File (c) 2012     //
//  Filename: AOUT.h                                //
//    Author: Michael P. Sama                       //
//    Date: 5/18/12                                 //
////////////////////////////////////////////////////

void InitializeI2C(void);
void SetVoltage(unsigned int V);
```

1.3 Analog Output Class

```
////////////////////////////////////////////////////
```

```
//    Title: Analog Ouput Class                    //
//  Filename: AOUT.c                               //
//    Author: Michael P. Sama                      //
//    Date: 5/18/12                                //
/////////////////////////////////////////////////////

#include <p30fxxxx.h> //base library for the dsPIC30F4011
#include "AOUT.h" //header file
#include <i2c.h> //I2C

//Function for intializing the I2C module
void InitializeI2C(void)
{
      unsigned int config2, config1;
      config2 = 0x36; // Baud rate is set for 100 Khz
      config1 = (I2C_ON & I2C_IDLE_CON & I2C_CLK_HLD &
            I2C_IPMI_DIS & I2C_7BIT_ADD &
            I2C_SLW_DIS & I2C_SM_DIS &
            I2C_GCALL_DIS & I2C_STR_DIS &
            I2C_NACK & I2C_ACK_DIS & I2C_RCV_DIS &
            I2C_STOP_DIS & I2C_RESTART_DIS &
            I2C_START_DIS);
      OpenI2C(config1, config2);
      IdleI2C();
}

//Function for setting the analog output voltage
void SetVoltage(unsigned int V)
{
      if (V > 4095)  //if requested voltage is over 5V
      {
            V = 4095; //limit the output to 5V
      }
      //compute the settings based on desire voltage
      unsigned char V2 = (unsigned char) (V%256);
      unsigned char V1 = (unsigned char) (V >> 8);
      StartI2C();  //start the I2C transmission
      while(I2CCONbits.SEN);  //wait until start sequence has completed
      IFS0bits.MI2CIF = 0;  //clear I2C interrupt flag
      IdleI2C(); //return the bus to idle
      MasterWriteI2C(0x90); //write the address
      IdleI2C(); //return the bus to idle
      MasterWriteI2C(V1); //write the first data byte
      IdleI2C(); //return the bus to idle
      MasterWriteI2C(V2); //write the second data byte
      IdleI2C(); //return the bus to idle
      StopI2C(); //terminate the I2C transmission
      IdleI2C(); //return the bus to idle
}
```

1.4 RS232 Header File

```
/////////////////////////////////////////////////////
//    Title: RS232 Header File (c) 2012            //
```

```
//    Filename: RS232.h                                 //
//    Author: Michael P. Sama                           //
//    Date: 1/28/11 - 5/18/12                           //
//////////////////////////////////////////////////////////

void CharacterBuffer(char character);
void MessageCompiler(unsigned int f, unsigned int l);
unsigned char NewMessage(void);
char *GetMessage(void);
```

1.5 RS232 Class

```
//////////////////////////////////////////////////////////
//    Title: RS232 Class (c) 2012                       //
//    Filename: RS232.h                                 //
//    Author: Michael P. Sama                           //
//    Date: 1/28/11 - 5/18/12                           //
//////////////////////////////////////////////////////////

#include <p30fxxxx.h> //base library for the dsPIC30F4011
#include "RS232.h" //header file
#include <stdio.h> //standard input/output
#include <uart.h> //universal asynchronous receiver/transmiter

unsigned char MessageFlag = 0; //message event flag
unsigned int recieved = 0; //# of characters received
char message[64]; //message string
char buffer[128]; //character buffer
unsigned char BufferIndex = 0; //buffer length
int first = -1; //first identifier location
int last = -1; //second identifier location

//Function that pulls a message out of the buffer
void MessageCompiler(unsigned int f, unsigned int l)
{
      unsigned int i; //iterating variable
      unsigned int j = 0;  //iterating variable
      if (f < l){  //if first identifier is before last
            for (i=f; i<l; i++) //for all characters between
identifiers
            {
                  message[j] = buffer[i]; //form the message
                  j++;
            }
            message[j] = '\0'; //terminate with a NULL character
      }
      else //otherwise message wraps around character buffer
      {
            for (i=f; i < 60; i++) //for the first part of the message
            {
                  message[j] = buffer[i]; //form the message
                  j++;
            }
            for (i=0; i < l; i++) //for the second part of the message
```

```
            {
                  message[j] = buffer[i]; //form the message
                  j++;
            }
            message[j] = '\0'; //terminate with a NULL character
      }
      MessageFlag = 1; //indicate a new message is available
      buffer[0] = '\0'; //clear the buffer
      BufferIndex = 0; //reset the number of characters in the buffer
}

//Function for loading new characters on the buffer
void CharacterBuffer (char character)
{
      unsigned int tempfirst;
      unsigned int templast;
      if (character == '$')  //if incoming character is a '$"
      {
            first = BufferIndex; //record its location in the buffer
      }
      else if (character == 0x0D) //if incoming character is a NL
      {
            last = BufferIndex; //record its location in the buffer
            if (first != -1)  //and start the compiling process
            {
                  recieved++;
                  tempfirst = first;
                  templast = last;
                  first = -1;
                  last = -1;
                  MessageCompiler(tempfirst,templast);
            }
      }
      buffer[BufferIndex] = character; //otherwise just load the
characer on the buffer
      buffer[BufferIndex+1] = '\0'; //and terminate the buffer with a
NULL character
      BufferIndex++; //and increment the buffer index

      if (BufferIndex == 249)  //if the buffer has reached the limit
      {
            BufferIndex = 0; //start back at the begining
      }
}

//This function resets the message event flag if raised
unsigned char NewMessage(void)
{
      if (MessageFlag == 1)
      {
            MessageFlag = 0;
            return 1;
      }
      else
      {
            return 0;
      }
```

```
}

//This function returns the location of the data message in memory
char *GetMessage(void)
{
      return message;
}
```

Appendix 2: TTS Test Fixture Program

2.1 Main Program

```
'*******************************************************************************
'*       TITLE: main.vb (c)2012                                                *
'*      AUTHOR: Michael P. Sama                                                *
'*     COMPANY: Biosystems & Agricultural Engineering, Univeristy of Kentucky  *
'*       DATES: 8/13/2012 - 8/16/2012                                          *
'* DESCRIPTION: This class is part of a VB.net form that interfaces with the   *
'*              rotary test fixture for making TTS accuracy measurements       *
'*******************************************************************************

Imports System.Math

Public Class Main

    Private WithEvents SerialPort1 As New TTS 'create an instance of the TTS class
    Private WithEvents SerialPort2 As New RS232 'create an instance of the RS232
class
    Private Logging As Boolean = False 'global variable used to indicate if data
is being logged
    Private nSamples As Integer 'global variable used to store the number of
samples per test
    Private SaveFile As System.Windows.Forms.SaveFileDialog 'dialog interface for
saving files
    Private DataPoints As New List(Of String) 'data stored in a list of strings
    Private PPStime As String 'pulse per second time
    Private PPSangle As String 'fixture angle at PPStime

    'this subfunction is called when the program is first run
    Private Sub Main_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        CheckForIllegalCrossThreadCalls = False 'allow access to objects in
different threads
    End Sub

    'this subfunction is called when th user clicks the connect button
    Private Sub ConnectButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ConnectButton.Click
        If SerialPort1.OpenPort("COM4", 19200, 8, "N", 1) = 1 Then 'connect to the
TTS

        End If
        If SerialPort2.OpenPort("COM3", 19200, 8, "N", 1) = 1 Then 'connect to the
test fixture

        End If
    End Sub

    'this subfunction is called when the user clicks the disconnect button
    Private Sub DisconnectButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles DisconnectButton.Click
        If SerialPort1.ClosePort Then 'disconnect from the TTS
```

```
        End If
        If SerialPort2.ClosePort Then 'disconnect from the text fixture

        End If
    End Sub

    'this subfunction is called when a new message is recieved from the test
fixture
    Private Sub NewPPSMessage() Handles SerialPort2.NewMessage
        Dim Data As String = SerialPort2.GetMessage 'local variable to store the
data string
        PPSTerminal.Text = Data 'display the string in a textbox
        Dim DataItems As String() = Split(Data, ",") 'local variable to spilt
elements out of the data string
        If DataItems(0) = "$PPS" Then 'check to see that the PPS message was
received
            PPStime = DataItems(1) 'fixture angle when PPS message was received
            PPSangle = DataItems(2) 'fixture angle when PPS message was received
        End If
    End Sub

    'this subfunction is called when a new TTS message is received
    Private Sub NewTTSMessage() Handles SerialPort1.NewMessage
        Dim Data As String = SerialPort1.GetMessage 'local variable to store the
data string
        Terminal.Text = Data 'display th string in a textbox
        Dim DataLine As String() = Split(Data, vbCrLf) 'local variable to split
elements out of the data string
        Dim xLine As String() = Split(DataLine(2), "=") 'x position data line
        Dim yLine As String() = Split(DataLine(1), "=") 'y position data line
        Dim zLine As String() = Split(DataLine(3), "=") 'z position data line
        Dim tLine As String() = Split(DataLine(4), ",") 'time data line
        Dim xTTS As Single = CSng(xLine(1)) 'x position
        Dim yTTS As Single = CSng(yLine(1)) 'y position
        Dim zTTS As Single = CSng(zLine(1)) 'z position
        Dim Angle As Integer = CInt(tLine(1)) 'fixture angle when TTS message was
received
        Dim LocalTime As Integer = CInt(tLine(0)) 'fixture time when TTS message
was received
        Dim xFixture As Single = 0.635 * Cos(2 * PI * Angle / 10000) 'calcualted
fixture x coordinate
        Dim yFixture As Single = 0.635 * Sin(2 * PI * Angle / 10000) 'calculated
fixture y coordinate
        Dim xError As Single = xFixture - xTTS 'x error between fixture and TTS
        Dim yError As Single = yFixture - yTTS 'y error between fixture and TTS
        xBox.Text = xTTS 'display the x position in a textbox
        yBox.Text = yTTS 'display the y position in a textbox
        zBox.Text = zTTS 'display the z position in a textbox
        AngleBox.Text = Angle 'display the angle in a textbox
        TimeBox.Text = LocalTime 'display the time in a textbox
        axBox.Text = xFixture 'display the fixture x position in a textbox
        ayBox.Text = yFixture 'display the fixture y position in a textbox
        xErrorBox.Text = xError 'display the x error in a textbox
        yErrorBox.Text = yError 'display the y error in a textbox

        If Logging Then 'if the program should log data
            If DataPoints.Count < nSamples Then 'if the number of desired data
points has not been met
```

```
                DataPoints.Add(LocalTime.ToString & "," & Angle.ToString & "," &
xFixture.ToString & "," & yFixture.ToString & "," & xTTS.ToString & "," &
yTTS.ToString & "," & zTTS.ToString & "," & xError.ToString & "," &
yError.ToString & "," & PPStime & "," & PPSangle)
                SamplesBox.Text = nSamples - DataPoints.Count
            Else 'stop recording a save the data to a file
                Logging = False
                LogButton.Text = "Log"
                SamplesBox.Text = nSamples
                SaveData()
            End If
        End If
    End Sub

    'this subfunction is called when the user clicks on the log button
    Private Sub LogButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles LogButton.Click
        If LogButton.Text = "Log" Then 'If the user intends to start logging
            SaveFile = New System.Windows.Forms.SaveFileDialog
            SaveFile.ShowDialog()
            nSamples = CInt(SamplesBox.Text)
            DataPoints.Clear()
            Logging = True
            LogButton.Text = "Stop Log"
        Else 'the user intends to stop logging
            Logging = False
            LogButton.Text = "Log"
            SamplesBox.Text = nSamples
            SaveData()
        End If
    End Sub

    'this subfunction is called when the correct ammount of data has been recorded
    Private Sub SaveData()
        My.Computer.FileSystem.WriteAllText(SaveFile.FileName,
"Time,Angle,xFixture,yFixture,xTTS,yTTS,zTTS,xError,yError,PPStime,PPSangle" &
vbCrLf, False)
        For i As Integer = 0 To DataPoints.Count - 1
            My.Computer.FileSystem.WriteAllText(SaveFile.FileName, DataPoints(i) &
vbCrLf, True)
        Next
    End Sub

    'this subfunction is called when the user clicks on the speed button
    Private Sub SpeedButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SpeedButton.Click
        Dim SpeedSetting As String = "$V," & Format(CUInt(SpeedBox.Text), "0000")
& "*" & vbCrLf
        If SerialPort2.IsOpen Then
            SerialPort2.SendMessage(SpeedSetting)
        End If
    End Sub

    'this subfunction is called when the user clicks on th stop button
    Private Sub StopButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StopButton.Click
        Dim Msg As String = "$M,1,1*" & vbCrLf
        If SerialPort2.IsOpen Then
```

```
            SerialPort2.SendMessage(Msg)
        End If
    End Sub

    'this subfunction is called when the user clicks on the forward button
    Private Sub ForwardButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ForwardButton.Click
        Dim Msg As String = "$M,1,0*" & vbCrLf
        If SerialPort2.IsOpen Then
            SerialPort2.SendMessage(Msg)
        End If
    End Sub

    'this subfunction is called when the user clicks on the reverse button
    Private Sub ReverseButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ReverseButton.Click
        Dim Msg As String = "$M,0,1*" & vbCrLf
        If SerialPort2.IsOpen Then
            SerialPort2.SendMessage(Msg)
        End If
    End Sub
End Class
```

2.2 RS232 Class

```
'*******************************************************************************
'*       TITLE: RS232.vb (c)2013                                               *
'*      AUTHOR: Michael P. Sama                                                *
'*     COMPANY: Biosystems & Agricultural Engineering, University of Kentucky  *
'*       DATES: 3/24/09 - Current                                              *
'* DESCRIPTION: This class provides a method for accessing a RS232 COM Port    *
'*              using the SerialPort class.  Input characters are buffered     *
'*              and searched for valid strings starting with "$" and ending    *
'*              with "\r".  When a valid string is found, it is removed from   *
'*              the buffer and stored as a separate string.  A public event is *
'*              raised to let the parent class know a new message is available. *
'*******************************************************************************

Public Class RS232

    Public Event NewMessage() 'event used to indicate a new message has been
received
    Private Buffer As String = "" 'character buffer
    Private Message As String 'data message
    Private WithEvents SerialPort1 As New System.IO.Ports.SerialPort 'serial port
    Private LastOutgoingMessage As String = "" 'stores the previous outgoing
message
    Public PauseSerialInput As Boolean = False 'public variable to pause serial
inputs

    'this subfunction is called when a new instance of the RS232 class is created
    Public Sub New()
        MyBase.New()
    End Sub
```

```
    'this subfunction is used to send a set of bytes from the computer to a device
    Public Sub Write(ByVal BytesToWrite() As Byte, ByVal StartIndex As Integer,
ByVal Length As Integer)
        SerialPort1.Write(BytesToWrite, StartIndex, Length) 'write all bytes
    End Sub

    'this subfunction is used to send a string from the computer to a device
    Public Sub SendMessage(ByVal OutgoingMessage As String)
        LastOutgoingMessage = OutgoingMessage 'store this message as the last
outgoing message
        Try
            SerialPort1.Write(OutgoingMessage) 'write the a string
        Catch ex As Exception
            Dim Dummy As Boolean = False
        End Try

    End Sub

    'this subfunction is used to resend the previous message
    Public Sub ResendMessage()
        Try
            SerialPort1.Write(LastOutgoingMessage) 'write the last outgoing
message
        Catch ex As Exception

        End Try

    End Sub

    'this function is used to retrieve a message
    Public Function GetMessage()
        Return Message
    End Function

    'this function is used to close the serial port
    Public Function ClosePort()
        If SerialPort1.IsOpen Then 'only close the port if it's already open
            Try
                SerialPort1.Close()
                Return 1
            Catch ex As Exception
                Return 0
            End Try
        Else
            Return 1
        End If
    End Function

    'this function is used to open the serial port
    Public Function OpenPort(ByVal PortName As String, ByVal BaudRate As Integer,
ByVal DataBits As Integer, ByVal Parity As Char, ByVal StopBits As Single)
        If SerialPort1.IsOpen Then 'if the port is already open
            Return 0 'return a zero
        Else
            Try
                SerialPort1.PortName = PortName 'set the port name
                SerialPort1.BaudRate = BaudRate 'set the baud rate
                SerialPort1.DataBits = DataBits 'set the number of data bits
```

```
                Select Case Parity 'set the parity
                    Case "N", "n", "0"
                        SerialPort1.Parity = IO.Ports.Parity.None
                    Case "E", "e", "2"
                        SerialPort1.Parity = IO.Ports.Parity.Even
                    Case "M", "m", "3"
                        SerialPort1.Parity = IO.Ports.Parity.Mark
                    Case "O", "o", "1"
                        SerialPort1.Parity = IO.Ports.Parity.Odd
                    Case " ", "_", "4"
                        SerialPort1.Parity = IO.Ports.Parity.Space
                    Case Else
                        Return 0
                End Select
                Select Case StopBits 'set the number of stop bits
                    Case 0
                        SerialPort1.StopBits = IO.Ports.StopBits.None
                    Case 1
                        SerialPort1.StopBits = IO.Ports.StopBits.One
                    Case 1.5
                        SerialPort1.StopBits = IO.Ports.StopBits.OnePointFive
                    Case 2
                        SerialPort1.StopBits = IO.Ports.StopBits.Two
                    Case Else
                        Return 0
                End Select

                SerialPort1.ReceivedBytesThreshold = 1 'set the receive byte
threshold
                SerialPort1.Open() 'open the port
                SerialPort1.DiscardInBuffer() 'discard any characters already
received

                'add a handler for the incoming serial data
                AddHandler SerialPort1.DataReceived, AddressOf
Me.SerialBytesReceived

            Catch ex As Exception
                Return 0 'return a zero if any of the previous steps fail
            End Try
            Return 1 'otherwise return a one to indicate the port was succefully
opened
        End If

    End Function

    'this function checks to see if the serial port is already opened
    Public Function IsOpen() As Boolean
        Return SerialPort1.IsOpen()
    End Function

    'this subfunction is automatically called when a new serial character has been
received
    Private Sub SerialBytesReceived(ByVal Sender As Object, ByVal e As
System.IO.Ports.SerialDataReceivedEventArgs)

        If Not PauseSerialInput Then 'if the serial input is not paused
            Try
```

```
                AddToBuffer(SerialPort1.ReadExisting) 'add incoming characters to
the buffer
            Catch ex As Exception

            End Try
        Else
            SerialPort1.DiscardInBuffer() 'otherwise throw out the existing
character
            Buffer = "" 'and clear the character buffer
        End If
    End Sub

    'this subfunction adds characters to the character buffer and calls
stringsearch()
    Private Sub AddToBuffer(ByVal characters As String)
        Buffer += characters 'add the character to the buffer
        StringSearch() 'search for a complete data message
    End Sub

    'this subfunction searches for complete data messages
    Private Sub StringSearch()

        'temporarily stop buffering characters to allow time to process the buffer
        RemoveHandler SerialPort1.DataReceived, AddressOf Me.SerialBytesRecieved

        Dim First As Integer = -1 'initialize the location of the first
identifying chararcter
        Dim Last As Integer = -1 'initialize the location of the last identifying
character
        Try
            First = Buffer.IndexOf("$") 'check if a '$' has been received
            Last = Buffer.LastIndexOf(vbCrLf) 'check if a carriage return and line
feed has been received
        Catch ex As Exception

        End Try

        Try
            If (First <> -1 And Last <> -1) And (Last > First) Then 'if
identifiers have been received in the correct order
                Message = Buffer.Substring(First, (Last - First)) 'pull out the
message
                Buffer = Buffer.Remove(0, Last) 'and remove the message from the
character buffer
                If Not PauseSerialInput Then
                    RaiseEvent NewMessage() 'raise the new message event if the
serial port is not paused
                End If
            End If
        Catch ex As Exception

        End Try

        'restart buffering characters
        AddHandler SerialPort1.DataReceived, AddressOf Me.SerialBytesRecieved

    End Sub
```

```
    'this function gets a list of available port names
    Public Function GetComPortNames()
        Dim PortNames As New List(Of String) 'a lost to store the port names
        For i As Integer = 0 To (My.Computer.Ports.SerialPortNames.Count - 1)
            PortNames.Add(My.Computer.Ports.SerialPortNames(i)) 'add each port
name on the computer to a list
        Next
        BubbleSort(Of String)(PortNames) 'sort the list in alphabetical order
        Return PortNames 'return the list of port names
    End Function

    'this subfunction sorts any list alphabetically
    Private Sub BubbleSort(Of ItemType)(ByRef SortByName As List(Of ItemType))
        Dim x As Integer, y As Integer
        For j As Integer = 0 To (SortByName.Count)
            For k As Integer = (SortByName.Count - 1) To 1 Step -1
                x = Mid(SortByName(k).ToString, 4, SortByName(k).ToString.Length -
3)
                y = Mid(SortByName(k - 1).ToString, 4, SortByName(k -
1).ToString.Length - 3)
                If x < y Then
                    Swap(Of ItemType)(SortByName(k), SortByName(k - 1))
                End If
            Next
        Next
    End Sub

    'this subfunction swaps two items in a list
    Private Sub Swap(Of ItemType)(ByRef v1 As ItemType, ByRef v2 As ItemType)
        Dim temp As ItemType
        temp = v1
        v1 = v2
        v2 = temp
    End Sub

End Class
```

2.3 TTS Class

```
'*******************************************************************************
'*       TITLE: TTS.vb (c)2012                                                 *
'*      AUTHOR: Michael P. Sama                                                *
'*     COMPANY: Biosystems & Agricultural Engineering, University of Kentucky  *
'*       DATES: 8/13/12 - 8/16/12                                             *
'* DESCRIPTION: This class provides a method for accessing a RS232 COM Port    *
'*              using the SerialPort class.  Input characters are buffered     *
'*              and searched for valid strings starting with "<" and ending    *
'*              with ">".  When a valid string is found, it is removed from    *
'*              the buffer and stored as a separate string.  A public event is *
'*              raised to let the parent class know a new message is available. *
'*******************************************************************************

Public Class RS232

    Public Event NewMessage() 'event used to indicate a new message has been
received
    Private Buffer As String = "" 'character buffer
    Private Message As String 'data message
    Private WithEvents SerialPort1 As New System.IO.Ports.SerialPort 'serial port
    Private LastOutgoingMessage As String = "" 'stores the previous outgoing
message
    Public PauseSerialInput As Boolean = False 'public variable to pause serial
inputs

    'this subfunction is called when a new instance of the RS232 class is created
    Public Sub New()
        MyBase.New()
    End Sub

    'this subfunction is used to send a set of bytes from the computer to a device
    Public Sub Write(ByVal BytesToWrite() As Byte, ByVal StartIndex As Integer,
ByVal Length As Integer)
        SerialPort1.Write(BytesToWrite, StartIndex, Length) 'write all bytes
    End Sub

    'this subfunction is used to send a string from the computer to a device
    Public Sub SendMessage(ByVal OutgoingMessage As String)
        LastOutgoingMessage = OutgoingMessage 'store this message as the last
outgoing message
        Try
            SerialPort1.Write(OutgoingMessage) 'write the a string
        Catch ex As Exception
            Dim Dummy As Boolean = False
        End Try

    End Sub

    'this subfunction is used to resend the previous message
    Public Sub ResendMessage()
        Try
            SerialPort1.Write(LastOutgoingMessage) 'write the last outgoing
message
        Catch ex As Exception
```

```
        End Try

    End Sub

    'this function is used to retrieve a message
    Public Function GetMessage()
        Return Message
    End Function

    'this function is used to close the serial port
    Public Function ClosePort()
        If SerialPort1.IsOpen Then 'only close the port if it's already open
            Try
                SerialPort1.Close()
                Return 1
            Catch ex As Exception
                Return 0
            End Try
        Else
            Return 1
        End If
    End Function

    'this function is used to open the serial port
    Public Function OpenPort(ByVal PortName As String, ByVal BaudRate As Integer,
ByVal DataBits As Integer, ByVal Parity As Char, ByVal StopBits As Single)
        If SerialPort1.IsOpen Then 'if the port is already open
            Return 0 'return a zero
        Else
            Try
                SerialPort1.PortName = PortName 'set the port name
                SerialPort1.BaudRate = BaudRate 'set the baud rate
                SerialPort1.DataBits = DataBits 'set the number of data bits
                Select Case Parity 'set the parity
                    Case "N", "n", "0"
                        SerialPort1.Parity = IO.Ports.Parity.None
                    Case "E", "e", "2"
                        SerialPort1.Parity = IO.Ports.Parity.Even
                    Case "M", "m", "3"
                        SerialPort1.Parity = IO.Ports.Parity.Mark
                    Case "O", "o", "1"
                        SerialPort1.Parity = IO.Ports.Parity.Odd
                    Case " ", "_", "4"
                        SerialPort1.Parity = IO.Ports.Parity.Space
                    Case Else
                        Return 0
                End Select
                Select Case StopBits 'set the number of stop bits
                    Case 0
                        SerialPort1.StopBits = IO.Ports.StopBits.None
                    Case 1
                        SerialPort1.StopBits = IO.Ports.StopBits.One
                    Case 1.5
                        SerialPort1.StopBits = IO.Ports.StopBits.OnePointFive
                    Case 2
                        SerialPort1.StopBits = IO.Ports.StopBits.Two
                    Case Else
                        Return 0
```

```
                End Select

                SerialPort1.ReceivedBytesThreshold = 1 'set the receive byte
threshold
                SerialPort1.Open() 'open the port
                SerialPort1.DiscardInBuffer() 'discard any characters already
received

                'add a handler for the incoming serial data
                AddHandler SerialPort1.DataReceived, AddressOf
Me.SerialBytesReceived

            Catch ex As Exception
                Return 0 'return a zero if any of the previous steps fail
            End Try
            Return 1 'otherwise return a one to indicate the port was succefully
opened
        End If

    End Function

    'this function checks to see if the serial port is already opened
    Public Function IsOpen() As Boolean
        Return SerialPort1.IsOpen()
    End Function

    'this subfunction is automatically called when a new serial character has been
received
    Private Sub SerialBytesReceived(ByVal Sender As Object, ByVal e As
System.IO.Ports.SerialDataReceivedEventArgs)

        If Not PauseSerialInput Then 'if the serial input is not paused
            Try
                AddToBuffer(SerialPort1.ReadExisting) 'add incoming characters to
the buffer
            Catch ex As Exception

            End Try
        Else
            SerialPort1.DiscardInBuffer() 'otherwise throw out the existing
character
            Buffer = "" 'and clear the character buffer
        End If
    End Sub

    'this subfunction adds characters to the character buffer and calls
stringsearch()
    Private Sub AddToBuffer(ByVal characters As String)
        Buffer += characters 'add the character to the buffer
        StringSearch() 'search for a complete data message
    End Sub

    'this subfunction searches for complete data messages
    Private Sub StringSearch()

        'temporarily stop buffering characters to allow time to process the buffer
        RemoveHandler SerialPort1.DataReceived, AddressOf Me.SerialBytesRecieved
```

```
        Dim First As Integer = -1 'initialize the location of the first
identifying chararcter
        Dim Last As Integer = -1 'initialize the location of the last identifying
character
        Try
            First = Buffer.IndexOf("<") 'check if a '<' has been received
            Last = Buffer.LastIndexOf(">")'check if a '>' has been received
feed has been received
        Catch ex As Exception

        End Try

        Try
            If (First <> -1 And Last <> -1) And (Last > First) Then 'if
identifiers have been received in the correct order
                Message = Buffer.Substring(First, (Last - First)) 'pull out the
message
                Buffer = Buffer.Remove(0, Last) 'and remove the message from the
character buffer
                If Not PauseSerialInput Then
                    RaiseEvent NewMessage() 'raise the new message event if the
serial port is not paused
                End If
            End If
        Catch ex As Exception

        End Try

        'restart buffering characters
        AddHandler SerialPort1.DataReceived, AddressOf Me.SerialBytesRecieved

    End Sub

    'this function gets a list of available port names
    Public Function GetComPortNames()
        Dim PortNames As New List(Of String) 'a lost to store the port names
        For i As Integer = 0 To (My.Computer.Ports.SerialPortNames.Count - 1)
            PortNames.Add(My.Computer.Ports.SerialPortNames(i)) 'add each port
name on the computer to a list
        Next
        BubbleSort(Of String)(PortNames) 'sort the list in alphabetical order
        Return PortNames 'return the list of port names
    End Function

    'this subfunction sorts any list alphabetically
    Private Sub BubbleSort(Of ItemType)(ByRef SortByName As List(Of ItemType))
        Dim x As Integer, y As Integer
        For j As Integer = 0 To (SortByName.Count)
            For k As Integer = (SortByName.Count - 1) To 1 Step -1
                x = Mid(SortByName(k).ToString, 4, SortByName(k).ToString.Length -
3)
                y = Mid(SortByName(k - 1).ToString, 4, SortByName(k -
1).ToString.Length - 3)
                If x < y Then
                    Swap(Of ItemType)(SortByName(k), SortByName(k - 1))
                End If
            Next
        Next
```

```
        End Sub

        'this subfunction swaps two items in a list
        Private Sub Swap(Of ItemType)(ByRef v1 As ItemType, ByRef v2 As ItemType)
            Dim temp As ItemType
            temp = v1
            v1 = v2
            v2 = temp
        End Sub

    End Class
```

Appendix 3: GNSS Test Fixture Program

3.1 Main Program

```
'*****************************************************************************
'*       TITLE: main.vb (c)2013                                              *
'*      AUTHOR: Michael P. Sama                                              *
'*     COMPANY: Biosystems & Agricultural Engineering, University of Kentucky*
'*       DATES: 9/1/12 - Current                                             *
'* DESCRIPTION: This class is part of a VB.net form that interfaces with the *
'*              rotary test fixture for making GNSS accuracy measurements.   *
'*****************************************************************************

Public Class Main

    Private WithEvents GPS1 As New RS232 'create an instance of the RS232 class
for the dynamic GPS receiver
    Private WithEvents GPS2 As New RS232 'create an instance of the RS232 class
for the static GPS receiver
    Private WithEvents PPS As New RS232  'create an instance of the RS232 class
for the test fixture
    Private Testing As Threading.Thread  'create a background thread for executing
a test
    Private Logging As Threading.Thread  'create a background thread for logging
data
    Private NewGPS1 As Boolean = False   'global variable used to determine if a
message has been received from GPS1
    Private NewGPS2 As Boolean = False   'global variable used to determine if a
message has been received from GPS2
    Private NewPPS As Boolean = False    'global variable used to determine if a
message has been received from the test fixture

    'this subfunction is called when the program is first run
    Private Sub Main_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        CheckForIllegalCrossThreadCalls = False  'allow access to objects in
different threads
    End Sub

    'this subfunction is called when the user clicks the connect button
    Private Sub CommConnect_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CommConnect.Click
        GPS1.OpenPort("COM4", 38400, 8, "N", 1) 'connect to the dynamic GPS
receiver
        GPS2.OpenPort("COM5", 38400, 8, "N", 1) 'connect to th static GPS receiver
        PPS.OpenPort("COM3", 19200, 8, "N", 1)  'connect to the test fixture
    End Sub

    'this subfunction is called when th user clicks the disconnect button
    Private Sub CommDisconnect_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CommDisconnect.Click
        GPS1.ClosePort() 'close the connection to th dynamic GPS receiver
        GPS2.ClosePort() 'close the connection to the static GPS receiver
        PPS.ClosePort()  'close the connection to the test fixture
    End Sub
```

```
    'this subfunction is called when a new message has been recieved from the
dynamic GPS receiver
    Private Sub GPS1_Message() Handles GPS1.NewMessage
        Try
            GPS1Terminal.Text = GPS1.GetMessage 'output the new message in a text
box
            NewGPS1 = True 'indicate that a new message has been received
        Catch ex As Exception

        End Try

    End Sub

    'this subfunction is called when a new message has been recieved from the
dynamic GPS receiver
    Private Sub GPS2_Message() Handles GPS2.NewMessage
        Try
            GPS2Terminal.Text = GPS2.GetMessage 'output the new message in a text
box
            NewGPS2 = True 'indicate that a new message has been received
        Catch ex As Exception

        End Try

    End Sub

    'this subfunction is called when a new message has been recieved from the test
fixture
    Private Sub PPS_Message() Handles PPS.NewMessage
        Try
            PPSTerminal.Text = PPS.GetMessage 'output the new message in a text
box
            NewPPS = True 'indicate that a new message has been received
        Catch ex As Exception

        End Try

    End Sub

    'this subfunction is called when the user clicks the stop button
    Private Sub StopButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StopButton.Click
        Try
            If Testing.IsAlive Then
                Testing.Abort()  'abort the testing thread if it is running
            End If
        Catch ex As Exception

        End Try
        StartTestButton.Enabled = True 're-enable the start test button
        Dim Direction As String = "$M,1,1*" & vbCrLf 'form a string to send to the
test fixture
        If PPS.IsOpen Then
            PPS.SendMessage(Direction) 'update the test fixture settings
        End If
    End Sub
```

```
    'this subfunction is a background thread when a test is in progress
    Private Sub TestingThread()
        StartTestButton.Enabled = False 'disable to start test button
        Dim Speeds As String() = Split(SpeedsBox.Text, ",") 'retrieve the desired
test speeds from a textbox
        Dim Duration As Integer = CInt(DurationBox.Text) 'retrieve the duration of
each test from a textbox
        Dim Delay As Integer = CInt(DelayBox.Text) 'retrieve the delay between
tests from a textbox
        Dim Replications As Integer = CInt(ReplicationsBox.Text) 'retrieve the
number of replications

        Dim DurationTimer As New Stopwatch 'create a stopwatch for measuring the
elapsed time of each test
        Dim Direction As String 'create a string to store the direction

        For i As Integer = 1 To Replications  'for each replication
            For j As Short = 0 To Speeds.Length - 1 'for each speed
                Dim SpeedSetting As String = "$V," & Speeds(j) & "*" & vbCrLf
'form a string to send to the test fixture
                If PPS.IsOpen Then
                    PPS.SendMessage(SpeedSetting) 'update the test fixture speed
setting
                End If
                Threading.Thread.Sleep(2000) 'wait 2 seconds before sending
another message
                Direction = "$M,1,0*" & vbCrLf 'form a string to send to the test
fixture
                If PPS.IsOpen Then
                    PPS.SendMessage(Direction) 'update the test fixture direction
setting
                End If
                Threading.Thread.Sleep(Delay * 1000) 'wait the desired ammount of
time before recording data
                DurationTimer.Reset() 'reset the stopwatch
                DurationTimer.Start() 'start the stopwatch
                SamplesBox.Text = 0 'reset the number of data samples
                NewGPS1 = False 'reset the new data indicator
                NewGPS2 = False 'reset the new data indicator
                NewPPS = False 'reset the new data indicator
                Dim FolderName As String = "C:\GPSTestData\"  'create a folder
location to store a new data file
                Dim FileNamePrefix As String = My.Computer.Clock.LocalTime.Month &
"-" & My.Computer.Clock.LocalTime.Day & "-" & My.Computer.Clock.LocalTime.Year &
"_" & My.Computer.Clock.LocalTime.Hour & "-" & My.Computer.Clock.LocalTime.Minute
& "-" & My.Computer.Clock.LocalTime.Second & "_"
                Dim FileNameSuffix1 As String = Speeds(j) & "_Mobile.csv"
                Dim FileNameSuffix2 As String = Speeds(j) & "_Static.csv"
                While DurationTimer.Elapsed.TotalSeconds < Duration 'while the
duration of a test has not been exceeded
                    Try
                        If NewGPS1 And NewGPS2 And NewPPS Then 'if new data has
been received from all three inputs, record it
                            NewGPS1 = False 'reset the new data indicator
                            NewGPS2 = False 'reset the new data indicator
                            NewPPS = False 'reset the new data indicator
```

```
                            My.Computer.FileSystem.WriteAllText(FolderName &
FileNamePrefix & FileNameSuffix1, GPS1Terminal.Text & "," & PPSTerminal.Text &
vbCrLf, True)
                            My.Computer.FileSystem.WriteAllText(FolderName &
FileNamePrefix & FileNameSuffix2, GPS2Terminal.Text & vbCrLf, True)
                            SamplesBox.Text = CInt(SamplesBox.Text) + 1
                        End If
                    Catch ex As Exception

                    End Try

                End While

            Next
        Next

        'Stop the test fixture...
        Direction = "$M,1,1*" & vbCrLf 'form a string to send to the test fixture
        If PPS.IsOpen Then
            PPS.SendMessage(Direction) 'update the test fixture direction setting
        End If
        StartTestButton.Enabled = True
    End Sub

    'this subfunction is called when the user clicks th start test button
    Private Sub StartTestButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StartTestButton.Click
        Testing = New Threading.Thread(AddressOf TestingThread) 'initialize the
testing thread
        Testing.Start() 'start the testing thread
    End Sub

    'this sub function is a background thread for logging static data
    Private Sub LoggingThread()
        Dim FolderName As String = "C:\GPSTestData\" 'create a folder location to
store a new data file
        Dim FileNamePrefix As String = My.Computer.Clock.LocalTime.Month & "-" &
My.Computer.Clock.LocalTime.Day & "-" & My.Computer.Clock.LocalTime.Year & "_" &
My.Computer.Clock.LocalTime.Hour & "-" & My.Computer.Clock.LocalTime.Minute & "-"
& My.Computer.Clock.LocalTime.Second & "_"
        Dim FileNameSuffix1 As String = "0000_Mobile.csv"
        Dim FileNameSuffix2 As String = "0000_Static.csv"
        NewGPS1 = False 'reset the new data indicator
        NewGPS2 = False 'reset the new data indicator
        NewPPS = False 'reset the new data indicator
        SamplesBox.Text = 0 'reset the number of data samples
        While CInt(SamplesBox.Text < 3600) 'for one hour...
            If NewGPS1 And NewGPS2 And NewPPS Then 'if new data has been received
from all three inputs, record it
                NewGPS1 = False 'reset the new data indicator
                NewGPS2 = False 'reset the new data indicator
                NewPPS = False 'reset the new data indicator
                Try
                    My.Computer.FileSystem.WriteAllText(FolderName &
FileNamePrefix & FileNameSuffix1, GPS1Terminal.Text & "," & PPSTerminal.Text &
vbCrLf, True)
                    My.Computer.FileSystem.WriteAllText(FolderName &
FileNamePrefix & FileNameSuffix2, GPS2Terminal.Text & vbCrLf, True)
```

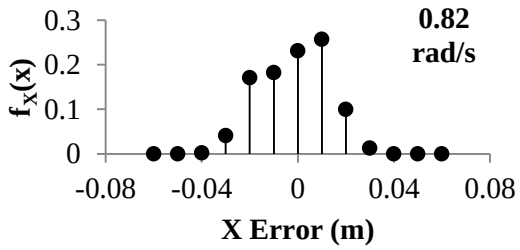
```
                    SamplesBox.Text = CInt(SamplesBox.Text) + 1
                Catch ex As Exception

                End Try

            End If
        End While
    End Sub

    'this subfunction is called when the user clicks the static log button
    Private Sub StaticLogButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StaticLogButton.Click
        Logging = New Threading.Thread(AddressOf LoggingThread) 'initialize the
logging thread
        Logging.Start() 'start the logging thread
    End Sub
End Class
```

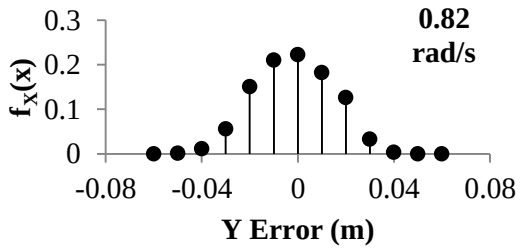
3.2 R2323 Class

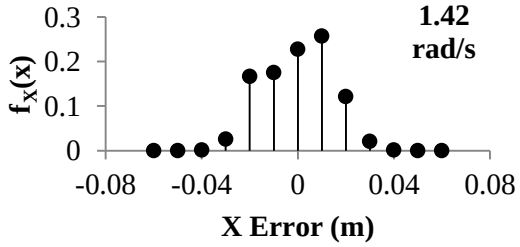
See Appendix 2: 2.2 RS232 Class

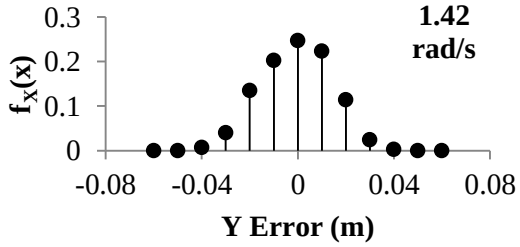
Appendix 4: X and Y Error Discrete Probability Density Functions

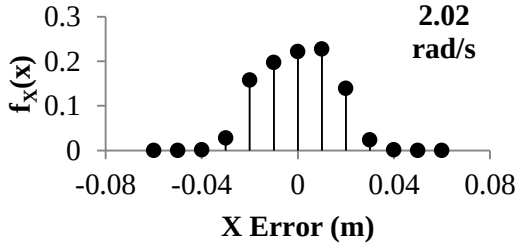
4.1 No Filter

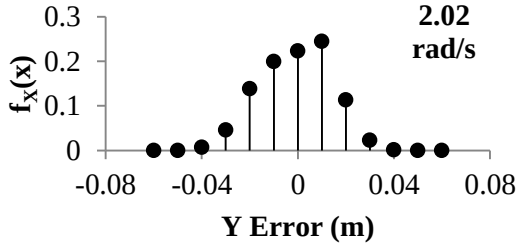


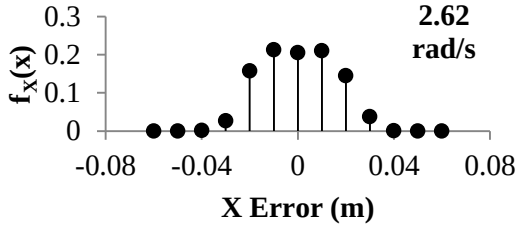


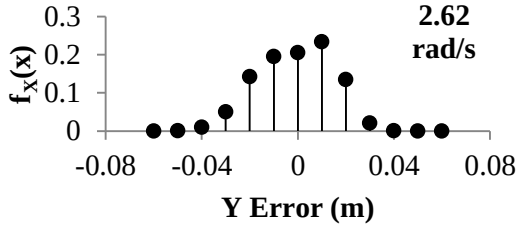


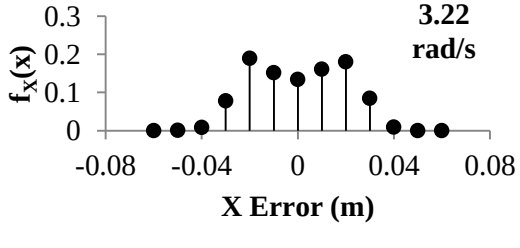


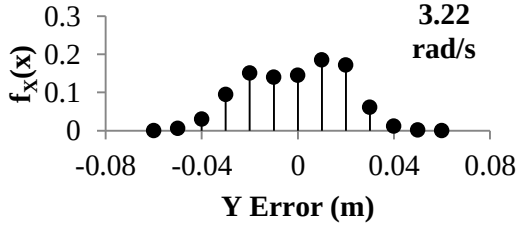




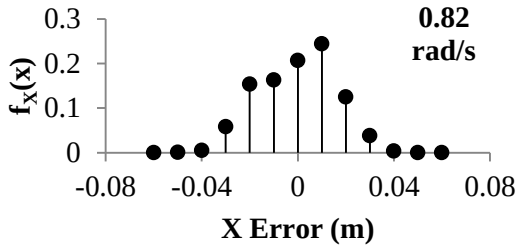


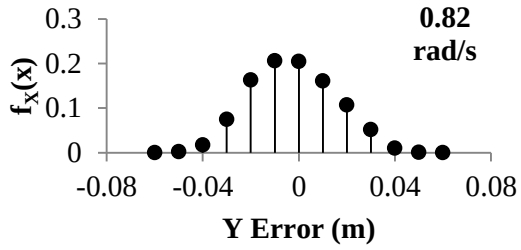


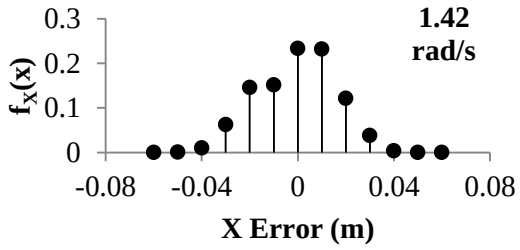


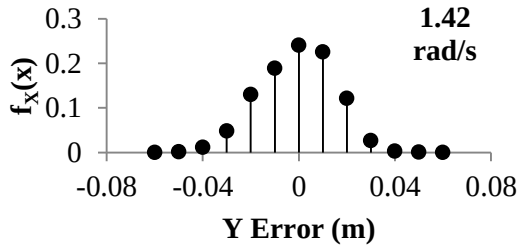


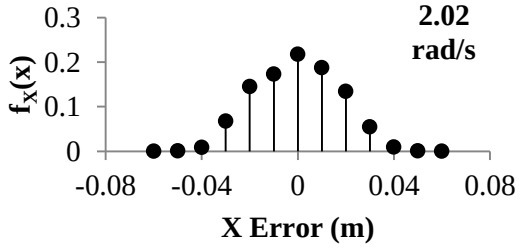
4.2 Normal Filter

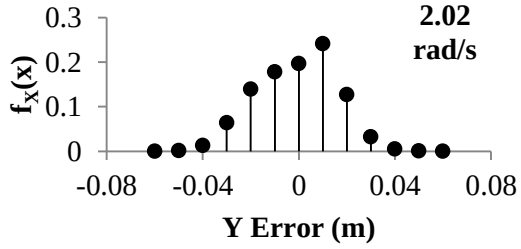


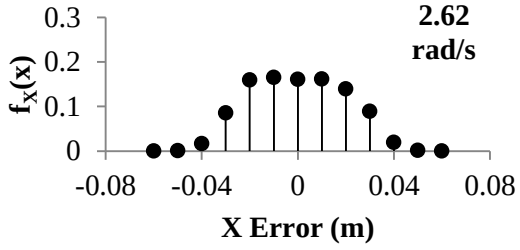


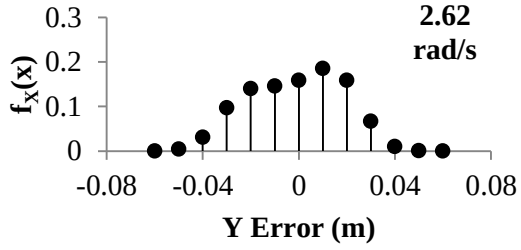


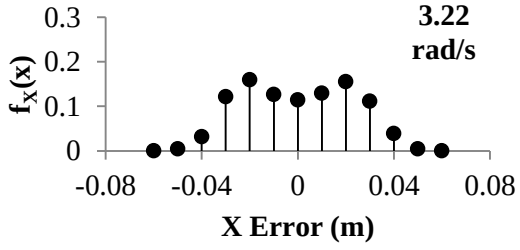


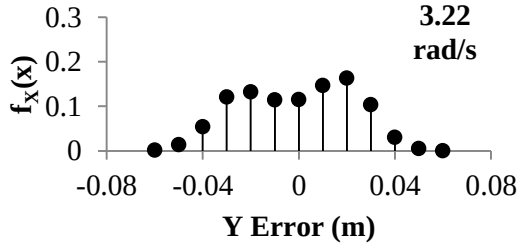




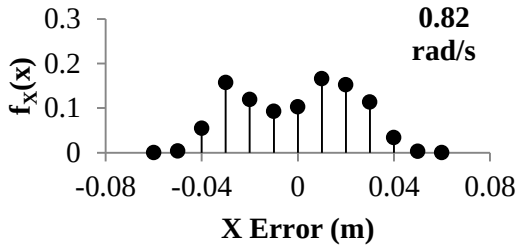


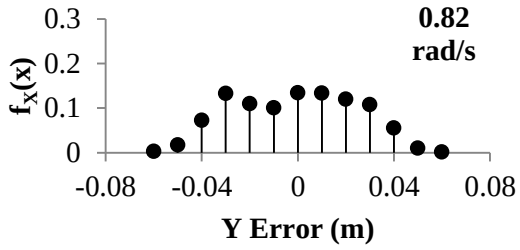


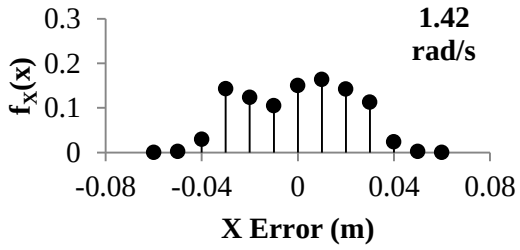


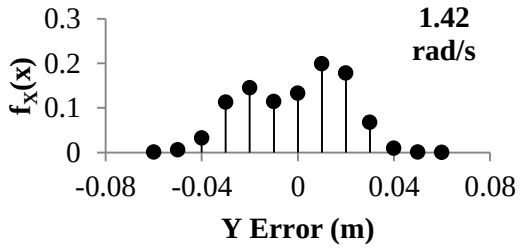


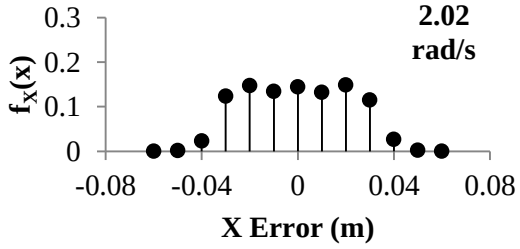
4.3 High Filter

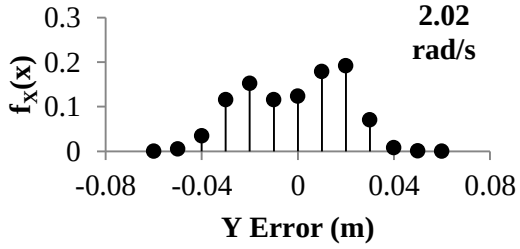


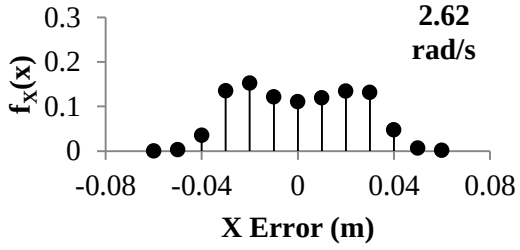


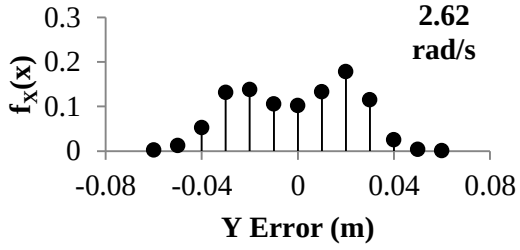


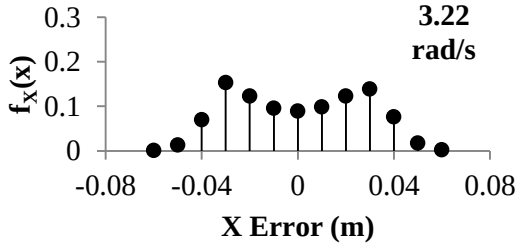


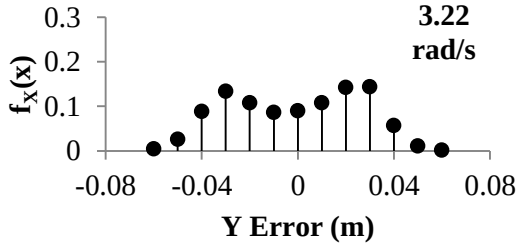




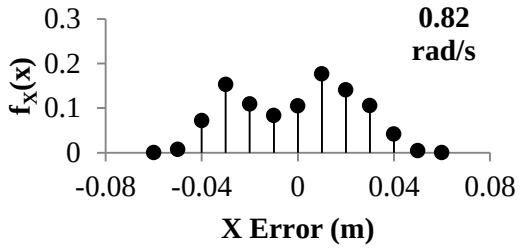


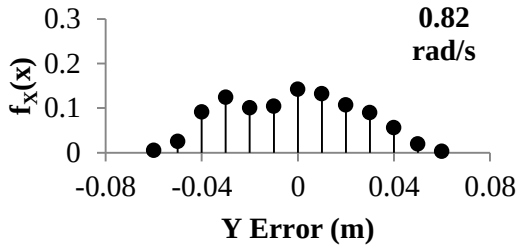


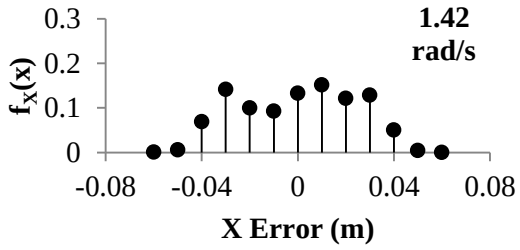


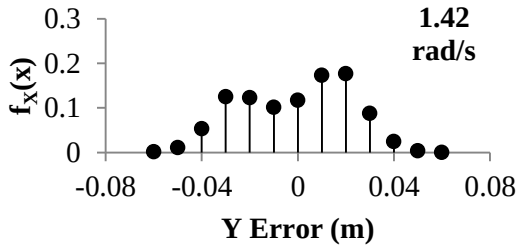


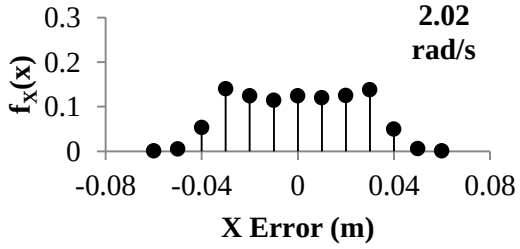
4.4 Max Filter

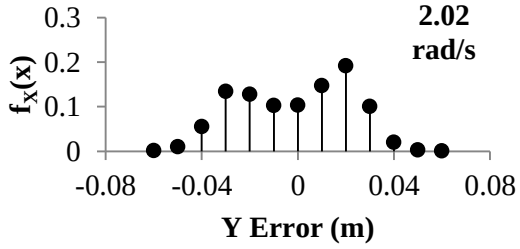


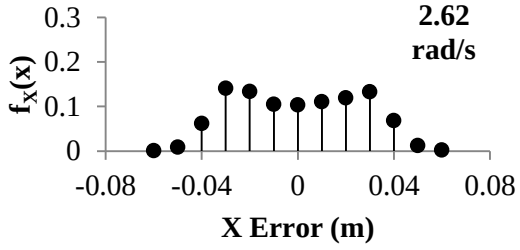


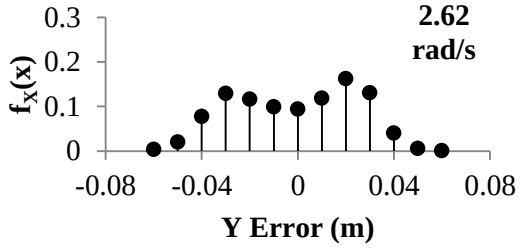


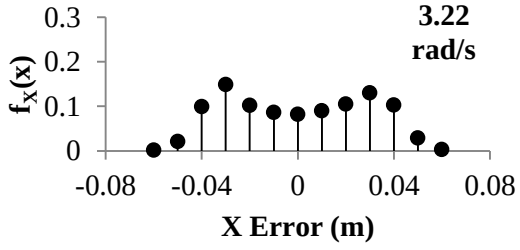


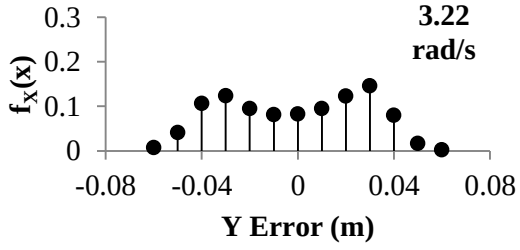






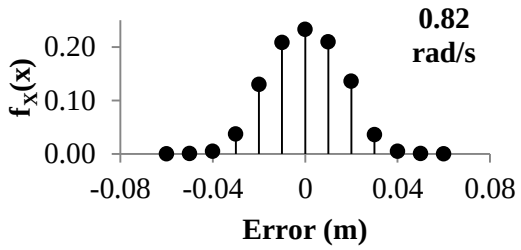


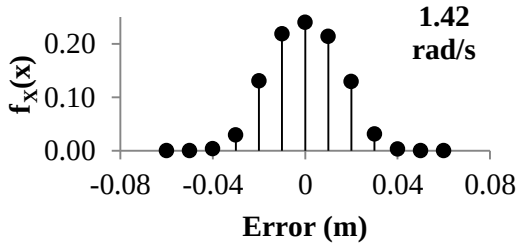


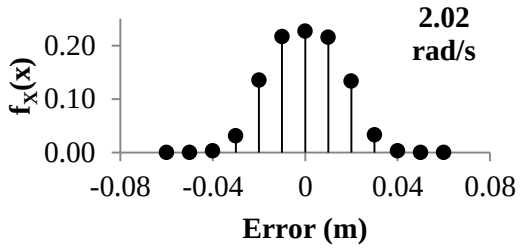


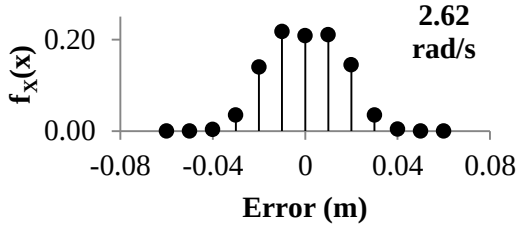
Appendix 5: Along-/Off-Track Error Probability Density Functions

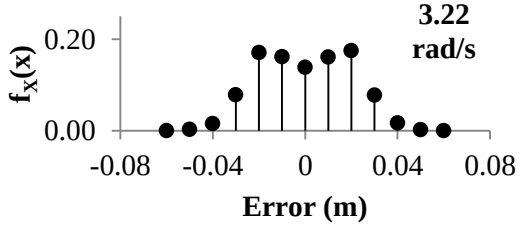
5.1 No Filter



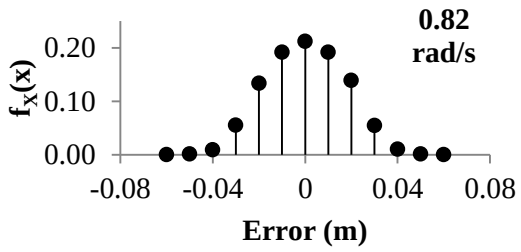


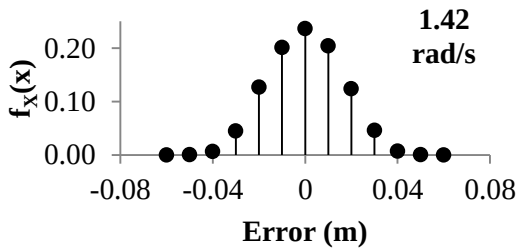


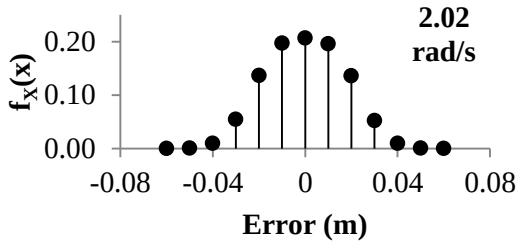


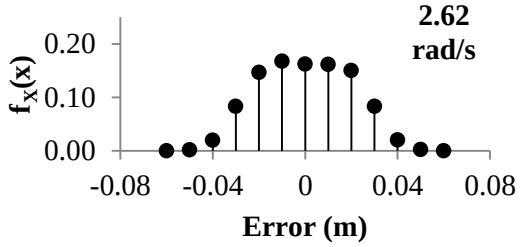


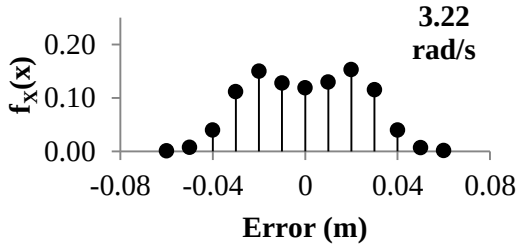
5.2 Normal Filter



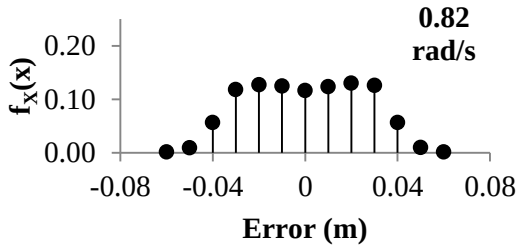


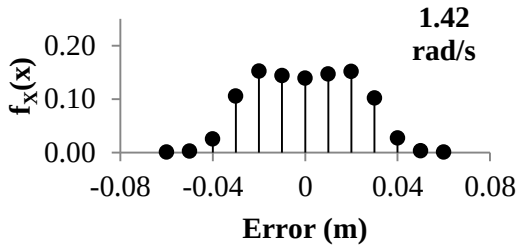


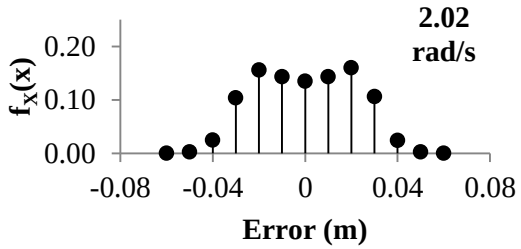


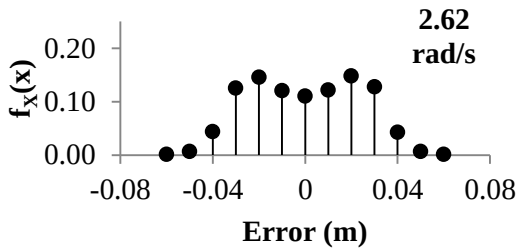


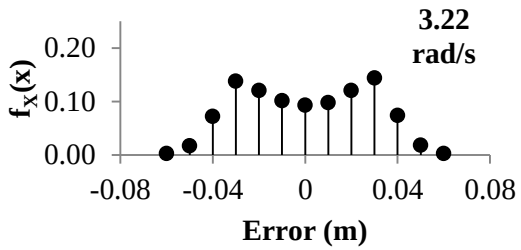
5.3 High Filter



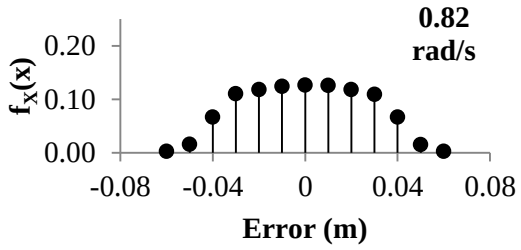


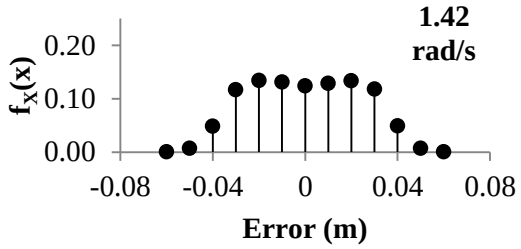


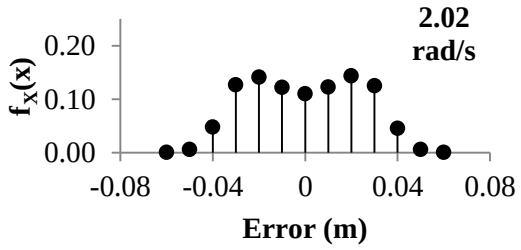


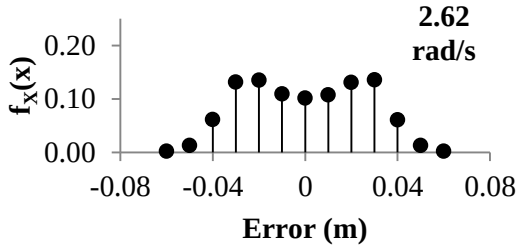


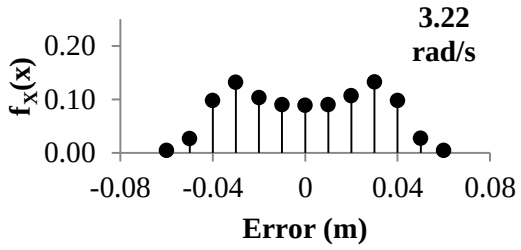
5.4 Max Filter







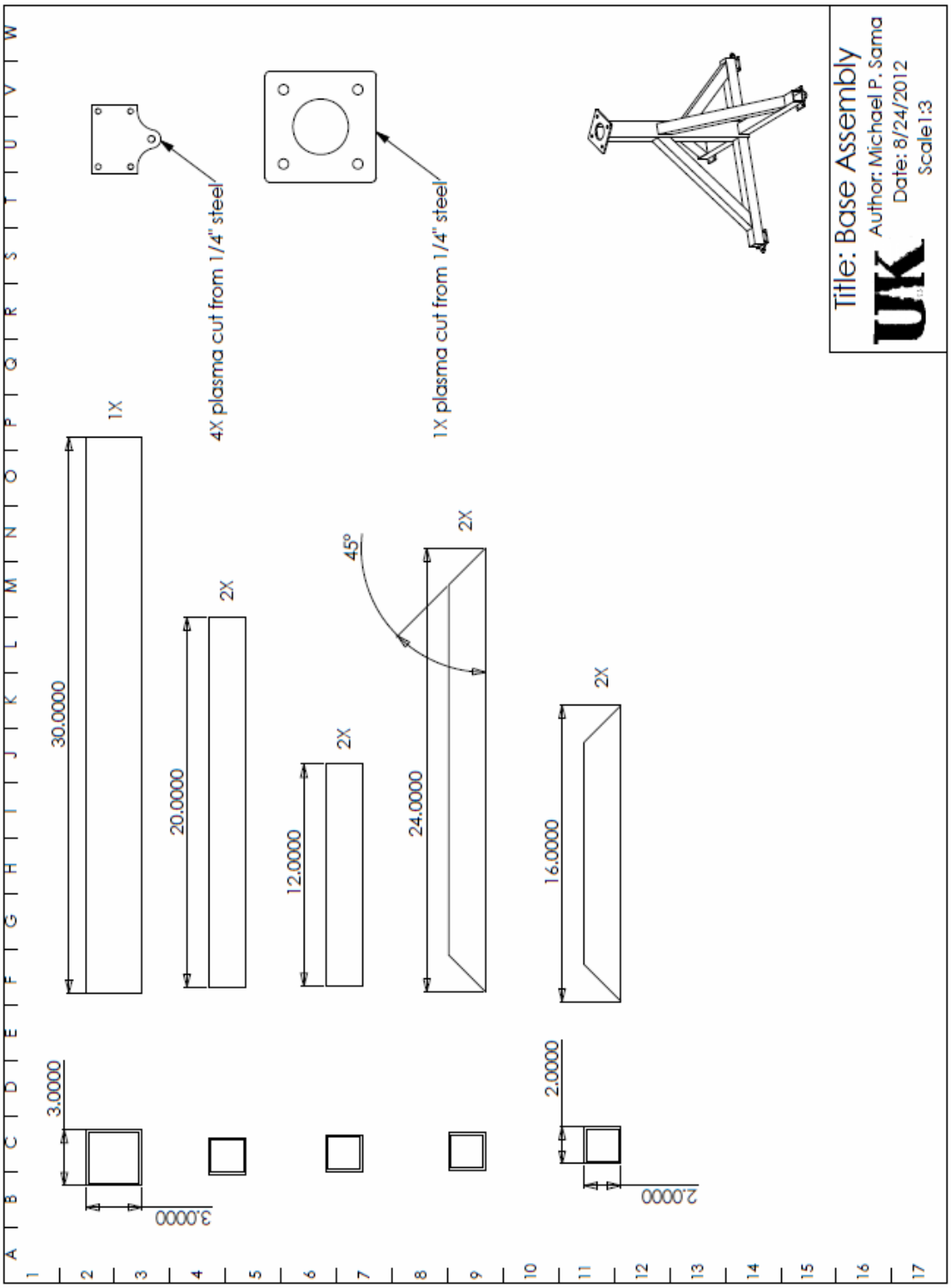




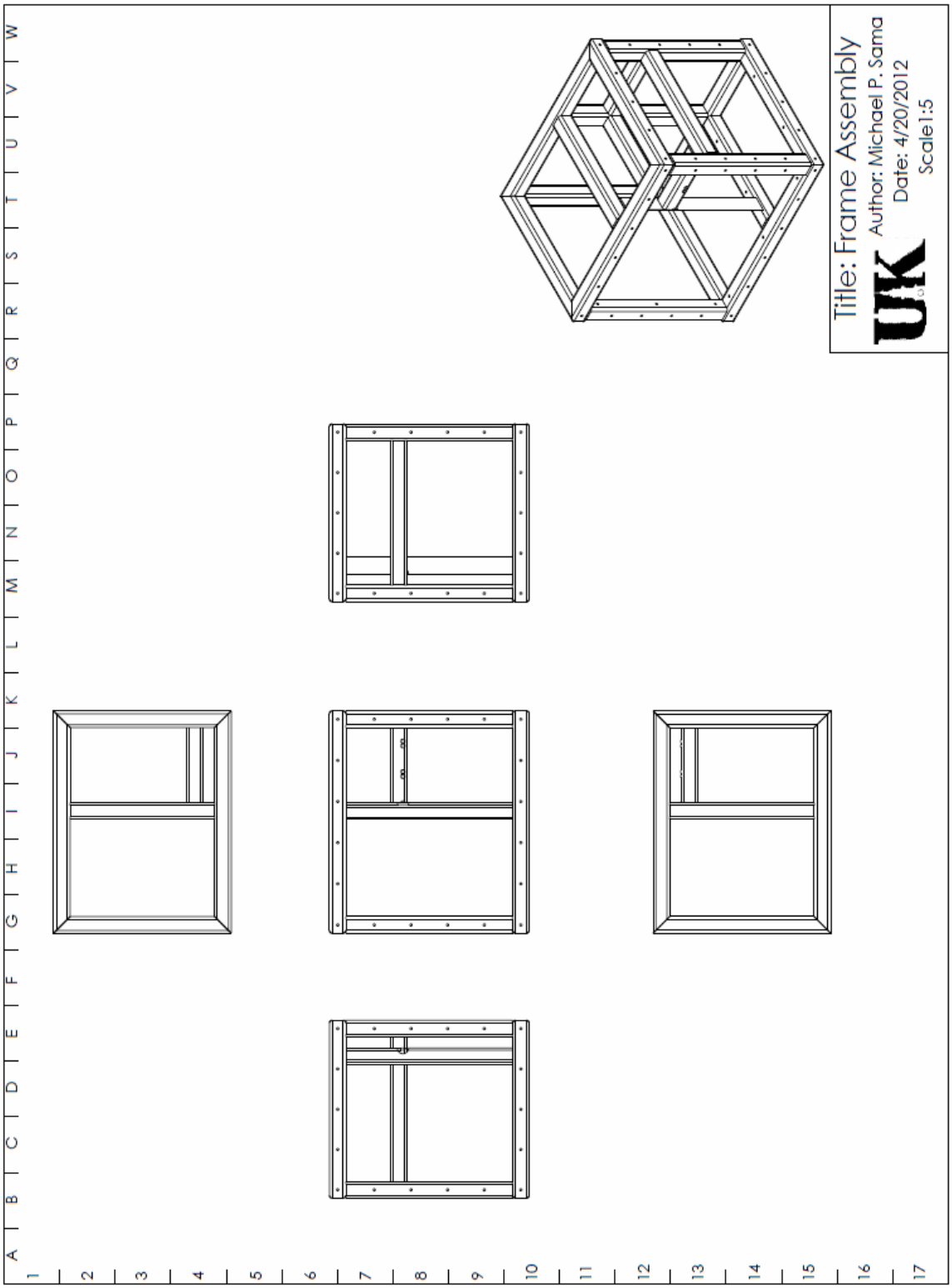
Appendix 6: Rotary Test Fixture Engineering Drawings

Note: The following drawings are not to scale. All units are in inches.

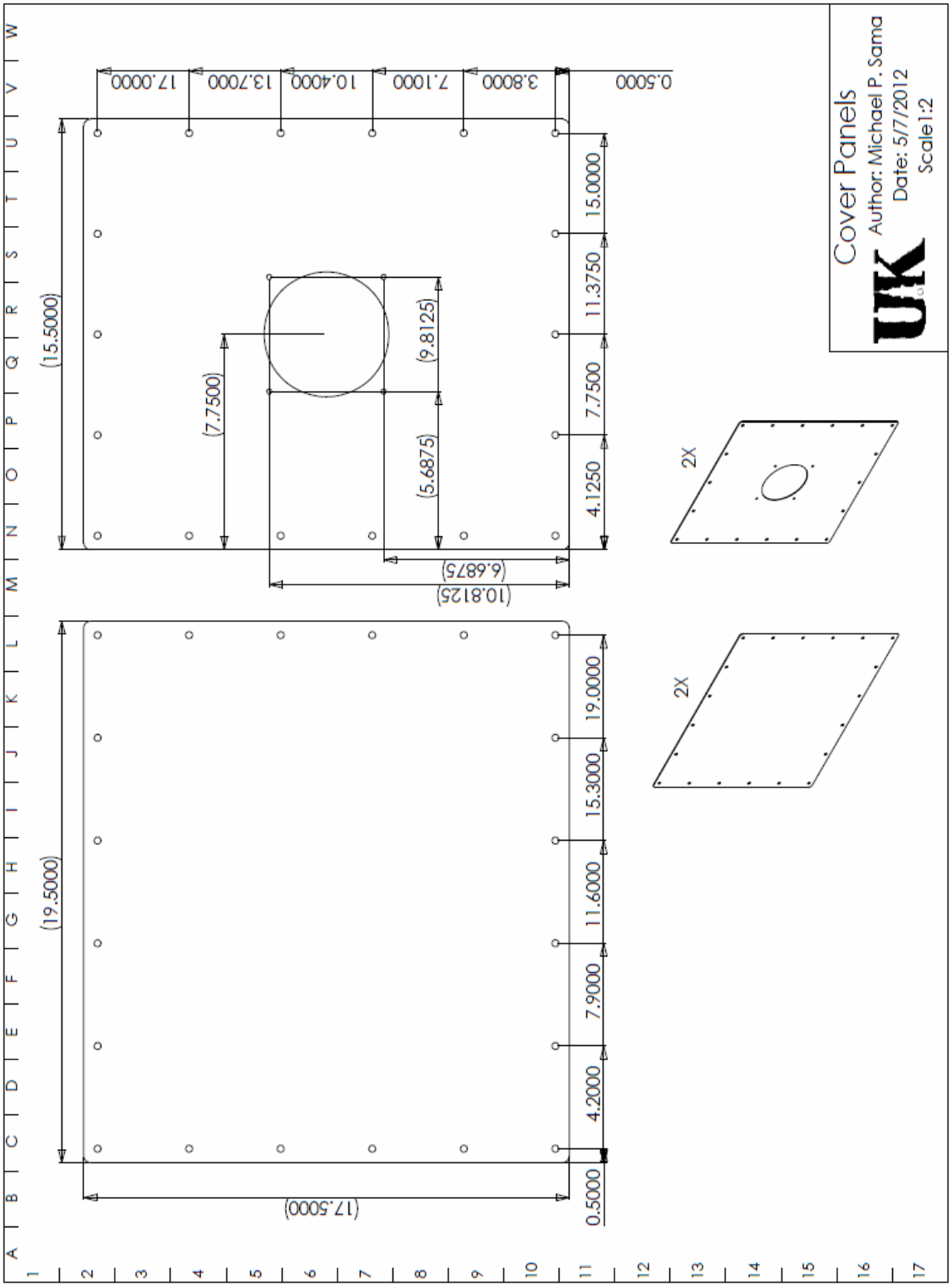
6.1 Base Assembly



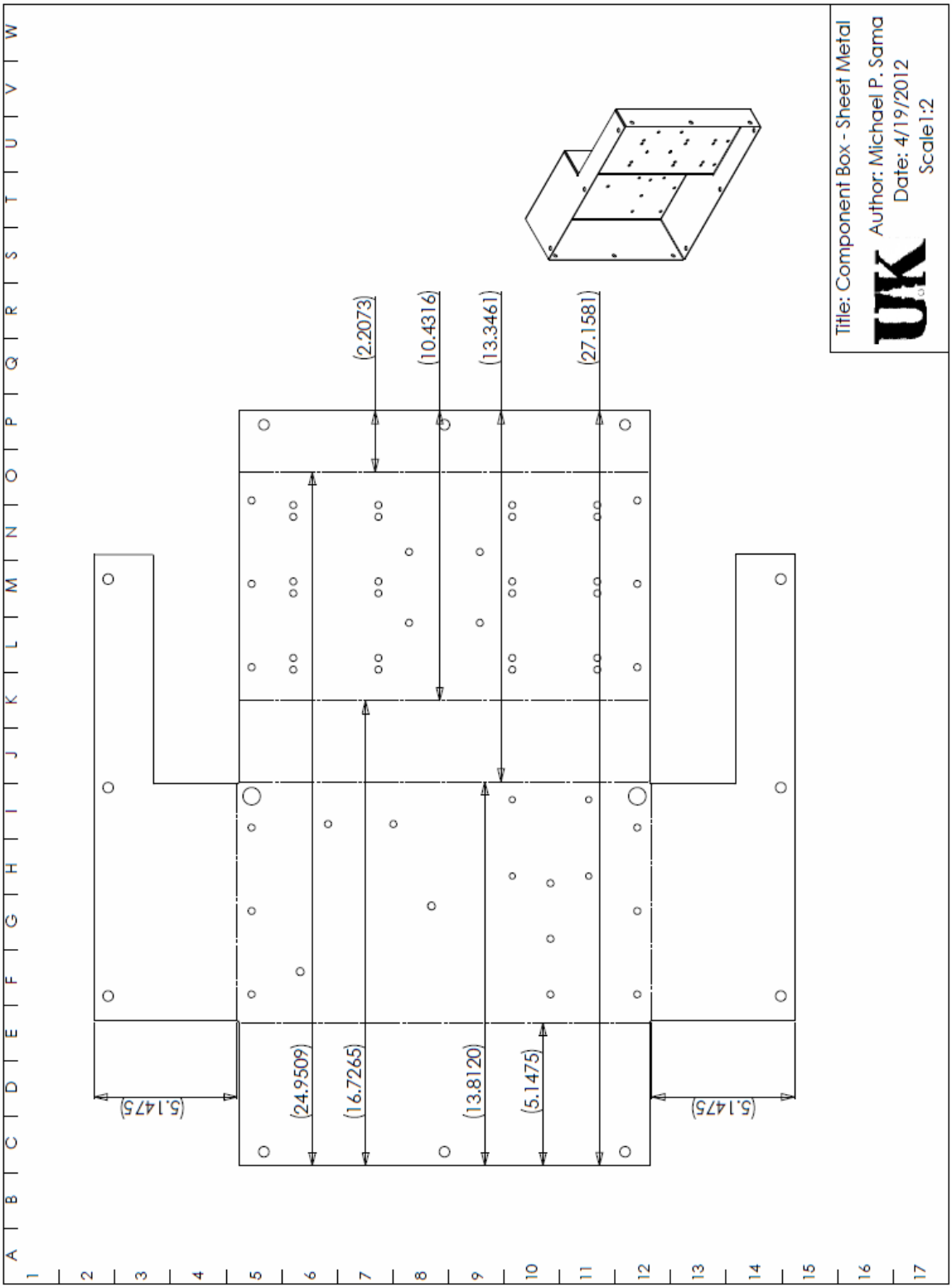
6.2 Frame Assembly



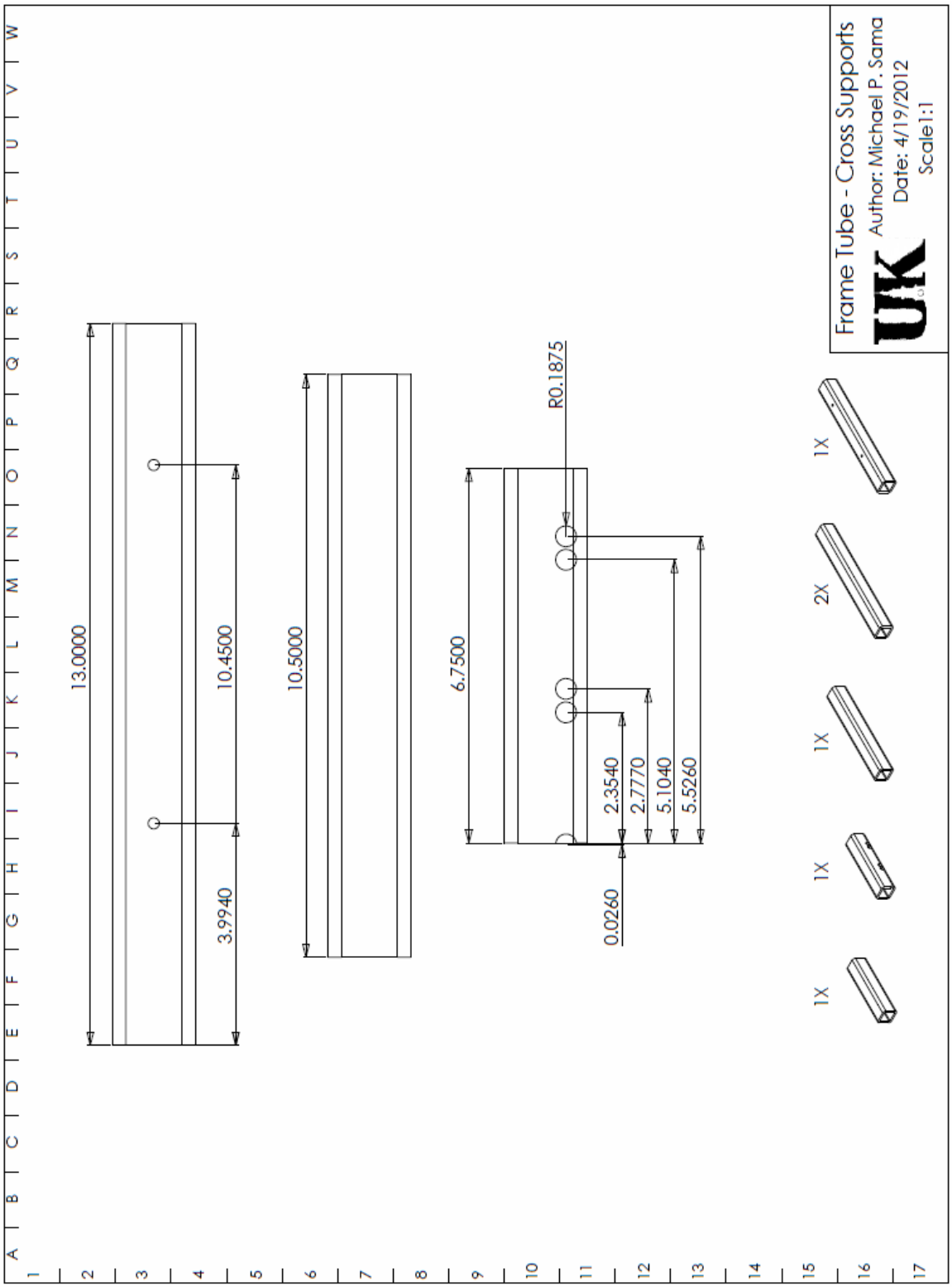
6.3 Cover Panels



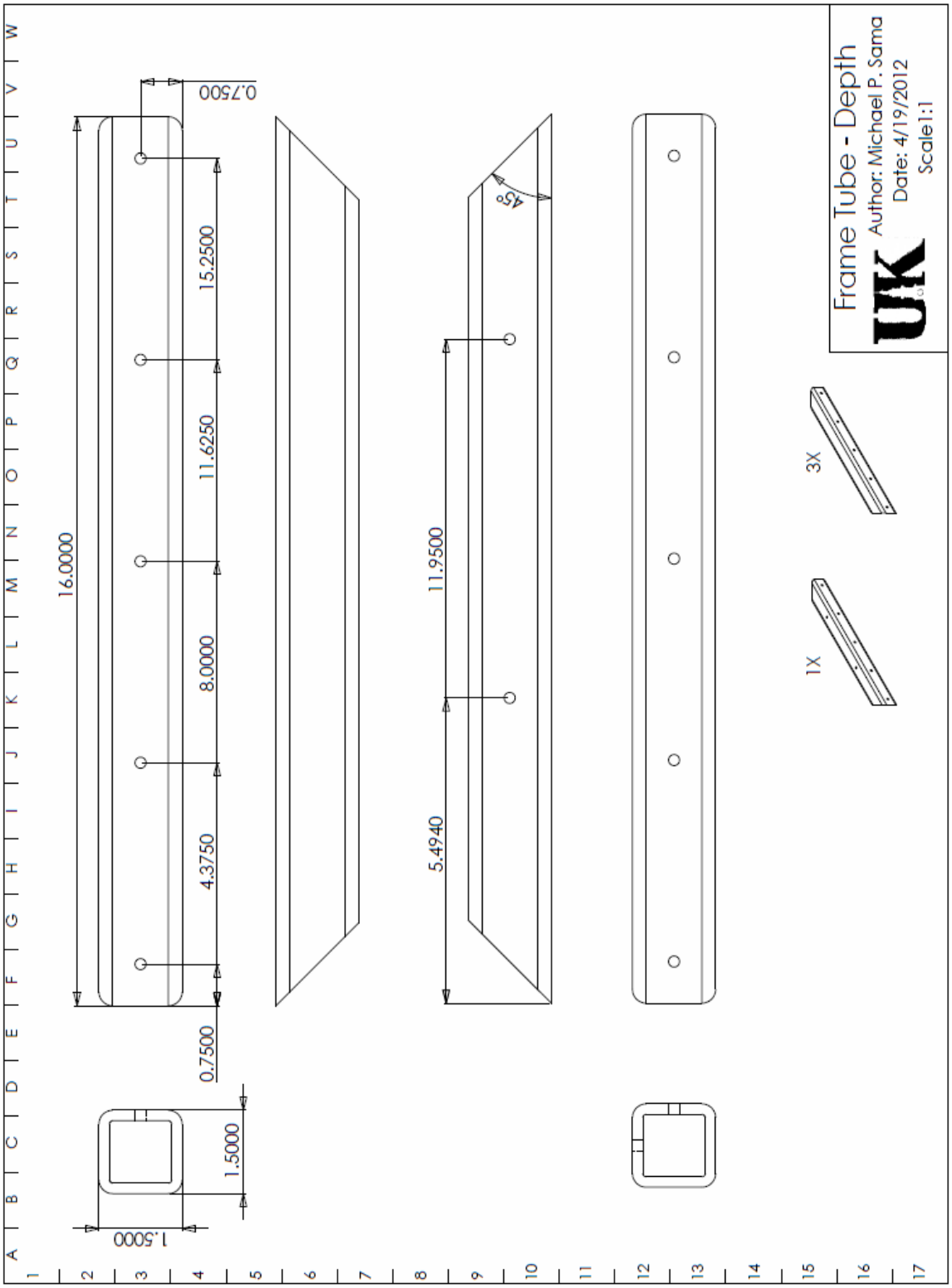
6.4 Component Box



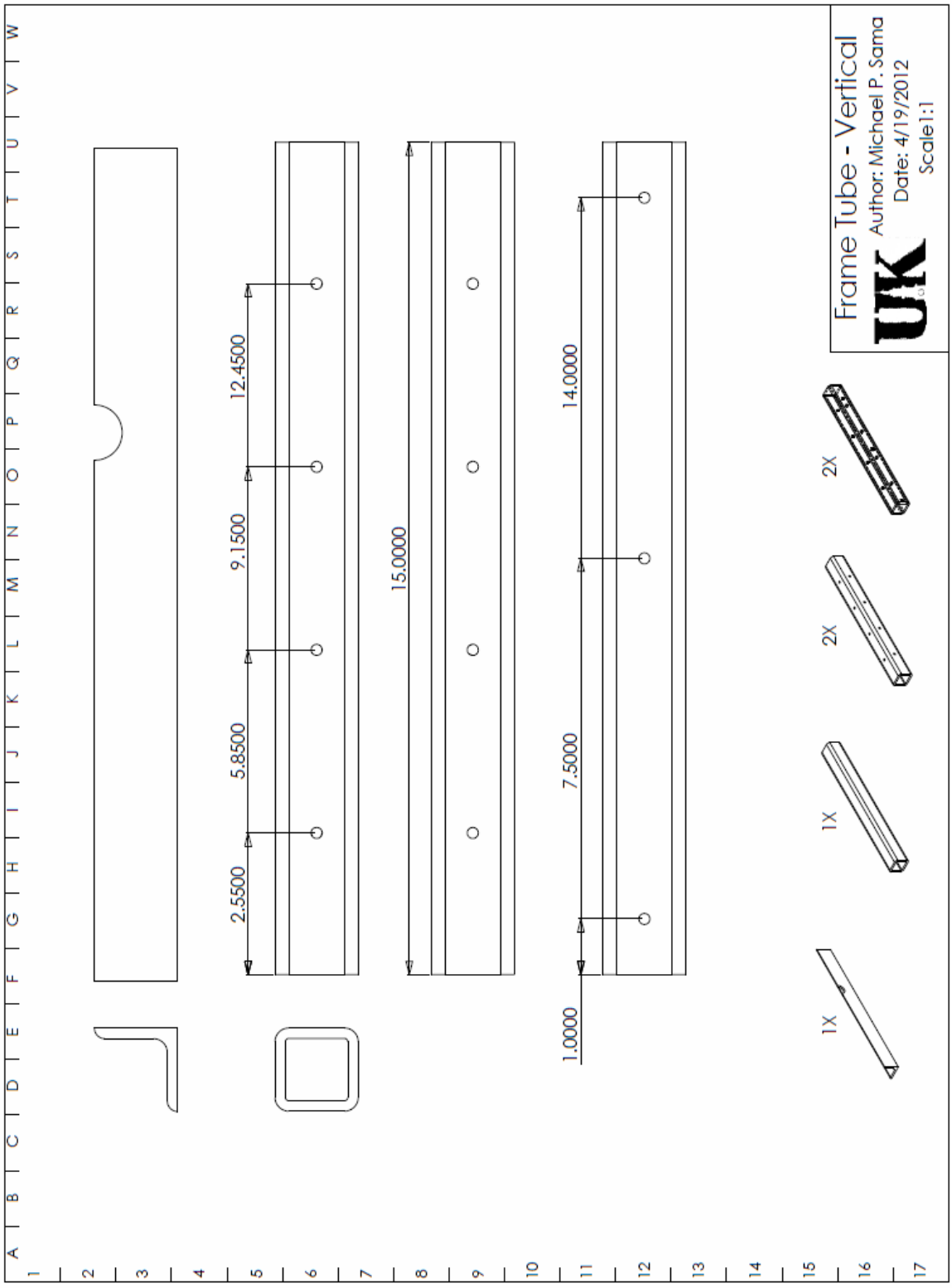
6.5 Frame Tube – Cross Supports



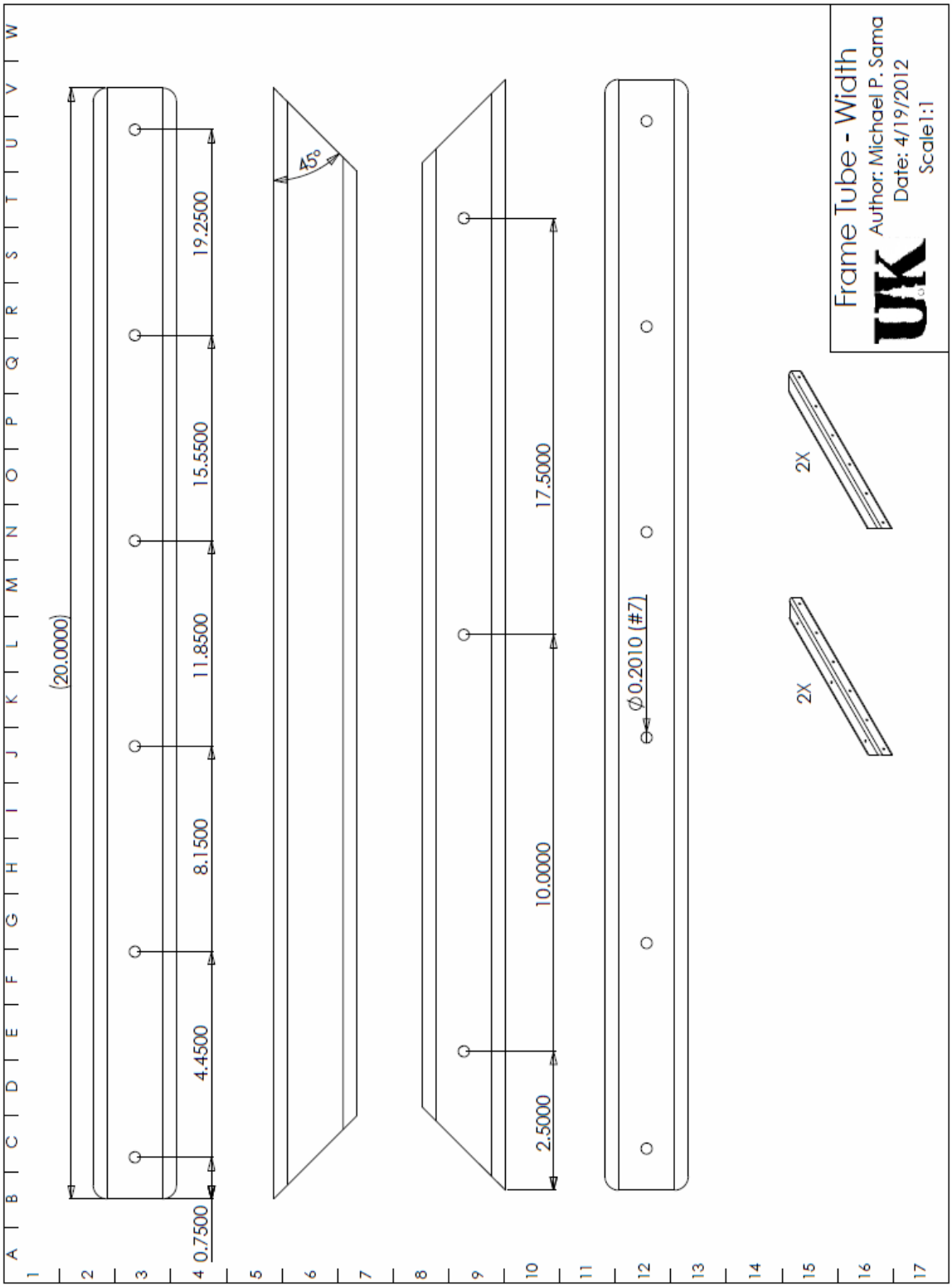
6.6 Frame Tube – Depth



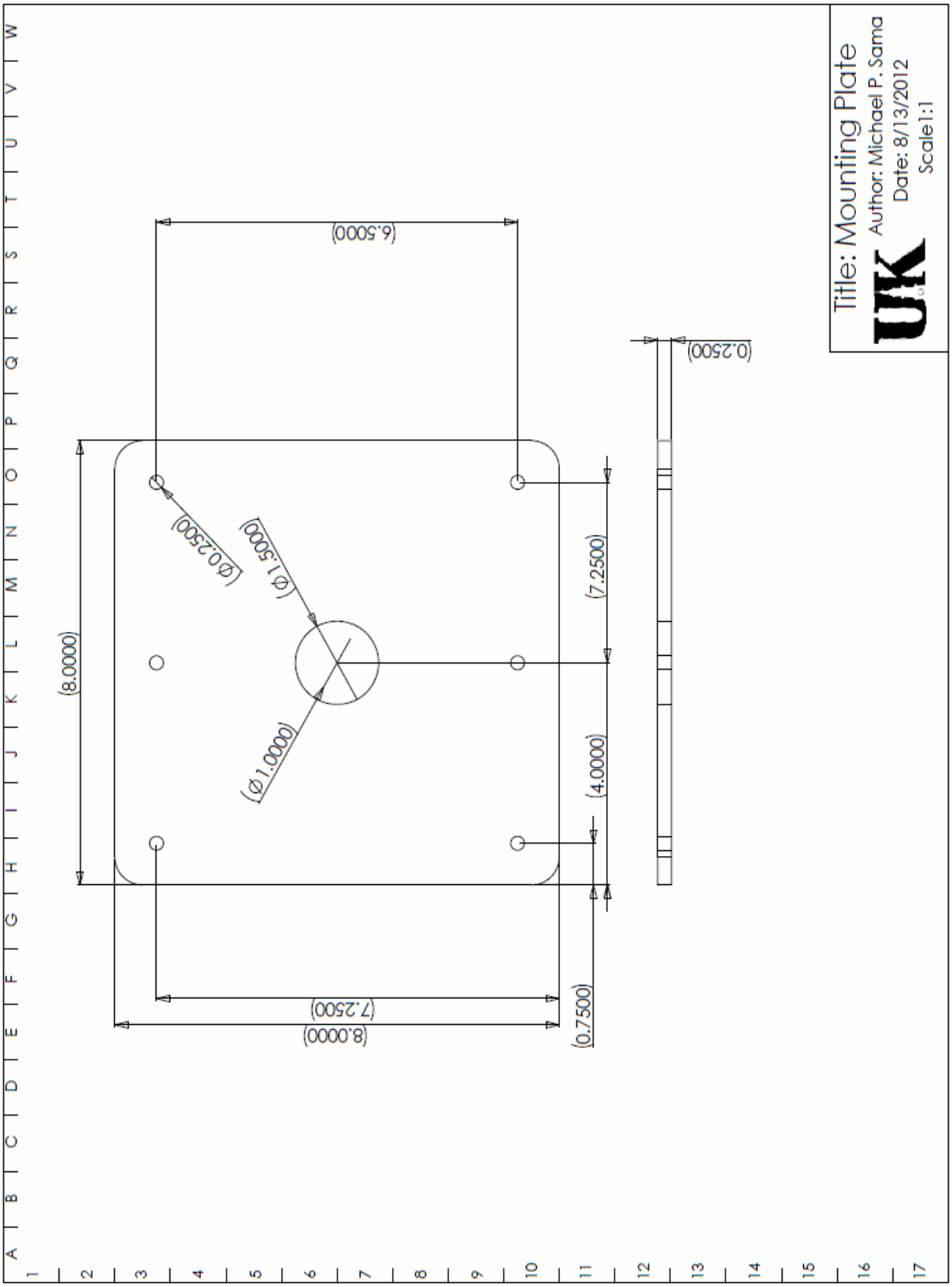
6.7 Frame Tube – Vertical



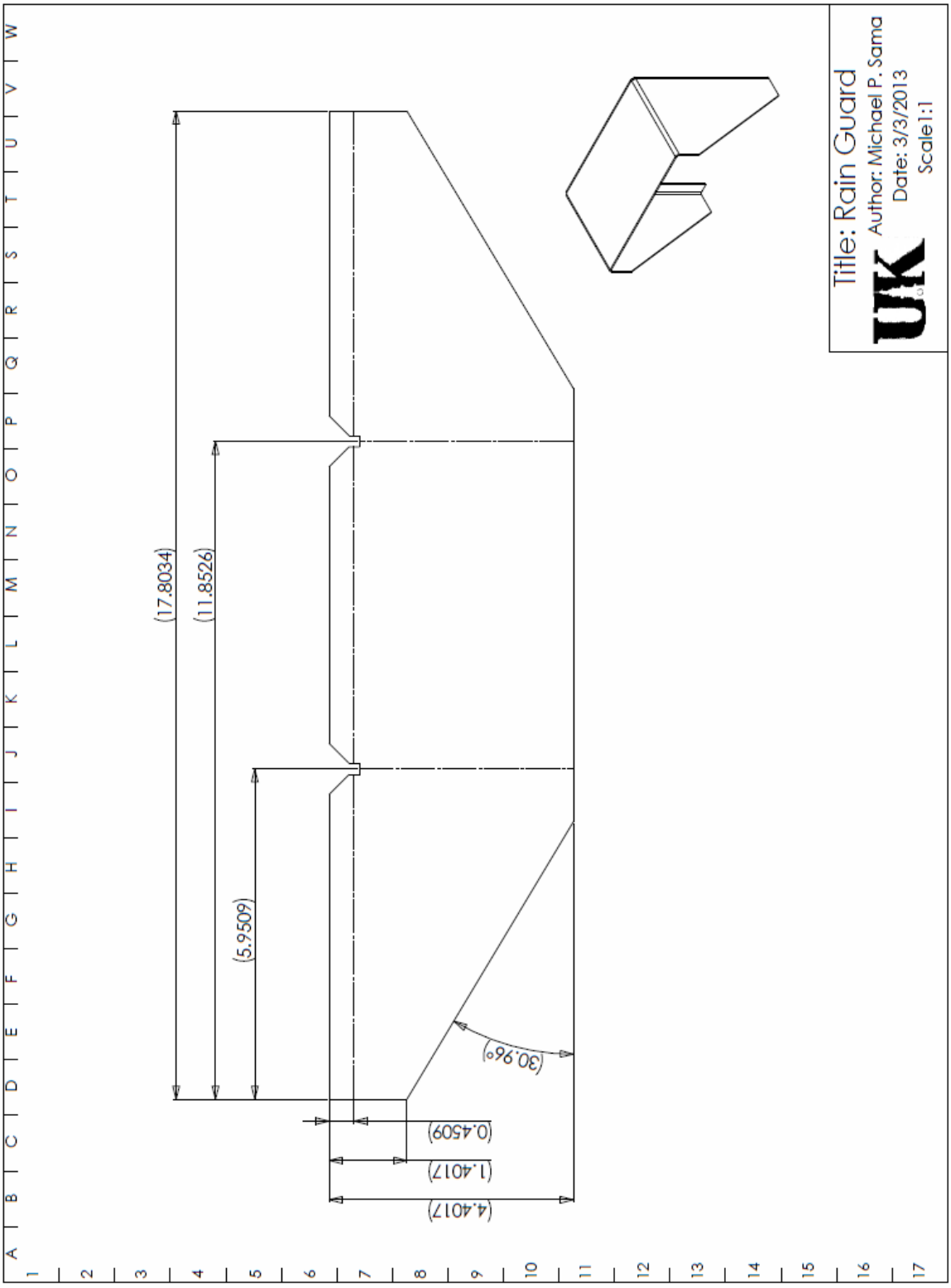
6.8 Frame Tube – Width



6.9 Mounting Plate



6.10 Rain Guard



Appendix 7: TTS Testing Analysis Scripts

7.1 TTS Latency Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Title: TTSanalysis.m                                                  %
%   Author: Michael P. Sama (c) 2012                                      %
%   Date: 8/25/12                                                         %
%   Function: This script reads in a TTS data file and                    %
%             calculates the latency of TTS measurements.                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear; %clear all variables
clc; %clear the command window

%read in a data file and sort the columns
DATA = csvread('C:\Users\michael.sama.BAE-UK\Dropbox\PhD
Project\TTSTestData\Tripod\Test13_3071.csv',1,0);
Time = DATA(:,1);
Ang = DATA(:,2);
xFIX = DATA(:,3);
yFIX = DATA(:,4);
xTTS = DATA(:,5);
yTTS = DATA(:,6);
zTTS = DATA(:,7);
xERR = DATA(:,8);
yERR = DATA(:,9);

%calculate the change in fixture time per sample
dTime = zeros(length(Time)-1,1);
for i=1:length(Time)-1
    if (Time(i+1) > Time(i))
        dTime(i) = Time(i+1)-Time(i);
    else
        dTime(i) = 65536-Time(i) + Time(i+1);
    end
end
dTime = dTime / 58593.75;

%calculate the change in fixture angle per sample
dAng = zeros(length(Ang)-1,1);
for i=1:length(Ang)-1
    if (Ang(i+1) > Ang(i))
        dAng(i) = Ang(i+1)-Ang(i);
    else
        dAng(i) = 10000-Ang(i) + Ang(i+1);
    end
end
dAng = dAng * 2*pi / 10000;

%calculate the velocity
k = 2:length(Time);
Vel = dAng./dTime;
```

```
%plot the velocity versus sample
figure(1)
plot(k,Vel);
xlabel('Sample')
ylabel('Angular Velocity (rad/s)')

%plot the fixture angle versus sample
figure(2)
AngTTS = atan2(yTTS,xTTS);
for i=1:length(AngTTS)
    if (AngTTS(i) < 0)
       AngTTS(i) = 2*pi+AngTTS(i);
    end
end
fAng = Ang* 2*pi / 10000;
plot(1:256,fAng,1:256,AngTTS)
axis([130,150,0,2*pi]);

%plot the difference between the TTS angle and fixture angle
eAng = zeros(size(fAng));
for i=1:length(fAng)
    if (fAng(i) > AngTTS(i))
        eAng(i) = fAng(i) - AngTTS(i);
    else
        eAng(i) = 2*pi - AngTTS(i) + fAng(i);
    end
end

%plot the latency
figure(3)
delay = zeros(size(Vel));
for i=1:length(Vel)
    delay(i) = eAng(i+1) / Vel(i);
end
plot(delay)

%plot the X measurements of the TTS and fixture versus sample
figure(4)
plot(1:256,xTTS,1:256,xFIX,[1,256],[0,0])
axis([37,38,-0.65,0.65]);

%plot a histogram of the TTS latency
figure(5)
hist(delay,16)

%plot the X/Y measurements of the TTS and fixture
figure(6)
plot(xFIX,yFIX,'X',xTTS,yTTS,'+')
axis square

%calculate the mean latency and standard deviation of latency
MeanDelay = mean(delay)
StdDelay = std(delay)
xMSE = mean(abs(xERR))
```

```
yMSE = mean(abs(yERR))

A = mean(Vel)
V = mean(Vel)*0.635
```

7.2 TTS Interpolation Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Title: TTSinterpolate.m                                               %
%   Author: Michael P. Sama (c) 2012                                      %
%   Date: 8/30/12                                                         %
%   Function: This script reads in a TTS data file and                    %
%             calculates the latency of TTS measurements.                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear; %clear all variables
clc; %clear the command window

%read in a data file and sort the columns
DATA = csvread('C:\Users\michael.sama.BAE-UK\Dropbox\PhD
Project\TTSTestData\Far\Test4_0767.csv',1,0);
Time = DATA(:,1);
Ang = DATA(:,2);
xFIX = DATA(:,3);
yFIX = DATA(:,4);
xTTS = DATA(:,5);
yTTS = DATA(:,6);
zTTS = DATA(:,7);
xERR = DATA(:,8);
yERR = DATA(:,9);
timePPS = DATA(:,10);
anglePPS = DATA(:,11);

%calculate the cumulative time
accTime = zeros(length(Time),1);
accTime(1) = 0;
for i=2:length(Time)
    if (Time(i) > Time(i-1))
        accTime(i) = accTime(i-1) + (Time(i) - Time(i-1))/58593.75;
    else
        accTime(i) = accTime(i-1) + (65536-Time(i-1) +
Time(i))/58593.75;
    end
end

%calculate the change in fixture time between samples
dTime = zeros(length(Time)-1,1);
for i=1:length(Time)-1
    if (Time(i+1) > Time(i))
        dTime(i) = Time(i+1)-Time(i);
    else
        dTime(i) = 65536-Time(i) + Time(i+1);
    end
```

```
end
dTime = dTime / 58593.75;

%calculate the change in fixture angle between samples
dAng = zeros(length(Ang)-1,1);
for i=1:length(Ang)-1
    if (Ang(i+1) > Ang(i))
        dAng(i) = Ang(i+1)-Ang(i);
    else
        dAng(i) = 10000-Ang(i) + Ang(i+1);
    end
end
dAng = dAng * 2*pi / 10000;

%calculate the velocity at each sample
k = 2:length(Time);
Vel = dAng./dTime;

%calculate the TTS angle
AngTTS = atan2(yTTS,xTTS);
for i=1:length(AngTTS)
    if (AngTTS(i) < 0)
       AngTTS(i) = 2*pi+AngTTS(i);
    end
end

%calculate the error between the TTS and fixture angles
fAng = Ang* 2*pi / 10000;
eAng = zeros(size(fAng));
for i=1:length(fAng)
    if (fAng(i) > AngTTS(i))
        eAng(i) = fAng(i) - AngTTS(i);
    else
        eAng(i) = 2*pi - AngTTS(i) + fAng(i);
    end
end

%calculate the TTS measurement latency
delay = zeros(size(Vel));
for i=1:length(Vel)
    delay(i) = eAng(i+1) / Vel(i);
end

%compensate for TTS measurement latency
accTime = accTime - mean(delay);

%remove any redundant PPS timestamps
ind = zeros(length(timePPS),1);
for i=2:length(timePPS)
   if (timePPS(i) == timePPS(i-1))
       ind(i) = i;
   end
end
ind = find(ind~=0);
timePPS(ind) = [];
```

```
anglePPS(ind) = [];

%convert timestamps from a 16-bit number to seconds
accPPStime = zeros(length(timePPS),1);
if (timePPS(1) < Time(1))
    accPPStime(1) = (timePPS(1)-Time(1))/58593.75;
else
    accPPStime(1) = -1*(65536-timePPS(1)+Time(1))/58593.75;
end
for i=2:length(timePPS)
   if (timePPS(i) > timePPS(i-1))
      accPPStime(i) = accPPStime(i-1) + (timePPS(i)-timePPS(i-
1))/58593.75;
   else
       accPPStime(i) = accPPStime(i-1) + (65536-timePPS(i-
1)+timePPS(i))/58593.75;
   end
end

%calculate the x and y position of the fixture at each PPS event
xPPS = zeros(length(anglePPS),1);
yPPS = zeros(length(anglePPS),1);
for i=1:length(anglePPS)
    xPPS(i) = 0.635*cosd(360*anglePPS(i)/10000);
    yPPS(i) = 0.635*sind(360*anglePPS(i)/10000);
end


%interpolate fixture measurements at the PPS interval to the TTS
interval
%and plot as a video
xINT = zeros(length(timePPS),1);
yINT = zeros(length(timePPS),1);
for i=1:length(accPPStime)
    tPPS = accPPStime(i);
    [~,I] = min(abs(accTime-tPPS));
    if (accTime(I)>tPPS)
        k = I;
    else
        k = I+1;
    end
    if (k < 3)
        k = 3;
    elseif (k > length(accTime)-1)
        k = length(accTime) - 1;
    end

    xCAL = [accTime(k-2)^3,accTime(k-2)^2,accTime(k-2),1; ...
        accTime(k-1)^3,accTime(k-1)^2,accTime(k-1),1; ...
        accTime(k)^3,accTime(k)^2,accTime(k),1; ...
        accTime(k+1)^3,accTime(k+1)^2,accTime(k+1),1]^-1 ...
        * [xTTS(k-2);xTTS(k-1);xTTS(k);xTTS(k+1)];
    yCAL = [accTime(k-2)^3,accTime(k-2)^2,accTime(k-2),1; ...
        accTime(k-1)^3,accTime(k-1)^2,accTime(k-1),1; ...
        accTime(k)^3,accTime(k)^2,accTime(k),1; ...
        accTime(k+1)^3,accTime(k+1)^2,accTime(k+1),1]^-1 ...
```

```
        * [yTTS(k-2);yTTS(k-1);yTTS(k);yTTS(k+1)];
    figure(1)
    t = accTime(k-2):1/58593.75:accTime(k+1);
    fxINT = xCAL(1)*t.^3+xCAL(2)*t.^2+xCAL(3)*t+xCAL(4);
    fyINT = yCAL(1)*t.^3+yCAL(2)*t.^2+yCAL(3)*t+yCAL(4);
    [~,I] = min(abs(t-accPPStime(i)));
    xINT(i) = fxINT(I);
    yINT(i) = fyINT(I);
    subplot(2,1,1);
    plot(accTime,xTTS,'Xk',t,fxINT,'r',accPPStime,xPPS,'o')
    axis([accPPStime(i)-10,accPPStime(i)+10,-0.8,0.8])
    xlabel('Time (s)');
    ylabel('X Position (m)');
    subplot(2,1,2);
    plot(accTime,yTTS,'Xk',t,fyINT,'r',accPPStime,yPPS,'o')
    axis([accPPStime(i)-10,accPPStime(i)+10,-0.8,0.8])
    xlabel('Time (s)');
    ylabel('Y Position (m)');
    F(i) = getframe(gcf);
end
for k = 0:length(xTTS)-4
xCAL = [accTime(1+k)^3,accTime(1+k)^2,accTime(1+k),1; ...
        accTime(2+k)^3,accTime(2+k)^2,accTime(2+k),1; ...
        accTime(3+k)^3,accTime(3+k)^2,accTime(3+k),1; ...
        accTime(4+k)^3,accTime(4+k)^2,accTime(4+k),1]^-1 ...
        * [xTTS(1+k);xTTS(2+k);xTTS(3+k);xTTS(4+k)];
figure(1)
t = accTime(1+k):0.01:accTime(4+k);
xINT = xCAL(1)*t.^3+xCAL(2)*t.^2+xCAL(3)*t+xCAL(4);
plot(accTime,xTTS,'Xk',t,xINT,'k')
axis([accTime(1+k)-10,accTime(4+k)+10,-0.8,0.8])
F(k+1) = getframe(gcf);
end
figure(2)
movie(F)

%calcualte the error between the interpolated measurements and the
fixture
xError = abs(xPPS-xINT);
xError(1) = [];
xError(length(xError)) = [];
yError = abs(yPPS-yINT);
yError(1) = [];
yError(length(yError)) = [];
xMeanError = mean(xError)
xStdError = std(xError)
yMeanError = mean(yError)
yStdError = std(yError)

%plot the error
figure(2)
subplot(2,1,1)
hist(xError,10)
xlabel('Error (m)')
ylabel('X Count')
subplot(2,1,2)
```

```
hist(yError,10)
xlabel('Error (m)')
ylabel('Y Count')
```

7.3 Sample Data

Time, Angle, xFixture, yFixture, xTTS, yTTS, zTTS, xError, yError, PPStime, PPSangle

34918, 6767,-0.2822242 ,-0.5688361,-0.372631,-0.513605,0.002079,0.09040681,-0.05523109, 54309,6212

58452,7048,-0.1779255,-0.6095634,-0.277994,-0.570675,0.002762,0.1000685,-0.03888839,48155,6924

16229,7327,-0.06888809,-0.6312523,-0.172593,-0.608698,0.003306,0.1037049,-0.02255428,48155,6924

39509,7606,0.04226086,-0.6335921,-0.062953,-0.628411,0.003557,0.1052139,-0.005181134,48155,6924

63864,7898,0.1571451,-0.6152483,0.048492,-0.628828,0.004013,0.1086531,0.01357973,41990,7636

21804,8181,0.2634916,-0.5777519,0.158394,-0.60948,0.004241,0.1050976,0.03172815,41990,7636

45078,8460,0.3602158,-0.5229432,0.264929,-0.570329,0.004562,0.09528679,0.04738581,35826,8348

2210,8731,0.4436206,-0.454341,0.361479,-0.513876,0.004648,0.08214158,0.059535,35826,8348

25659,9012,0.5165253,-0.3693597,0.447105,-0.441513,0.0046,0.06942034,0.07215333,35826,8348

49851,9301,0.5747349,-0.2700088,0.517332,-0.355686,0.004561,0.05740291,0.08567724,29668,9061

7867,9584,0.6134317,-0.1640932,0.572205,-0.258604,0.004221,0.04122669,0.09451084,29668,9061

31319,9868,0.6328173,-0.0526053,0.609962,-0.153314,0.004343,0.02285528,0.1007087,23500,9774

54790,150,0.6321818,0.05975878,0.627016,-0.043225,0.004148,0.005165815,0.1029838,23500,9774

12770,433,0.611644,0.170636,0.627126,0.069272,0.003802,-0.01548201,0.101364,23500,9774

36013,713,0.5723376,0.2750539,0.605549,0.179042,0.00342,-0.03321135,0.09601192,17349,489

59569,996,0.5146622,0.3719513,0.563584,0.282776,0.002885,-0.04892182,0.08917531,17349,489

17464,1278,0.4410443,0.4568424,0.505985,0.378005,0.002317,-0.06494075,0.07883736,11182,1203

40759,1560,0.353616,0.5274284,0.431908,0.460707,0.001836,-0.07829198,0.06672138,11182,1203

64302,1845,0.2540185,0.581979,0.343135,0.530093,0.001307,-0.08911648,0.05188602,11182,1203

22109,2126,0.1478498,0.6175479,0.244575,0.581947,0.00048,-0.09672518,0.0356009,5026,1919

45619,2408,0.03668593,0.6339394,0.138086,0.616589,-2E-05,-0.1014001,0.01735038,5026,1919

2833,2682,-0.07245661,0.6308526,0.028085,0.630991,-0.000596,-0.1005416,-0.0001383424,64402,2634

26915,2972,-0.1855712,0.6072794,-0.081561,0.627076,-0.000911,-0.1040102,-0.01979661,64402,2634

49741,3249,-0.2879284,0.5659702,-0.193223,0.60164,-0.001292,-0.0947054,-0.0356698,64402,2634

7717,3534,-0.3841323,0.5056356,-0.296837,0.557848,-0.001682,-0.08729526,-0.05221236,58234,3352

31021,3816,-0.4672416,0.430012,-0.391923,0.496717,-0.001845,-0.07531855,-0.06670502,58234,3352

55364,4109,-0.5380637,0.3372127,-0.473307,0.419951,-0.001728,-0.06475669,-0.08273825,52079,4069

12387,4380,-0.587424,0.2411597,-0.541136,0.33035,-0.00159,-0.04628801,-0.08919029,52079,4069

Appendix 8: GNSS Testing Analysis Scripts

8.1 GNSS Testing Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Title: GPSanalysis.m                                                  %
%   Author: Michael P. Sama (c) 2012                                      %
%   Date: 12/7/12                                                         %
%   Function: This script reads in a series of GNSS tests   %
%             and calcualtes X/Y, along-, and off-track     %
%             errors.                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear; %clear all variables
clc; %clear the command window
format long; %display all significant figures

wb = waitbar(0,'Reading List File...'); %progress bar for monitoring
status
folder = 'C:\Users\michael.sama.BAE-UK\Dropbox\PhD
Project\GPSTestData\Filter Max\'; %folder containing GNSS data files
fid = fopen([folder,'FileList.txt']); %open the file containing a list
of GNSS data files to process
txt = textscan(fid,'%s'); %read all filenames to process
ID = txt{1}; %sort by lines
fclose(fid); %close the file
samples = length(ID); %number of data files to read
%*** Initialize Data Variables ****
FileList = {zeros(samples),1}; FileList = FileList';
Speeds = zeros(samples,1);
MobileData = {zeros(samples),1}; MobileData = MobileData';
MobileLat = {zeros(samples),1}; MobileLat = MobileLat';
MobileLon = {zeros(samples),1}; MobileLon = MobileLon';
MobileX = {zeros(samples),1}; MobileX = MobileX';
MobileY = {zeros(samples),1}; MobileY = MobileY';
MobileFix = {zeros(samples),1}; MobileFix = MobileFix';
StaticData = {zeros(samples),1}; StaticData = StaticData';
StaticLat = {zeros(samples),1}; StaticLat = StaticLat';
StaticLon = {zeros(samples),1}; StaticLon = StaticLon';
StaticX = {zeros(samples),1}; StaticX = StaticX';
StaticY = {zeros(samples),1}; StaticY = StaticY';
FixtureAng = {zeros(samples),1}; FixtureAng = FixtureAng';
FixtureX = {zeros(samples),1}; FixtureX = FixtureX';
FixtureY = {zeros(samples),1}; FixtureY = FixtureY';
ErrorX = {zeros(samples),1}; ErrorX = ErrorX';
ErrorY = {zeros(samples),1}; ErrorY = ErrorY';
ErrorA = {zeros(samples),1}; ErrorA = ErrorA';
ErrorO = {zeros(samples),1}; ErrorO = ErrorO';
ErrorOstd = zeros(samples,1);
ErrorAstd = zeros(samples,1);
ErrorSstd = zeros(samples,1);

%*** Read all data files listed in the 'FileList.txt' document ***
j = 1;
```

```
for i=1:samples
    waitbar(i/samples,wb,['Reading Raw Data File ',num2str(i),'...']);
    FileList(j) = {[char(ID(i)),'_Mobile.csv']};
    FileList(j+1) = {[char(ID(i)),'_Static.csv']};
    [~,~,MobileData{i}] = xlsread([folder,char(FileList(j))]);
    [~,~,StaticData{i}] = xlsread([folder,char(FileList(j+1))]);
    j = j + 2;
end

%*** Extract the fixture speed from each file ***
FileElements = regexp(FileList,'_','split');
j = 1;
for i=1:samples
    X = FileElements{j};
    Speeds(i) = str2double(cell2mat(X(3)));
    j = j + 2;
end

%*** Extract latitudes, longitudes, and fixture angle ***
for i=1:samples
    waitbar(i/samples,wb,['Extracting Positions From File
',num2str(i),'...']);
    X = MobileData{i};
    MobileFix{i} = X(:,7);
    I = find(cell2mat(MobileFix{i})==4);
    MobileLat{i} = X(I,3);
    MobileLon{i} = X(I,5);
    FixtureAng{i} = X(I,18);
    Y = StaticData{i};
    StaticLat{i} = Y(:,3);
    StaticLon{i} = Y(:,5);
end
clear X
clear Y

%*** Convert latitudes and longitudes from DDMM.MM format to D.DD ***
for i=1:samples
    waitbar(i/samples,wb,['Converting Positions From File
',num2str(i),'...']);
    X = cell2mat(MobileLat{i});
    Y = cell2mat(MobileLon{i});
    Z = cell2mat(StaticLat{i});
    W = cell2mat(StaticLon{i});
    MobileLat{i} = (floor(X/100)+mod(X,100)/60);
    MobileLon{i} = -1*(floor(Y/100)+mod(Y,100)/60);
    StaticLat{i} = (floor(Z/100)+mod(Z,100)/60);
    StaticLon{i} = -1*(floor(W/100)+mod(W,100)/60);
end
clear X
clear Y
clear Z
clear W

%*** Transform latitdues and longitudes to local coordinate system ***
for i=1:samples
```

```
   waitbar(i/samples,wb,['Transforming Positions From File
',num2str(i),'...']);
   refLat = 38.027003344143857;
   refLon = -84.509631727984825;
%  refLat = mean(MobileLat{i});
%  refLon = mean(MobileLon{i});
   Phi = refLat * pi/180;  %Convert the latitude from degrees to
radians
   h = -33.022;    %Set the height above the elipsoid
   a = 6378137;    %Set the semimajor-axis of the elipsoid
   b = 6356752.3142;   %Set the semiminor-axis of the elipsoid
   Flat = (pi / 180) * (((a ^ 2 * b ^ 2) / ((a ^ 2 * cos(Phi) *
cos(Phi) + b ^ 2 * sin(Phi) * sin(Phi)) ^ (3 / 2))) + h);
   Flon = (pi / 180) * ((a ^ 2 / sqrt(a ^ 2 * cos(Phi) * cos(Phi) + b ^
2 * sin(Phi) * sin(Phi))) + h) * cos(Phi);
   MobileX{i} = (MobileLon{i}-refLon) .* Flon;
   MobileY{i} = (MobileLat{i}-refLat) .* Flat;
   refLat = mean(StaticLat{i});
   refLon = mean(StaticLon{i});
   Phi = refLat * pi/180;  %Convert the latitude from degrees to
radians
   h = -33.022;    %Set the height above the elipsoid
   a = 6378137;    %Set the semimajor-axis of the elipsoid
   b = 6356752.3142;   %Set the semiminor-axis of the elipsoid
   Flat = (pi / 180) * (((a ^ 2 * b ^ 2) / ((a ^ 2 * cos(Phi) *
cos(Phi) + b ^ 2 * sin(Phi) * sin(Phi)) ^ (3 / 2))) + h);
   Flon = (pi / 180) * ((a ^ 2 / sqrt(a ^ 2 * cos(Phi) * cos(Phi) + b ^
2 * sin(Phi) * sin(Phi))) + h) * cos(Phi);
   StaticX{i} = (StaticLon{i}-refLon) .* Flon;
   StaticY{i} = (StaticLat{i}-refLat) .* Flat;
end

%*** Remove the mean error for all dynamic tests ***
for i=2:samples
   MobileX{i} = MobileX{i} - mean(MobileX{i});
   MobileY{i} = MobileY{i} - mean(MobileY{i});
end

%*** Caculate the fixture position at each sample and X/Y position
error **
for i=1:samples
   A = cell2mat(FixtureAng{i});
   FixtureX{i} = 1.*cos(A./10000.*2.*pi-3.736); %3.726 Test 1, %3.755
5800
   FixtureY{i} = 1.*sin(A./10000.*2.*pi-3.736);
   ErrorX{i} = MobileX{i} - FixtureX{i};
   ErrorY{i} = MobileY{i} - FixtureY{i};
end

%*** Calcuate along-, off-track errors, and standard deviations ***
for i=1:samples
    waitbar(i/samples,wb,['Calculating Off- and Along-Track Errors From
File ',num2str(i),'...']);
    A = -1*cell2mat(FixtureAng{i});
    EX = ErrorX{i};
    EY = ErrorY{i};
```

```
    ESX = StaticX{i};
    ESY = StaticY{i};
    EV = sqrt(ESX.^2+ESY.^2);
    X = zeros(length(EX),1);
    Y = zeros(length(EX),1);
    for j = 1:length(EX)
        XY = [cos(A(j)),-sin(A(j));sin(A(j)),cos(A(j))]*[EX(j);EY(j)];
        X(j) = XY(1);
        Y(j) = XY(2);
        ErrorO{i} = X;
        ErrorA{i} = Y;
        ErrorOstd(i) = std(X);
        ErrorAstd(i) = std(Y);
        ErrorSstd(i) = std(EV);
    end
end

%*** Plot the X and Y positions ***
figure(1)
for i = 1:samples
   subplot(1,samples,i);
   plot(MobileX{i},MobileY{i},'.',FixtureX{i},FixtureY{i},'.')
   axis([-1.1,1.1,-1.1,1.1])
   axis square
end

%*** Plot the X and Y errors ***
figure(2)
for i = 1:samples
   subplot(1,samples,i);
   plot(ErrorX{i},ErrorY{i},'.')
   axis([-0.2,0.2,-0.2,0.2])
   axis square
end

%*** Plot the along- and off-track errors ***
figure(3)
for i = 1:samples
   subplot(1,samples,i);
   plot(ErrorO{i},ErrorA{i},'.')
   axis([-0.2,0.2,-0.2,0.2])
   axis square
end

%*** Plot the X and Y positions of a static receiver ***
figure(4)
for i = 1:samples
   subplot(1,samples,i);
   plot(StaticX{i},StaticY{i},'.')
   axis([-0.2,0.2,-0.2,0.2])
   axis square
end

%*** Plot the along- and off-track standard deviations of error ***
figure(5)
plot(Speeds,ErrorOstd,'.',Speeds,ErrorAstd,'.')
```

```
axis([500,2750,0,0.05]);

close(wb) %closs the progress bar
```

8.2 Sample Data

$GPGGA,184338.00,3801.61965289,N,08430.57795676,W,4,07,2.1,316.958,M,-33.022,M,1.0,0000*7D,$PPS,24142,3362

$GPGGA,184339.00,3801.61978985,N,08430.57743988,W,4,07,2.1,316.959,M,-33.022,M,1.0,0000*73,$PPS,17202,4675

$GPGGA,184340.00,3801.62018922,N,08430.57722149,W,4,07,2.1,316.928,M,-33.022,M,1.0,0000*78,$PPS,10263,5986

$GPGGA,184341.00,3801.62059377,N,08430.57743916,W,4,08,1.4,316.909,M,-33.022,M,1.0,0000*79,$PPS,3323,7295

$GPGGA,184342.00,3801.62073047,N,08430.57795595,W,4,07,2.1,316.946,M,-33.022,M,1.0,0000*7C,$PPS,61920,8601

$GPGGA,184343.00,3801.62052491,N,08430.57843994,W,4,07,2.1,316.985,M,-33.022,M,1.0,0000*77,$PPS,54981,9907

$GPGGA,184344.00,3801.62011484,N,08430.57858229,W,4,07,2.1,316.998,M,-33.022,M,1.0,0000*78,$PPS,48041,1216

$GPGGA,184345.00,3801.61973996,N,08430.57829877,W,4,07,2.1,317.013,M,-33.022,M,1.0,0000*75,$PPS,41102,2525

$GPGGA,184346.00,3801.61966301,N,08430.57776146,W,4,07,2.1,316.988,M,-33.022,M,1.0,0000*72,$PPS,34163,3838

$GPGGA,184347.00,3801.61992105,N,08430.57731923,W,4,07,2.1,316.938,M,-33.022,M,1.0,0000*7D,$PPS,27223,5152

$GPGGA,184348.00,3801.62034828,N,08430.57725879,W,4,08,1.4,316.945,M,-33.022,M,1.0,0000*7A,$PPS,20284,6463

$GPGGA,184349.00,3801.62068164,N,08430.57760665,W,4,08,1.4,316.931,M,-33.022,M,1.0,0000*72,$PPS,13345,7772

$GPGGA,184350.00,3801.62069023,N,08430.57815791,W,4,07,2.1,316.950,M,-33.022,M,1.0,0000*70,$PPS,6405,9077

$GPGGA,184351.00,3801.62038226,N,08430.57854170,W,4,07,2.1,316.979,M,-33.022,M,1.0,0000*75,$PPS,65002,389

$GPGGA,184352.00,3801.61996054,N,08430.57851770,W,4,07,2.1,316.979,M,-33.022,M,1.0,0000*7C,$PPS,58062,1697

$GPGGA,184353.00,3801.61968180,N,08430.57810494,W,4,07,2.1,316.957,M,-33.022,M,1.0,0000*74,$PPS,51123,3008

$GPGGA,184354.00,3801.61972258,N,08430.57756296,W,4,07,2.1,316.980,M,-33.022,M,1.0,0000*7D,$PPS,44184,4322

$GPGGA,184355.00,3801.62007118,N,08430.57724087,W,4,07,2.1,316.949,M,-33.022,M,1.0,0000*71,$PPS,37244,5635

$GPGGA,184356.00,3801.62048602,N,08430.57733625,W,4,07,2.1,316.988,M,-33.022,M,1.0,0000*70,$PPS,30305,6946

$GPGGA,184357.00,3801.62072150,N,08430.57780736,W,4,07,2.1,316.961,M,-33.022,M,1.0,0000*74,$PPS,23366,8254

$GPGGA,184358.00,3801.62061671,N,08430.57833242,W,4,07,2.1,316.960,M,-33.022,M,1.0,0000*7D,$PPS,16426,9561

$GPGGA,184359.00,3801.62022586,N,08430.57858283,W,4,07,2.1,317.003,M,-33.022,M,1.0,0000*7D,$PPS,9487,871

$GPGGA,184400.00,3801.61982257,N,08430.57840338,W,4,08,1.4,316.992,M,-33.022,M,1.0,0000*7C,$PPS,2548,2181

$GPGGA,184401.00,3801.61965097,N,08430.57789959,W,4,08,1.4,316.974,M,-33.022,M,1.0,0000*75,$PPS,61144,3495

$GPGGA,184402.00,3801.61982299,N,08430.57740706,W,4,08,1.4,316.963,M,-33.022,M,1.0,0000*74,$PPS,54205,4809

$GPGGA,184403.00,3801.62024071,N,08430.57722759,W,4,08,1.4,316.925,M,-33.022,M,1.0,0000*7B,$PPS,47266,6122

$GPGGA,184404.00,3801.62062202,N,08430.57748019,W,4,08,1.4,316.936,M,-33.022,M,1.0,0000*75,$PPS,40326,7430

$GPGGA,184405.00,3801.62073230,N,08430.57801512,W,4,08,1.4,316.926,M,-33.022,M,1.0,0000*78,$PPS,33387,8739

Appendix 9: Application of a Dynamic GNSS Error Model

9.1 P/A versus Off-Rate Error Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Title: ErrorExample.m                                               %
%   Author: Michael P. Sama (c) 2010                                    %
%   Date: 10/8/2012                                                     %
%   Function: This script simulates the effect of dynamic   %
%             GNSS error on an application map.                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear; %clear all variables
clc; %clear the command window

n = 400; %set the size of the error filter kernal (cm)
m = n;

sx0 = 2.54; %standard deviation 0 (cm)
sy0 = 2.54;
sx1 = 10.2; %standard deviation 1 (cm)
sy1 = 10.2;
sx2 = 100; %standard deviation 2 (cm)
sy2 = 100;

mx = floor(n/2)+1; %find the center of the error distribution
my = floor(m/2)+1;

D0 = zeros(m,n); %initialize the error distribution variables
D1 = zeros(m,n);
D2 = zeros(m,n);

%calculate the error distributions using a bivariate normal
distribution
for i=1:n
    for j=1:m
        D0(j,i) = (1/(2*pi()*sx0*sy0))*exp((-1/2)*(((i-mx)/sx0)^2+((j-
my)/sy0)^2));
        D1(j,i) = (1/(2*pi()*sx1*sy1))*exp((-1/2)*(((i-mx)/sx1)^2+((j-
my)/sy1)^2));
        D2(j,i) = (1/(2*pi()*sx2*sy2))*exp((-1/2)*(((i-mx)/sx2)^2+((j-
my)/sy2)^2));
    end
end

%plot the error distributions
figure(1)
subplot(1,3,1)
imagesc(D0)
title('\sigma = 2.54 cm')
xlabel('Off-Track Error (cm)')
ylabel('Along-Track Error (cm)')
colormap gray
axis image
```

```
subplot(1,3,2)
imagesc(D1)
title('\sigma = 10.2 cm')
xlabel('Off-Track Error (cm)')
ylabel('Along-Track Error (cm)')
colormap gray
axis image
subplot(1,3,3)
imagesc(D2)
title('\sigma = 100 cm')
xlabel('Off-Track Error (cm)')
ylabel('Along-Track Error (cm)')
colormap gray
axis image

%*** Use the following line to load a custom application map ***
%S = double(~imread('test.bmp'));

%*** Use the following lines to set the application map as a square ***
% S = zeros(400,400);
% S(100:300,100:300) = 1;

%*** Use the following lines to set the application map as a circle ***
N = 400;
S = zeros(N,N);
d = 200;
x = size(S,1)/2;
y = size(S,2)/2;
for i = 1:size(S,1)
    for j = 1:size(S,2)
        if sqrt((i-x)^2+(j-y)^2) <= d/2
            S(i,j) = 1;
        end
    end
end

%conver the application map to a binary image and display parameters
BW = im2bw(S,graythresh(S));
PAR = regionprops(BW,'area','perimeter');
Area = PAR.Area
Perimeter = PAR.Perimeter

%plot the application map
figure(2)
imagesc(S)
title('Application Map without Position Error')
colormap gray
axis image

%integrate GNSS error using convolution
V0 = conv2(S,D0,'same');
V1 = conv2(S,D1,'same');
V2 = conv2(S,D2,'same');

%plot the application maps with GNSS error
```

```
figure(3)
subplot(1,3,1)
imagesc(V0)
title('\sigma = 2.54 cm')
xlabel('X-Direction (cm)')
ylabel('Y-Direction (cm)')
colormap gray
axis image
subplot(1,3,2)
imagesc(V1)
title('\sigma = 10.2 cm')
xlabel('X-Direction (cm)')
ylabel('Y-Direction (cm)')
colormap gray
axis image
subplot(1,3,3)
imagesc(V2)
title('\sigma = 100 cm')
xlabel('X-Direction (cm)')
ylabel('Y-Direction (cm)')
colormap gray
axis image

%plot the applications maps with GNSS error inside the boundary
figure(4)
Y0 = V0.*S;
Y1 = V1.*S;
Y2 = V2.*S;
subplot(1,3,1)
imagesc(Y0)
title('\sigma = 2.54 cm')
xlabel('X-Direction (cm)')
ylabel('Y-Direction (cm)')
colormap gray
axis image
subplot(1,3,2)
imagesc(Y1)
title('\sigma = 10.2 cm')
xlabel('X-Direction (cm)')
ylabel('Y-Direction (cm)')
colormap gray
axis image
subplot(1,3,3)
imagesc(Y2)
title('\sigma = 100 cm')
xlabel('X-Direction (cm)')
ylabel('Y-Direction (cm)')
colormap gray
axis image

%find the elements inside the boundary that correspond to off-rate
error
y0 = find(0<Y0 & Y0<0.9);
y1 = find(0<Y1 & Y1<0.9);
y2 = find(0<Y2 & Y2<0.9);
```

```
%plot the applications maps with GNSS error outside the boundary
figure(5)
Z0 = V0.*(1-S);
Z1 = V1.*(1-S);
Z2 = V2.*(1-S);
subplot(1,3,1)
imagesc(Z0)
title('Application Outside Boundary')
colormap gray
axis image
subplot(1,3,2)
imagesc(Z1)
title('Application Outside Boundary')
colormap gray
axis image
subplot(1,3,3)
imagesc(Z2)
title('Application Outside Boundary')
colormap gray
axis image

%find the elements outside the boundary that correspond to off-rate
error
z0 = find(Z0>0.1);
z1 = find(Z1>0.1);
z2 = find(Z2>0.1);


%display the off-rate error inside the boundary as a percentage of
field area
Average_Error1 = length(y0)/Area
Average_Error2 = length(y1)/Area
Average_Error3 = length(y2)/Area
```

9.2 GNSS Model Application Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Title: GNSSmodel.m                                                  %
%   Author: Michael P. Sama (c) 2012                                    %
%   Date: 3/15/12                                                       %
%   Function: This script reads in an application map and               %
%             estimates the off-rate error due to boundary              %
%             effects in the field.                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear; %clear all variables
clc; %clear the command window

%read in the application map, convert to a binary image, and find the
%perimeter and area
I = imread('C:\Users\michael.sama.BAE-UK\Dropbox\PhD
Project\Dissertation\Field Boundaries\Field8.png');
BW = im2bw(I, graythresh(I));
BW2 = bwperim(BW,4);
```

```
[B,L] = bwboundaries(BW,'noholes');
[N,M] = size(BW);
S = regionprops(BW,'area','perimeter');
a = S.Area;
p = S.Perimeter;
pa = p/a;

%estimate off-rate errors based on three levels of GNSS accuracy
good = (0.0297+0.032)/2*pa;
average = (0.1176+0.1221)/2*pa;
poor = (0.6737+0.6512)/2*pa;

%plot the results
figure(1)
imshow(BW)
hold on
boundary = B{1};
plot(boundary(:,2),boundary(:,1),'r', 'LineWidth', 3);
title(['P/A = ',num2str(pa),', Good = ',num2str(good*100),'%, Average =
',num2str(average*100),'%, Poor = ',num2str(poor*100),'%'])
hold off
```

Appendix 10: Abbreviated Terminology

CORS – Continuously Operating Reference Station

CSV – Comma-Separated Values

DGPS – Differential Global Position System

DOP – Dilution Of Precision

GLONASS – Globalnaya Navigatsionnaya Sputnikovaya Sistema (Russian Global Navigation Satellite System)

GNSS – Global Navigation Satellite based System

GPS – Global Positioning System

ION – The Institute of Navigation

ISO – International Organization for Standardization

KTC – Kentucky Transportation Cabinet

OPUS – Online Position User Service

P/A – Perimeter to Area

PPS – Pulse Per Second

RTK – Real Time Kinematic

TTFF – Time To First Fix

TTS – Tracking Total Station

VFD – Variable Frequency Drive

WAAS – Wide Area Augmentation System

WGS – World Geodetic System

REFERENCES

Al-Gaadi, K. A., and P. D. Ayers. 1999. Integrating GIS and GPS into a spatially variable rate herbicide application system. *Applied Engineering in Agriculture* 15(4):255-262.

Behn, M., V. Hohreiter, and A. Muschinski. 2008. A scalable datalogging system with serial interfaces and integrated GPS time stamping. *Journal of Atmospheric and Oceanic Technology* 25(9):1568-1578.

Berns, H. G., and R. J. Wilkes. 2000. GPS time synchronization system for K2K. *Ieee Transactions on Nuclear Science* 47(2):340-343.

Boniger, U., and J. Tronicke. 2010. On the Potential of Kinematic GPR Surveying Using a Self-Tracking Total Station: Evaluating System Crosstalk and Latency. *Ieee Transactions on Geoscience and Remote Sensing* 48(10):3792-3798.

Borgelt, S. C., J. D. Harrison, K. A. Sudduth, and S. J. Birrell. 1996. Evaluation of GPS for Applications in Precision Agriculture. 12(6):633-638.

Chan, W. S., Y. L. Xu, X. L. Ding, Y. L. Xiong, and W. J. Dai. 2006. Assessment of dynamic measurement accuracy of GPS in three directions. *Journal of Surveying Engineering-Asce* 132(3):108-117.

Cole, J. T., T. S. Stombaugh, and S. A. Shearer. 2004. Development of a Test Track for the Evaluation of GPS Receiver Dynamic Performance. ASABE.

Daly, P., I. D. Kitching, D. W. Allan, and T. K. Peppler. 1991. Frequency and Time Stability of Gps and Glonass Clocks. *International Journal of Satellite Communications* 9(1):11-22.

Depenthal, C. 2008. A Time-referenced 4D Calibration System for Kinematic Optical Measuring Systems. In *1st International Conference on Machine Control & Guidance*.

Easterly, D. R., V. I. Adamchuk, M. F. Kocher, and R. M. Hoy. 2010. Using a vision sensor system for performance testing of satellite-based tractor auto-guidance. *Computers and Electronics in Agriculture* 72(2):107-118.

Ehsani, M. R., M. D. Sullican, T. L. Zimmerman, and T. S. Stombaugh. 2003. Evaluating the Dynamic Accuracy of Low-Cost GPS Receivers. ASABE.

Gavric, M., M. Martinov, S. Bojic, D. Djatkov, and M. Pavlovic. 2011. Short- and long-term dynamic accuracies determination of satellite-based positioning devices using a specially designed testing facility. *Computers and Electronics in Agriculture* 76(2):297-305.

Han, S., Q. Zhang, and H. Noh. 2002. Kalman filtering of DGPS positions for a parallel tracking application. *Transactions of the Asae* 45(3):553-559.

Han, S., Q. Zhang, H. Noh, and B. Shin. 2004. A dynamic performance evaluation method for DGPS receivers under linear parallel-tracking applications. *Transactions of the Asae* 47(1):321-329.

Harbuck, T. L., J. P. Fulton, T. P. McDonald, and C. J. Brodbeck. 2006. Evaluation of GPS Autoguidance Systems over Varying Time Periods. ASABE.

Hwang, S. Y., D. H. Yu, and K. J. Li. 2004. Embedded system design for network time synchronization. *Embedded and Ubiquitous Computing, Proceedings* 3207:96-106.

ION. 1997. ION STD 101 Recommended Test Procedures For GPS Receivers Revision C. The Institute of Navigation.

ISO. 2010. Tractors and machinery for agriculture and forestry -- Test procedures for positioning and guidance systems in agriculture -- Part 1: Dynamic testing of satellite-based positioning devices. International Orginization for Standardization.

Krischner, H., and W. Stempfhuber. 2008. The Kinematic Potential of Modern Tracking Total Stations. In *1st International Conference on Machine Control & Guidance*.

Luck, J. D., S. K. Pitla, R. S. Zandonadi, M. P. Sama, and S. A. Shearer. 2011. Estimating off-rate pesticide application errors resulting from agricultural sprayer turning movements. *Precision Agriculture* 12(4):534-545.

Luck, J. D., R. S. Zandonadi, B. D. Luck, and S. A. Shearer. 2010. Reducing Pesticide over-Application with Map-Based Automatic Boom Section Control on Agricultural Sprayers. *Transactions of the ASABE* 53(3):685-690.

MathWorks. 2012. *MATLAB.* Ver. 7.14.0.739 (R2012a). Natick, Massachusetts: The MathWorks Inc.

NMEA. 2000. NMEA 0183 Interface Standard. National Marine Electronics Association.

Perez-Ruiz, M., J. Carballido, J. Aguera, and J. A. Gil. 2010. Assessing GNSS correction signals for assisted guidance systems in agricultural vehicles. *Precision Agriculture* 12(5):639-652.

Robertson, M., P. Carberry, and L. Brennan. 2009. Economic benefits of variable rate technology: case studies from Australian grain farms. *Crop & Pasture Science* 60(9):799-807.

Sama, M. P., T. S. Stombaugh, R. S. Zandonadi, and S. A. Shearer. 2009. Dynamic GNSS Testing and Applications.

Saunders, S. P., G. Larscheid, B. S. Blackmore, and J. V. Stafford. 1996. A Method for Direct Comparison of Differential Global Positioning Systems Suitable for Precision Farming. *Precision Agriculture* acsesspublicati(precisionagricu3):663-674.

Silva, C. B., S. M. L. R. do Vale, F. A. C. Pinto, C. A. S. Muller, and A. D. Moura. 2007. The economic feasibility of precision agriculture in Mato Grosso do Sul State, Brazil: a case study. *Precision Agriculture* 8(6):255-265.

Smith, L. A., and S. J. Thomson. 2005. Gps position latency determination and ground speed calibration for the SATLOC Airstar M3. *Applied Engineering in Agriculture* 21(5):769-776.

Stombaugh, T. S., M. P. Sama, R. S. Zandonadi, S. A. Shearer, and B. K. Koostra. 2008. Standardized Evaluation of Dynamic GPS Performance.

Stombaugh, T. S., S. A. Shearer, J. P. Fulton, and M. R. Ehsani. 2002. Elements of a Dynamic GPS Test Standard. ASABE.

Taylor, R. K., M. D. Schrock, J. Bloomfield, G. Bora, G. Brockmeier, W. Burton, B. Carlson, J. Gattis, R. Groening, J. Kopriva, N. Oleen, J. Ney, C. Simmelink, and J. Vondracek. 2004. Dynamic testing of gps receivers. *Transactions of the Asae* 47(4):1017-1025.

Thomson, S. J., L. A. Smith, and J. E. Hanks. 2007. An instrumentation platform and GPS position latency issues for remote sensing on agricultural aircraft. *Transactions of the ASABE* 50(1):13-22.

Thrikawala, S., A. Weersink, G. Kachanoski, and G. Fox. 1998. Economic feasibility of variable rate technology for nitrogen on corn. *American Journal of Agricultural Economics* 80(5):1176-1176.

Watkins, K. B., Y. C. Lu, and W. Y. Huang. 1998. Economic and environmental feasibility of variable rate nitrogen fertilizer application with carry-over effects. *Journal of Agricultural and Resource Economics* 23(2):401-426.

Wu, C., P. D. Ayers, and A. B. Anderson. 2006. Influence of travel direction on GPS accuracy for vehicle tracking. 49(3):623-634.

Zandonadi, R. S., J. D. Luck, T. S. Stombaugh, M. P. Sama, and S. A. Shearer. 2011. A Computational Tool for Estimating Off-Target Application Areas in Agricultural Fields. *Transactions of the ASABE* 54(1):41-49.

VITA

MICHAEL P. SAMA, P.E.

EDUCATION

Ph.D. 2013: University of Kentucky – Biosystems and Agricultural Engineering
M.S. 2008: University of Kentucky – Biosystems and Agricultural Engineering
B.S. 2004: Rensselaer Polytechnic Institute – Computer and Systems Engineering

LISCENSURE

Professional Engineer, Electrical and Computer Engineering, Kentucky, 2011– Current

PROFESSIONAL EXPERTISE

Machine Systems Automation Engineering, Software Development, Instrumentation, Data Acquisition, Serial Communication, Embedded Control Systems, Computer-Aided Design

TEACHING

Instructor, BAE 658: Instrumentation for Engineering Research, 2009 – Current
Assistant, BAE 658: Instrumentation for Engineering Research, 2005 – 2006
Lab Instructor, BAE 305: DC Circuits and Microelectronics, 2005 – 2012

ADVISING

Advisor, UK Quarter Scale Tractor Team, 2010 – Current

PROFESSIONAL MEMBERSHIP

American Society of Agricultural and Biological Engineers
Alpha Epsilon: The Honor Society of Agricultural, Food, and Biological Engineering
Gamma Sigma Delta: The Honor Society of Agriculture

AWARDS

Outstanding Doctoral Student, Gamma Sigma Delta, 2012
New Faces of Engineering, National Engineers Week, 2012
New Faces of ASABE, American Society of Agricultural and Biological Engineers, 2012
Sunkist Young Designer Award, American Society of Agricultural and Biological Engineers, 2011
Outstanding Masters Student, Gamma Sigma Delta, 2006

SERVICE

FFA Planning Committee, ASABE, 2012 – Current
PM-54 (Precision Agriculture) Member, ASABE, 2011 – Current
Computer Committee, Biosystems and Agricultural Engineering, 2005 – 2011
Cooking Crew, Ag Roundup, 2005 – Current
Team Member, UK Quarter Scale Tractor Team, 2005 – 2010
Presenter, Engineering Day, 2005 – 2013

RESEARCH & PROJECTS

Mobile Device Applications for Agricultural Machine Monitoring, 2012 – Current
Wind Tunnel Pressure Control, 2011– 2012
Variable Flow Rate Nozzle Controller, 2011 – 2012
Evaluating GPS Autoguidance for Specialty Crop Management, 2011
Scalable Control and Data Acquisition for Variable-Rate Applications, 2010 – Current
Instrumentation of a Grain Compaction Device, 2010 – 2011
High Pressure Liquid Pesticide Metering and Injection System, 2009 – 2012
Water Level Sensor for Shrimp Tanks, 2008
Gas Measurement Control System for Multiple Biofilters, 2008
Fan Assessment Numeration System (FANS), 2007 – Current
Rainfall Simulator Control System, 2006
Controlling Feedlot Runoff in a Basin, 2005 – 2008
Standardized Testing of Satellite-Based Navigation Systems, 2005 – Current
Analyzing Sprayer Droplet Distribution using MATLAB, 2005 – 2007
Low-Cost Remote Sensing in Agriculture, 2004 – 2008
Camera & Controls for Big Blue 3, 2004 – 2005

FUNDING - $428,778

Pitla, S.K., **M.P. Sama**, J.D. Luck. 2012. *AgStatMonitor: A Mobile Device Application for Agricultural Machine Monitoring*. ASABE Mobile App Challenge. **$9,397**.

Sama, **M.P.**, G.B. Day. 2011. *Fan Assessment Numeration Systems (FANS) for Agricultural Building Ventilation Measurement in Emissions Testing.* Individual Contracts with UIUC (1), IASU (1), UGPH (1), UDEL (1). **$39,000**.

Sama, M.P., T.S. Stombaugh, J.D. Luck. 2010. *Scalable Control and Data Acquisition for Variable-Rate Applications.* USDA-NIFA. **$48,710**.

Luck, J.D., S.K. Pitla, **M.P. Sama**, S.A. Shearer. 2010. *Sprayer Controller Evaluation for Improving Spatial Application of Pesticides*. USDA-NIFA. **$49.976**.

Zandonadi, R.S., T.S. Stombaugh, **M.P. Sama**. 2009. *Reduced Equipment Set for Multiple Vehicle Guidance*. USDA-CSREES. **$49,992**.

Luck, J.D., S.K. Pitla, **M.P. Sama**, S.A. Shearer. 2009. *A Pneumatic Nozzle Control System for Variable-Rate Pesticide Application*. USDA-CSREES. **$49,432**.

Sama, M.P., T.S.Stombaugh, S.A.Shearer. 2008. *A System for Implementing Dynamic Accuracy Standards for Machine Guidance Technology in Agriculture*. USDA-CSREES. **$62,271**.

Gates, R.S , **M.P. Sama**, G.B. Day. 2007. *Fan Assessment Numeration Systems (FANS) for Agricultural Building Ventilation Measurement in Emissions Testing*. Contract with Purdue University. **$60,000**.

Gates, R.S., **M.P. Sama**, G.B. Day. 2007. *Fan Assessment Numeration Systems (FANS) for Agricultural Building Ventilation Measurement in Emissions Testing*. Individual Contracts with ISU(2), SDSU(1), UMN(1), UAR(1), USDA-ARS-MS(1). **$60,000**.

REFEREED PUBLICATIONS

Sama, M.P., T.S. Stombaugh, J.E. Lumpp. *A Hardware Method for Time-Stamping Asynchronous Serial Data Streams Relative to GNSS Time*. (In Review)

Maupin, T.P., C.T. Argouridis, D.R. Edwards, C.D. Barton, R.C. Warner, **M.P. Sama**. *Specific Conductivity Sensor Performance: II. Field Evaluation*. International Journal of Mining, Reclamation and Environment.

Zandonadi, R.S., T.S. Stombaugh, J.D. Luck, **M.P. Sama**, S.A. Shearer. 2011. *A Computational Tool for Estimating Off-Target Application Areas in Agricultural Fields*. Transactions of ASABE. PM-08498-2010.

Luck, J.D., S.K. Pitla, R.S. Zandonadi, **M.P. Sama**, S.A. Shearer. 2011. *Estimating Off-Rate Pesticide Application Errors Resulting from Agricultural Sprayer Turning Movements*. Precision Agriculture. 12(4): 534-545.

Zandonadi, R.S., T.S. Stombaugh, S.A. Shearer, D.M. Queiroz, **M.P. Sama**. 2010. *Laboratory Performance of a Mass Flow Sensor for Dry Edible Bean Harvesters*. Applied Engineering in Agriculture. Vol. 26(1): 11-20.

INVITED SPEAKER PRESENTATIONS

Sama, M.P., T.S. Stombaugh. 2007. *Low Cost Remote Sensing Platform*. IV Simpósio Internacional de Agricultura de Precisão, 23 a 25 de outubro de 2007, Viçosa-MG. (Presented by T.S. Stombaugh).

CONFERENCE PRESENTATIONS

Agouridis, C., T. Maupin, C. Barton, D. Edwards, R. Warner, **M.P. Sama**. 2012. *Assessing Conductivity Sensor Performance: A Laboratory and Field Study.* 2012 Southeast Regional Stream Restoration Conference.

Sama, M.P., L.M. Pepple, G.B. Day, D.G. Overhults, G.M. Morello, I.M. Lopes, J. Earnest, K.D. Casey, R.S. Gates. 2012. *Calibration Drift Assessment and Upgrades to the Fan Assessment Numeration System (FANS)*. Paper Number 121337770, 2012 ASABE Annual Meeting.

Sama, M.P., G.M. Morello, I.M. Lopes, G.B. Day, D.G. Overhults. 2012. *Visualizing Airflow Using the Fan Assessment Numeration System (FANS)*. Paper Number 121337883, 2012 ASABE Annual Meeting.

Luck, J.D., **M.P. Sama**, S.K. Pitla, S.A. Shearer. 2012. *Droplet Spectra Characteristics from a Variable-Orifice Nozzle at Constant Pressures*. Paper Number 121337472, 2012 ASABE Annual Meeting.

Lopes, I.M., F.A. Damasceno, G.B. Day, **M.P. Sama**, D.G. Overhults. 2012. *WINTAC: A Wind Tunnel Transition Assessment Chamber at the Biosystems and Agricultural Engineering Department at University of Kentucky*. Paper Number 121337360, 2012 ASABE Annual Meeting.

Luck, J.D., **M.P. Sama**, S.A. Shearer. 2012. *Spray Pattern and Droplet Spectra Characteristics from an Actively Controlled Variable-Orifice Nozzle*. 2012 International Conference on Precision Agriculture.

Black, R.A., T.S. Stombaugh, S.R. Luciani, **M.P. Sama**, R.L. Klingefus, A.B. Klingefus, J.M. Bewley. 2012. *Potential for a Real-Time Location System for Dynamic Tracking of Dairy Cow Location within Dairy Facilities*. 2012 American Dairy Science Annual Meeting.

Sama, M.P., R.S. Zandonadi, J.D. Luck, T.S. Stombaugh, S.A. Shearer. 2011. *A Static Evaluation of Continuously Operating Reference Stations.* 2011 ASABE Annual Meeting.

Montross, M.D., W.C. Adams, L. Mathis, S. McNeill, **M.P. Sama**, S. Thompson, J. Boac, M. Casada. 2011. *Laboratory Data with Hard Red Winter Wheat to Support New Grain Packing Factors*. 2011 ASABE Annual Meeting.

Zandonadi, R.S., T.S. Stombaugh, **M.P. Sama**, S.K. Pitla, R. Baldo. 2011. *Evaluation of a Reduced Equipment Set for Multiple Vehicle Guidance Using Distance Sensors to Determine Relative Position between Vehicles*. 2011 ASABE Annual Meeting.

Sama, M.P., R.S. Zandonadi, J.D. Luck, T.S. Stombaugh, S.A. Shearer. 2010. *Development of a Scalable Control System for Variable-Rate Applications*. 2010 ASABE Annual Meeting.

Luck, J.D., M.P. Sama, S.K. Pitla, S.A. Shearer. 2010. *Pneumatic Control of a Variable Orifice Nozzle*. 2010 ASABE Annual Meeting.

Sama, M.P., T.S. Stombaugh, R.S. Zandonadi, S.A.Shearer. 2009. *Dynamic GNSS Testing and Applications*. Paper Number 096714, 2009 ASABE Annual Meeting.

Sama, M.P., T.S. Stombaugh, R.S. Zandonadi, S.A. Shearer, W.C. Adams, 2009. *A Mechanism for Evaluating Dynamic GNSS Accuracy on a Test Fixture*. 2009 ASABE AETC.

Sama, M.P., R.S. Gates, W.C.Adams, G.B.Day, C.L.King, 2008. *Fan Assessment Numeration System (FANS) Scaling and Upgrades*. Paper Number 084723, 2008 ASABE Annual Meeting.

Sombaugh, T.S., **M.P. Sama**, R.S. Zandonadi, S.A. Shearer, 2008. *Standardized Evaluation of Dynamic GPS Performance*. Paper Number 084728, 2008 ASABE Annual Meeting.

Zandonadi, R.S., T.S.Stombaugh, S.A.Shearer, **M.P. Sama**, 2008. *Laboratory Performance of a Low Cost Mass Flow Sensor for Combines*. Paper Number 084167, 2008 ASABE Annual Meeting.

Dougherty C.T., E.S. Flynn, R.J. Coleman, **M.P. Sama**, T.S. Stombaugh, 2006. *Remote Sensing of Equine Bermudagrass Pastures from a Helikite™.*

Sama M.P., T.S. Stombaugh, B.K. Koostra, 2006. *Calibration and Verification of Low-Cost Image Tools for Remote Sensing*. Paper Number 061166, 2006 ASABE Annual Meeting.

Brown D., T. Arrowsmith, A. Mylin, R. Koontz, A. Fox, N. Phelps, D. Thomas, J. Rowe, R. Jones, D. Jackson, A. Groves, **M.P. Sama**. 2005. *AIRCAT: Airborn Intelligent Research Craft for Autonomous Technology*. 2005 AUVSI Unmanned Systems Meeting.

Sama M.P., D.E. Hershman, T.S. Stombaugh, S.G. McNeill, P. Needham. 2005. *An Analysis of Sprayer Droplets Using Matlab*. 2005 National Soybean Rust Symposium.

Sama M.P., T.S. Stombaugh. 2005. *Adaptation and Modification of Digital Imaging Systems for Remote Sensing*. Paper Number 051016, 2005 ASAE Annual Meeting.