



University of Kentucky
UKnowledge

Theses and Dissertations--Biosystems and
Agricultural Engineering

Biosystems and Agricultural Engineering


2019

REMOVING VEHICLE SPEED FROM APPARENT WIND VELOCITY

Austin M. Weiss

University of Kentucky, austin.weiss@uky.edu

Author ORCID Identifier:

 <https://orcid.org/0000-0002-6767-0039>

Digital Object Identifier: <https://doi.org/10.13023/etd.2019.323>

[Right click to open a feedback form in a new tab to let us know how this document benefits you.](#)

Recommended Citation

Weiss, Austin M., "REMOVING VEHICLE SPEED FROM APPARENT WIND VELOCITY" (2019). *Theses and Dissertations--Biosystems and Agricultural Engineering*. 67.
https://uknowledge.uky.edu/bae_etds/67

This Master's Thesis 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 needed written permission statement(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) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free 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 an 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 thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Austin M. Weiss, Student

Dr. Michael P. Sama, Major Professor

Dr. Donald G. Colliver, Director of Graduate Studies

REMOVING VEHICLE SPEED FROM APPARENT WIND VELOCITY

THESIS

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Biosystems and Agricultural Engineering in the Colleges of Agriculture and Engineering at the University of Kentucky

By

Austin Meade Weiss

Lexington, Kentucky

Director: Dr. Michael Sama, Associate Professor of Biosystems and Agricultural Engineering

Lexington, Kentucky

2019

Copyright © Austin Meade Weiss 2019
<https://orcid.org/0000-0002-6767-0039>

ABSTRACT OF THESIS

REMOVING VEHICLE SPEED FROM APPARENT WIND VELOCITY

Variable-rate technologies for sprayer applications stand to increase efficacy by ensuring the right amount of chemical is applied at the right location. However, external environmental factors such as droplet drift caused by variable ambient condition, are not yet integrated into modern sprayer systems. Real-time wind velocity measurements can be used to control droplet spectra for reducing spray drift by actuating a variable-orifice nozzle. This work aimed to develop data processing methods needed to filter noise and remove vehicle speed from wind velocity measurements collected with an ultrasonic anemometer aboard a moving platform. Using a global navigation satellite system (GNSS), vehicle speed was calculated in the field and subtracted from apparent wind velocity for comparison to static measurements. Experiments under stationary and dynamic sensor deployments were used to develop an algorithm to provide instantaneous local wind velocity and to better understand the local spatiotemporal variability of wind under field conditions.

KEYWORDS: Wind measurement, Ultrasonic Anemometer, Correlation, Sprayer drift, Variable-rate technology

Austin Meade Weiss

(Name of Student)

06/06/2019

Date

REMOVING VEHICLE SPEED FROM APPARENT WIND VELOCITY

By
Austin Meade Weiss

Michael P. Sama

Director of Thesis

Donald G. Colliver

Director of Graduate Studies

06/06/2019

Date

ACKNOWLEDGMENTS

This work was the result of much support at the University of Kentucky. First, I am deeply grateful to my advisor Dr. Michael Sama for providing me guidance and mentorship throughout my time at UK. Aside from his many technical contributions throughout the project, his flexibility, support, and optimism made my graduate experience enjoyable. Also, thank you to my committee members Dr. Michael Montross and Dr. Ole Wendroth for their support throughout my graduate program. I'd also like to give credit to Matt Miller for his work designing the printed circuit boards, Shawn O'Neal for his utility vehicle preparations, and Amanda Williams for assisting with field trials. Additionally, thanks to Luis Felipe Pampolini for assistance on several tasks and for his friendship. Thanks to the USDA for funding this research as a collaboration with the University of Nebraska-Lincoln.

I thank the University of Kentucky's Department of Biosystems & Agricultural Engineering for setting me on my career path. I have grown tremendously in my time at BAE, and I greatly appreciate the many dedicated faculty that I've learned from in my college years. Finally, I'd like to express gratitude to my mom and brother, who continue to support me unconditionally in all of my endeavors.

Table of Contents

ACKNOWLEDGMENTS	iii
List of Tables	viii
List of Figures.....	ix
CHAPTER 1. PROJECT MOTIVATION AND LITERATURE REVIEW	1
1.1 Introduction:.....	1
1.2 Objectives:.....	3
1.3 Causes and Effects of Spray Drift on Efficacy of Liquid Application.....	3
1.4 The Variable Orifice Nozzle	4
1.5 Anemometers for Quantifying Wind Velocity.....	5
1.5.1 Uncertainties Surrounding Ultrasonic Anemometers for Agricultural Applications	6
1.5.2 Timestamping Wind Velocity Measurements	9
1.6 Statistical Analysis for Similitude of Wind Data.....	11
1.7 Previous Spray Drift Characterization Methods and Modeling.....	12
CHAPTER 2. CORRELATION TESTING BETWEEN TWO STATIONARY ULTRASONIC ANEMOMETERS.....	14
2.1 Methods and Materials:.....	14
2.1.1 Ultrasonic Anemometer Interfacing:	15
2.1.2 Hardware Timestamping of Serial Data from Ultrasonic Anemometers.....	20

2.1.3	Test Fixture Design and Assembly	22
2.1.4	Preliminary Data Collection and Setup Procedure	24
2.1.5	Field Testing	27
2.1.6	Processing Procedures	31
2.1.7	Assumptions and Validity of Correlation Analysis	32
2.1.8	Quantifying Uncertainty in Processed Wind Velocities	34
2.2	Results:	36
2.2.1	Visualizing Data and Filter Uncertainties	36
2.2.2	Correlation Results:	38
2.2.3	Distribution of measurement differences between the two anemometers: ..	43
2.2.4	A Note on Filtering Uncertainty Analysis by Probability Distribution	47
2.2.5	Conclusions from Static Testing	48
CHAPTER 3. CORRELATION TESTING BETWEEN ONE DYNAMIC AND ONE STATIONARY ANEMOMETER		50
3.1	Methods and Materials:	50
3.1.1	Equipment Setup and Vehicle Mounting	53
3.1.2	Methods for Calculation of Weather Station Velocity	53
3.1.3	Method 1: Successive Coordinates	55
3.1.4	Method 2: Relative Velocity to GNSS	62

3.1.5	Calculating Velocity at Weather Station’s Location relative to GNSS Location	63
3.1.6	Method 3: Calculating Haversine velocity at GNSS location and using the relative velocity method to transform velocity to the weather station’s location	68
3.1.7	Adjustments and Limitations for Methods for A Circular Path.....	68
3.1.8	Comparing Methods of Anemometer Velocity.....	70
3.1.9	Removing Vehicle Velocity from Apparent Wind Data.....	73
3.1.10	Quantifying Similitude Between Anemometer and Dynamic Weather Station	78
3.2	Results:	79
3.2.1	Vehicle Velocity and Course Comparisons	79
3.2.2	Visualizing Filter Uncertainty.....	85
3.2.3	Validating Wind Gusts by Comparing Direction.....	87
3.2.4	Cross-correlation Results	89
3.2.5	Special Cases in Cross-Correlation Analysis.....	94
3.2.6	Distribution of Measurement Differences	97
3.3	Discussion:	100
3.3.1	Weather Station Velocity Transformation	100
3.3.2	Feasibility at Varied Distances	103
CHAPTER 4. CONCLUSIONS AND FUTURE WORK.....		104

APPENDICES	105
Appendix 1. Schematics for Machined and 3D Printed Parts.....	105
Appendix 2. MATLAB Code for Stationary Testing	109
Appendix 3. MATLAB Code for Dynamic Testing	119
Appendix 4. R Script.....	158
Appendix 5. Data Logging Code (VB.NET)	161
Appendix 6. Microcontroller Design and Code:.....	170
References:.....	186
VITA.....	188

List of Tables

Table 2.1 Output data from sensors can be separated in MATLAB by their sensor addresses.	19
Table 2.2: Completely randomized block design for stationary testing	28
Table 2.3: Completely randomized block design for stationary testing	30
Table 2.4: Unfiltered trials generally all had structured decay in cross-correlograms except for one trial.	41
Table 3.1: The completely randomized block design is shown. The procedure moved down each block in order, completing separations from top to bottom, and vehicle velocities from left to right. Two of the 14.48 km/h trials were omitted because of difficulty turning the vehicle at that speed.....	51
Table 3.2: A summary of cross-correlogram structure for unfiltered dynamic trials. “YES” indicates a symmetrical decay in cross-correlation coefficient, “NO” indicates no structured decay. Descriptive notes are included for special cases.	92
Table 3.3: Ranking for each trial's cumulative probability data is shown. A rank equal to 1 indicates closer wind velocities between sensors, as a higher percent of velocity pairs have smaller differences.	98

List of Figures

Figure 1.1 The components of the variable orifice nozzle consist of a metering stem moved by a linear actuator (D. Luck, A. Shearer, P. Sama, & K. Pitla, 2015).....	5
Figure 2.1 Ultrasonic anemometers used in this study were manufactured by RM Young (Young).....	15
Figure 2.2 The Serial Communication Interface allows control of the data logging procedure and provides graphical display of ultrasonic anemometer sensor parameters .	17
Figure 2.3: Data sentence format for the 92000 ResponseOne (Young) and 86000 (Young) ultrasonic anemometers from the user manuals. Each sentence is separated by a carriage return <CR>. This is used to separate sentences from each other. The class definition of carriage return in Visual Basic is "vbCr"	18
Figure 2.4 The assembled PCB allows a hardware method of timestamping	21
Figure 2.5: Assembly for Preliminary Test Fixture	23
Figure 2.6 The assembled PCB and case	24
Figure 2.7 A satellite image displays the location of the test site at 38.026924° , - 84.509623° (Google, n.d)	25
Figure 2.8 A cart was used as a surface for the computer, power supply, PCBs, and GPS receivers	26
Figure 2.9: Static testing was completed at the University of Kentucky’s C. Oran Little Research center in Versailles, KY on October 5 th , 2018.	28
Figure 2.10: Static testing at the University of Kentucky’s North Farm was completed on February 13 th , 2019.....	30

Figure 2.11: Wind velocity data is shown for the first 30 seconds of data at 20 ft separation, block 1, filtered using a 3-point moving average	37
Figure 2.12: Wind direction is shown for the first 30 seconds of data at 20 ft separation, block 1, filtered using a 3-point moving average	38
Figure 2.13: Autocorrelograms for 20ft separation, trial 1, show structured autocorrelation warranting a cross-correlation analysis.	39
Figure 2.14: Cross-Correlogram for 20ft separation, trial 1, shows structured decay in correlation with increasing lag.....	40
Figure 2.15: Cross-Correlogram for all separation distances in block 1 shows decreasing correlation with increasing distance and flatter structure, suggesting less similarity in wind velocity with increasing distance on the scale of spray boom size.....	42
Figure 2.16: Cross-correlogram for all separation distances in block 2 shows decreasing correlation with increasing distance and flatter structure, suggesting less similarity in wind velocity with increasing distance	42
Figure 2.17: Cross-correlogram for 60 ft separation, trial 3, shows similar trends as Figure 2.17 and Figure 2.18, but displays the highly varied 40 ft trial.....	43
Figure 2.18: Cumulative probability chart for 20ft separation distance, trial 1.....	44
Figure 2.19: Cumulative probability plot for all trials in block 1	45
Figure 2.20: Cumulative probability plot for all trials in block 2.....	46
Figure 2.21: Cumulative probability plot for all trials in block 3.....	46
Figure 2.22: Sample histogram of filtering uncertainties distribution for block 1, 20 ft separation	48
Figure 3.1: Aerial photo visualizing the three radii for dynamic testing.....	52

Figure 3.2: Diagram shows the definition of azimuth forward bearing θ , and bearing facing the weather station $\theta_G \rightarrow w$	56
Figure 3.3: Legend for equations 3.1-3.10.....	58
Figure 3.4: The Haversine calculated form of method 1 yielded poor similitude when using equations 4-6 for new coordinate calculation	59
Figure 3.5: Weather stations coordinates solved using Vincenty's formula were effective at transforming at a set distance.....	61
Figure 3.6 Distances between GNSS coordinates and transformed coordinates (coordinate pairs) show better accuracy using the WGS-84 Earth model	62
Figure 3.7: Diagram displaying turn radii and velocity vectors at the GNSS and weather station locations. The direction of the velocity vectors are pointing to the rear of the vehicle for visibility.....	65
Figure 3.8: Turn radius visualization at the GNSS location requires two velocity measurements. Velocity vectors are pointing in the opposite direction for improved visibility.....	67
Figure 3.9: Legend for parameters in equations 16-18.....	67
Figure 3.10: Flow chart visualizing Method 1 weather station velocity calculation.....	72
Figure 3.11: Flow chart visualizing Method 2 weather station velocity calculation for known turn radii.....	72
Figure 3.12: Flow chart visualizing method 3 weather station velocity calculation	72
Figure 3.13: A diagram visualizing wind components and resultant vectors after subtraction of vehicle velocity from the "Y" component. VWind Corrected is the resultant vector of Y' and X components.....	75

Figure 3.14: Visualization of wind direction calculation according to equations 21-24.. 76

Figure 3.15: Diagram displays principle of correction of wind directions to be relative to True North..... 78

Figure 3.16: Three methods of calculation for weather station velocity yield close results for dynamic testing at 20 ft anemometer separation, approximately 3 mph vehicle speed, trial 1. Method 1 (M1) results were noisier when unfiltered. 80

Figure 3.17: Three methods of calculation for weather station velocity yield close results for dynamic testing at 20 ft anemometer separation, approximately 6 mph vehicle speed, trial 1. 81

Figure 3.18: Three methods of calculation for weather station velocity yield close results for dynamic testing at 20 ft anemometer separation, approximately 9 mph vehicle speed, trial 1. 81

Figure 3.19: The data for block 3, at a 40 ft separation, 6 mph, no filter (top) and 7-point moving average (bottom) show filtering’s effect of smoothing weather station dynamics on method 1 for reducing noise before backing it out from wind data. 82

Figure 3.20: Course calculations for all three methods have good alignment with minimal noise. This sampled data is from Block 1, with an anemometer separation distance of 20 ft and 3 mph vehicle velocity target..... 83

Figure 3.21: Course calculations for all three methods from sampled data from Block 1, with an anemometer separation distance of 20 ft and 6 mph vehicle velocity target. 84

Figure 3.22: Course calculations for all three methods from sampled data from Block 1, with an anemometer separation distance of 20 ft and 9 mph vehicle velocity target. 85

Figure 3.23 Scatter plot with filter uncertainties defined as the standard deviation over the filtering window.....	86
Figure 3.24: Noise from calculating vehicle dynamics carry into wind calculations when unfiltered. This data is from block 1, 40ft separation, 3 mph speed.....	88
Figure 3.25: Noise in wind data were reduced using moving average filtering. This dataset was from block 1, 40 ft separation, 3 mph, with a 7-point moving average applied.	88
Figure 3.26: Increasing filtering window size improves correlation Block 1, 60 ft, 9 mph	89
Figure 3.27: Increasing filter size results in method 1 having the highest cross-correlation and similarity. Data from block 2, 20 ft separation distance, 3 mph is shown with no filtering (a. top) and 7-point moving average (b. bottom)	90
Figure 3.28: Block 1, 20 ft, 3 mph (a. top), 6 mph (b. bottom)	93
Figure 3.29: Block 1, 40ft, 9 mph, 7-point moving average.....	94
Figure 3.30: The cross-correlogram for the trial in Block 2, 40ft separation, 6 mph target vehicle velocity exhibits a flat shape even with a 7-point moving average applied, suggesting more random variability among one of the time-series datasets	95
Figure 3.31: Plotted data for Block 2, 40 ft separation, 6 mph targeted vehicle velocity still displays an abundance of noise after filtering with a 7-point moving average	96
Figure 3.32: Vehicle velocity calculations (Method 1) show minimal noise, eliminating it as a source for irregularities in processed wind data	96
Figure 3.33: Cumulative probability distribution shows increasing the moving average filtering window smooths velocity data and minimizes differences in values	99

Figure 3.34: Cumulative probability chart shows higher percent of small wind velocity differences at slower vehicle speeds 99

CHAPTER 1. PROJECT MOTIVATION AND LITERATURE REVIEW

1.1 Introduction:

Sprayer drift is the phenomenon of misapplication of liquids on non-targeted areas often caused by wind interference. Sprayer drift affects farm efficiency through the introduction of off-target and off-rate application errors (Grover, Maybank, Caldwell, & Wolf, 1997), has potential to negatively affect neighboring crops, and has potential to harm the environment (de Snoo & van der Poll, 1999). A common solution to the problem is to avoid application on windy days at all (Mekonnen & Agonafir, 2002), while some producers could be tempted to increase application of water and crop protection chemicals to compensate for losses displaced by wind. Much research has indicated that over-application or drift of crop controlling chemicals such as pesticides, herbicides, and fertilizers contribute to issues such as crop loss (Everitt & Keeling, 2009), herbicide resistance in weeds (Duke, 2005), and damages to the environment (Gove, Power, Buckley, & Ghazoul, 2007). Even non-occupational exposure has been shown to cause risks to human health (Azaroff & Neas, 1999). Increases in some herbicide-resistant weeds can be attributed to selection pressure against non-resistant weeds, and herbicides subjected to drift can reduce natural fauna health and biodiversity in non-targeted areas (Duke, 2005). In some woodland species fauna, drift from nearby application of fertilizer has been shown to cause reduced fertility, while herbicide reduced biomass and increased mortality rates (Gove et al., 2007). These chemicals can contaminate nearby water sources and pose health risks to human health (Holt, 2000). Under-application on targeted areas caused by drift can also fail to fulfill the desired

effects. Movement of a spray boom itself at different speeds also can cause variation in longitudinal droplet distribution that affect drift (Ooms, Ruter, Lebeau, & Destain, 2003).

To mitigate the effects of spray drift, a collaborative group was developed comprising researchers from the University of Nebraska-Lincoln and the University of Kentucky. The goal of the project was to develop a system in which a variable-orifice nozzle, capable of adjusting the droplet spectrum (e.g. distribution of droplet sizes) of applied liquids, could be actuated according to real-time wind velocity data from an on-board weather station. By increasing droplet size during wind gusts, coverage of applied products may be better assured as wind's influence of droplet's trajectories are reduced while droplets that are too large may fail to stick to their targeted areas. A calibrated system may ensure optimal coverage for weather conditions in real time. The goal for this work is to design an interface that integrates real-time wind velocity data from an on-board weather station for the control input of a variable-orifice nozzle. Additionally, the project aims to test and determine the feasibility of sensor accuracy onboard a moving platform, as well as determine appropriate sampling rates and filtering processes to produce highly correlated, accurate results between stationary and dynamic wind velocity measurements. The following specific objectives were developed in effort to achieve the project goal.

1.2 Objectives:

1. Interface two ultrasonic anemometers with global navigation satellite system (GNSS) receivers to timestamp and record wind velocity data.
2. Determine the appropriate sampling rate and filtering process to produce highly correlated wind velocity measurements between multiple sensors.
3. Remove vehicle speed from apparent wind velocity measurements.

1.3 Causes and Effects of Spray Drift on Efficacy of Liquid Application

Many studies have been conducted investigating the causes of droplet drift during liquid application using spray nozzles. Experiments show that nozzle height, droplet speed in the direction of gravity, flow rate, pressure, wind speed, and air temperature are significant variables contributing to droplet drift in a single sprayer system (B. Smith, E. Bode, & D. Gerard, 2000). Nozzle height and droplet speed provide potential for drift by affecting the time for droplets to reach the ground. Larger distances between a nozzle and the ground allow greater opportunity for interference with droplet trajectories by wind gusts. These observations on the causes of sprayer drift are valuable however – to mitigate nozzle drift for boom applications – a means of collecting and filtering ambient wind measurements near the ground must be developed and investigated if real-time decisions on droplet spectra are to be made.

Variability of field wind velocity is consistently a source of uncertainty for modeling spray drift (Butler Ellis et al., 2017). Wind tunnel testing has proven a viable method for simulating these field conditions, in which Butler used a wind tunnel to estimate spray drift in simulated environmental conditions for the purpose of minimizing

agricultural buffer zones (unused buffer space between crops and natural fauna for drift particles to land). The tunnel data was compared to field data for analysis of the process' similitude. They found that their method is successful for short distance drift in the field up to 20 meters for simulating field conditions. Testing at the UNL Spray Application Testing Lab and the UNL West Central Research and Extension Center Pesticide Application Technology Laboratory will investigate drift estimation. Meanwhile, this work aims to provide a mechanism for real-time quantification of wind conditions in the field.

1.4 The Variable Orifice Nozzle

Tests on a single electronically actuated variable-orifice nozzle found that pressure affects droplet size or spectra more than flow rate (D. Luck, K. Pitla, P. Sama, & A. Shearer, 2015). Higher pressures yielded finer liquid particles while lower pressures create larger droplets. A metering stem was used to actuate the variable-orifice, and analysis of data showed its position/nozzle flow rate had no significant trend on droplet diameter. Using pressure as a means to actuate this variable-orifice technology stands to reduce errors in application caused by pressure loss in fixed-orifice systems and will be the preferred control response for adjusting droplet spectra according to instantaneous wind velocity inputs. By adjusting to larger droplets during wind gusts, wind may have a reduced impact on droplet trajectories.

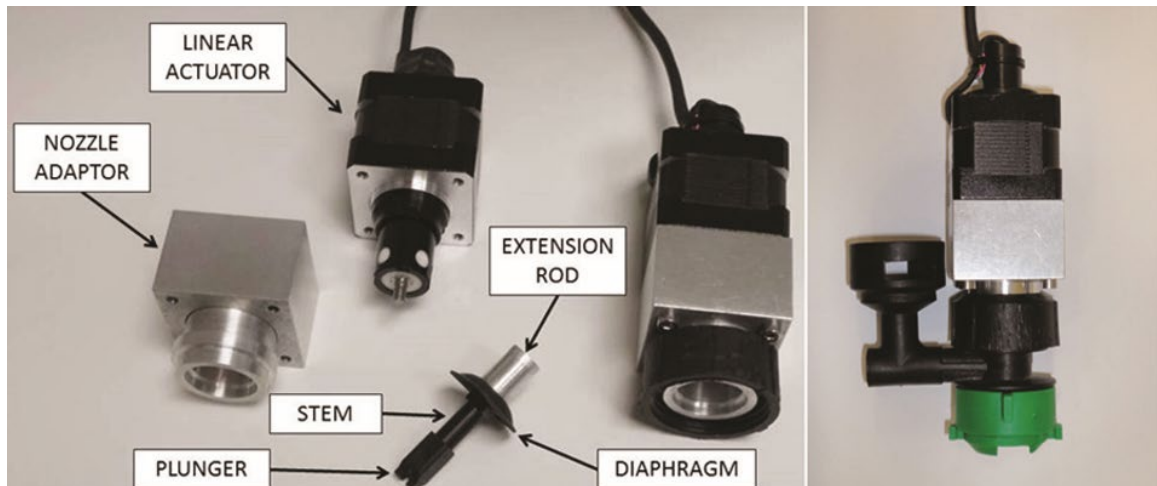


Figure 1.1 The components of the variable orifice nozzle consist of a metering stem moved by a linear actuator (D. Luck, A. Shearer, P. Sama, & K. Pitla, 2015)

1.5 Anemometers for Quantifying Wind Velocity

Cup anemometers are a relatively low-cost method for measuring wind speed. However, they have inherent disadvantages stemming from their design. By calculating wind speeds from the mechanical rotation of cups, many cup anemometers are limited by starting thresholds of about 0.5 m/s with resolutions also equal to 0.5 m/s. Additionally these sensors risk mechanical wear from long term continuous operation and by collision or jamming from dust/debris that can affect the accuracy of measurements if not properly maintained. They are also incapable of measuring direction without a separate wind vane.

Alternatively, ultrasonic anemometers use transducers to emit and detect sonic pulses at desired intervals. When a sonic pulse is emitted from the sending transducer, the time-of-flight to receiving ends are measured. As wind gusts on the sensor, the time-of-flight of sonic pulses is changed by the physical contact of air on the sonic wave. The device takes this difference of time-of-flight measurement from each receiving transducer

and calculates components of the wind's magnitude and direction in either two or three orthogonal dimensions depending on the sensor configuration. Ultrasonic anemometers allow high frequency collection of wind velocity as vector components. Measurements are more accurate compared to cup anemometers because the absence of moving parts eliminates uncertainty caused by cup momentum and mechanical wear. Also, they aren't limited by a minimum threshold as cup anemometers are. Ultrasonic anemometers are ideal for this project because of their higher accuracy and reliability.

1.5.1 *Uncertainties Surrounding Ultrasonic Anemometers for Agricultural Applications*

Typically for agricultural and meteorological applications, ultrasonic anemometers are mounted on stationary platforms as a component in a weather station. While these anemometers are used onboard vessels at sea, wind measurements seen by the sensor are used to provide information about high-volume air collision on the side of sea vessels (which contribute to yaw and rocking), or for estimating propulsion (for sailboats). Although the vessel's dynamics cause false wind velocities, this fact can be disregarded because accuracy of wind gusts (especially at low speeds) aren't used as control inputs. Corrections for vessel tilt-induced airflow distortion for better agreement with weather stations have been investigated in literature to some success (Landwehr, O'Sullivan, & Ward, 2015) however aim to reduce much larger errors than expected on land where large vehicle tilt is less likely.

Anemometers mounted aboard agricultural vehicles will produce a false apparent wind velocity as the anemometer is moved through the ambient flow field. The close proximity to the ground and the irregular shape of high-clearance self-propelled sprayers (the target application of this work) will likely add turbulence that is readily detected by

an ultrasonic anemometer. Obstacles surrounding the device can also interfere with the ambient flow field. Uncertainty in stationary ultrasonic anemometer data has been shown to increase close to equipment or obstacles (Contini, Donateo, & Belosi, 2006). Contini used paired anemometers at different mounting orientations, spacing, and in proximity with other detectors to investigate uncertainties in measurements. The results showed that uncertainty increased when the anemometers were mounted on separate masts, or alongside other detectors or obstacles. Larger distances between two anemometers also yielded larger inconsistency in measurements.

Uncertainty was calculated using time averaged measurements of the sensors and comparing the results. To quantify these uncertainties, equations using uncertainty values γ from Gaussian distributions with equal variances and systematic uncertainties ω were derived for each anemometer as equation 1.1. The difference between these equations is denoted in equation 1.2 (Contini et al., 2006), and was considered the prediction of uncertainty for time-averaged datasets. The random differenced uncertainty component η assumes identical wind velocity at both locations, meaning that the difference in recorded wind velocities consisted only of systematic and random uncertainty. In Contini's 2006 study, this generalization appears reasonable for quantifying uncertainties, because wind velocity data were collected in very close proximity. In this work, wind velocities were recorded at distances up to 60 feet and actual wind velocity V_{actual} was expected to be different. Systematic uncertainty refers to uncertainties present in both sensors and can be attributed to sensor geometry and position. In this study, system uncertainty is well quantified by manufacturer supplied measurement tolerances, since both anemometers were positioned clear of obstacles and with the same transducer designs.

$$V_{output} = V_{actual} + \gamma + \omega \quad (1.1)$$

Where:

V_{actual} = Actual measurement (m/s)

V_{output} = Measurement output by anemometer (m/s)

γ = Random uncertainty extracted from Gaussian distribution (m/s)

ω = Systematic uncertainty (m/s)

$$D_X = \gamma_1 - \gamma_2 + \omega_1 - \omega_2 = \eta + \beta \quad (1.2)$$

Where:

$\eta = \gamma_1 - \gamma_2$ = Random uncertainties (m/s) obtained from data values of Gaussian distribution (mean=0 and equal standard deviation)

$\beta = \omega_1 - \omega_2$ = Difference in systematic uncertainties (m/s)

Contini et. al (2006) also found that uncertainty decreased with increasing averaging time, yielding impressively small uncertainties as low as 0.01 m s⁻¹ for an averaging time of 30 minutes. These results indicate good performance for meteorological applications – however, to provide control inputs in real-time, similitude of data at much shorter intervals is needed. For an acceptable resolution to be obtained in spraying applications corresponding to the fate of a spray droplet, filtering must occur over a couple seconds or less. Regardless, averaging wind data is a proven correction for

smoothing problematic or noisy data and was used in this study. Additionally, a means of evaluating the effects of filtering on measurement uncertainty needed to be addressed. A modified interpretation of Contini's uncertainty calculation was used for this evaluation where random uncertainty originating from sensor orientation is replaced by uncertainty derived from the filtering process. This is described in more detail in section 2.1.8.

A study in Oldenburg, Germany looked at sensor accuracy affected by transducer shadowing, the effect of added turbulence by the obstruction of airflow from transducers, and found a significant effect based on wind direction (Heinemann, Langner, Stabe, & Waldl, 1997). These uncertainties are of minimal concern in this study since both anemometers described in section 2.1.1 have the same transducer design and equal uncertainties documented in their user manuals. Ultimately obtaining the exact wind velocity is not of focus, but instead a calibrated response of nozzle orifice diameter to wind fluctuations.

Previous literature has studied the effects of sensor tilt on the accuracy of wind vector measurements, and correction methods for tilting have been studied as well (Landwehr et al., 2015). Translational motion correction, however, is much less researched. The dynamics of the anemometer on a moving platform needs to be subtracted from the observed wind velocity to obtain accurate measurements.

1.5.2 *Timestamping Wind Velocity Measurements*

To test corrections on measurements from an ultrasonic anemometer in-motion, a stationary anemometer is needed for reference. Before testing on-board a moving platform, validation of correlation between two stationary anemometers at the same fixed

distances was necessary for comparison. If there were no significant differences between wind velocities at a known spacing, then correlation of corrected measurements taken while in motion at the same or similar spacing to a stationary reference anemometer could help prove feasibility for sprayer applications. Before correlation testing for wind data from the two anemometers can be completed, the addition of temporal measurement data was needed for analysis and post-processing. Since the target application for the interfaced anemometers is outdoors, a reliable source of timekeeping is needed that can operate on DC power. A global navigation satellite-based system (GNSS) receiver is ideal for these requirements because it allows accurate timekeeping while recording the location of the moving platform if needed. A hardware method for capturing temporal data and concatenating with serial data streams was developed by Sama et al. (2013) that eliminates uncertainties caused by software latency, described as the delay between measurement and timestamping. They utilized a pulse-per-second (PPS) signal output from a GNSS as a reference to synchronize location readings from a tracking total station (TTS) using a signal timing device. The resulting data from experimentation showed acceptable precision in the timestamps for agricultural applications (devices moving 10 m/s or less). An input capture on the signal timing device's microcontroller was used to keep track of when the PSS occurred relative to a 3.75 MHz local clock source, which allowed incoming serial data to be timestamped to universal coordinated time (UTC) with an accuracy of less than 1 millisecond. A comparison between this hardware method to a software method of timestamping utilizing a PC's system clock was explored in this project during experimentation with the two stationary ultrasonic anemometers. To investigate similarity between measurements, a study of correlation between data from

two anemometers at temporal resolutions equal or close to droplet airtime from a sprayer boom's height needed to be studied.

1.6 Statistical Analysis for Similitude of Wind Data

Determining the feasibility of using weather data on-board a sprayer system heavily relies on proving similitude among wind data from a stationary anemometer and processed dynamic weather data. Correlation as a means of similitude for paired measurement is widely used in literature (Contini et al., 2006, Imai et al., 2009, Reid & Turner, 2001), however, these studies used filters over timespans of minutes or longer and have a significant reduction in noise. If weather data is to be used for real-time actuation of a variable-orifice nozzle, filtering must be completed at a much smaller scales, with the cost of increased noise. These noisier datasets were destined to yield poorer correlation for a classical pearson correlation test (See section 2.1.7), so alternate indicators of similitude are needed. In literature, a study using multiple weather stations to quantify the spatial structure of wind from a tower for wind power simulation purposes used cross-correlation to characterize the relationship between lateral and vertical components of wind gusts (Fujimura & Maeda, 2009). This approach took wind velocities from two anemometers to test for structured decay over lag times or the time difference between paired velocity measurements. In other words, multiple correlation tests were run on data pairs separated by increasing time between measurements. If cross-correlation was markedly improved with zero lag, then some degree of similitude can be assured. This is because time-synchronized measurements with better correlation

compared to measurements out of phase imply that small scale fluctuations match and are not spatially independent.

Contini et al. (2006) looked at differenced wind velocity values to quantify and analyze the distribution of random uncertainties. This work looks to compare wind velocities at two locations with much larger distances, and are not expected to be equal at any time. With random uncertainty replaced with filtering uncertainty, and systematic uncertainties added, a range of the potential wind velocity any time may be depicted similarly as uncertainty was in Contini's work. In this study, differenced wind velocities and derived uncertainties were used as an additional means of comparing wind intensity between two locations.

1.7 Previous Spray Drift Characterization Methods and Modeling

Some previous work and developed models were studied for considering wind effects on droplet drift which were developed using ultrasonic anemometers for quantifying drift per wind velocity. A few models for droplet trajectory simulate spray drift under input sprayer configurations. RTDrift (Lebeau, Verstraete, Stainier, & Destain, 2011) is a Gaussian plume model that allows input parameters for pressure, movement, and height of a spray nozzle. Equations for spray drift were derived from previous studies and used experimental wind data collected from ultrasonic anemometers mounted on a spray nozzle. Droplet spectra data were collected using a Phase Doppler Interferometer and were used to predict resulting drift and evaporation of liquids during trajectory was also accounted for. A Computational Fluid Dynamics (CFD) model, compared to experimental data, displayed success in predicting drift for varying fixed

nozzle diameters, wind velocity, height, and pressure of spray booms (Baetens et al., 2009). This model was validated also by using ultrasonic anemometers at different elevations. Important parameters for drift included the boom's height, wind speed, and nozzle-orifice size.

CHAPTER 2. CORRELATION TESTING BETWEEN TWO STATIONARY ULTRASONIC ANEMOMETERS

2.1 Methods and Materials:

Determining correlation between readings from two stationary ultrasonic anemometers was completed to help validate a claim that wind gust velocities measured in close proximity to each other are statistically similar. The purpose of this experiment was a feasibility study before moving forward with testing of the sensors onboard a moving platform. The experiment involved the design of a rigid stationary test fixture to mount both anemometers, a data acquisition Windows Form Application, a MATLAB script for processing data and performing statistical analysis, and also a SAS script for statistical analysis. Additional testing was completed using a GNSS receiver's pulse-per-second (PPS) signal in conjunction with output UTC time for improved precision of timestamps at a millisecond scale. The receiver was interfaced to add timestamps to the serial data stream output from the anemometers using a PCB board. A variety of weather conditions were observed and tested for correlation while also using different filtering techniques for noise reduction. A desired result for these tests was a successful filter that could be implemented in the time-domain for developing control inputs to the variable-orifice nozzle.

2.1.1 Ultrasonic Anemometer Interfacing:

Two ultrasonic anemometers (86000 and ResponseOne 92000, RM Young, Traverse City, MI) were selected to collect weather data including temperature, relative humidity, wind speed, direction, and air pressure as inputs for the system. A 13.8 V nominal power supply (1680, BK Precision, Yorba Linda, CA) was connected to supply the required power (10-30VDC) to both sensors, and output communication to a PC was configured using RS-232-to-USB converters (Keyspan U209-000-R, Tripp Lite, Chicago, IL). Using manufacturer provided configuration programs, the serial data output format was set to ASCII with a 38,400 baud rate, 8 data bits, no parity, and 1 stop bit (38400-8-N-1) for both anemometers and the units were set to metric.



Figure 2.1 Ultrasonic anemometers used in this study were manufactured by RM Young (Young)

Microsoft Visual Studio was used to develop a user interface and the supporting code writing in the Visual Basic language (VB.NET). Functions of the program included setting the serial port baud rate for data retrieval, assigning the ports for serial data collection, opening and closing the ports, a space to send commands if necessary,

automatic timestamping and parsing of data strings, display for incoming and parsed data, and the ability to start and stop logging to a .csv file. The code is included as Appendix 5 and the graphical user interface is shown in Figure 2.2.

At the program's startup, a Sub statement "Build_Interface" ran immediately to set up the user interface and display available COM ports for assignment to the anemometers. COM port names were automatically assigned by the RS-232-to-USB converter driver and set accordingly in the program. The remaining COM port settings were configured to match the anemometers' settings (38400-8-N-1). Before data were read and saved, a destination file was needed to store the information. The filename must be input into a text box, and then a save location must be chosen. A "FolderBrowserDialog" object was used to browse save locations and automatically create a .csv file with the input name once a destination is chosen.

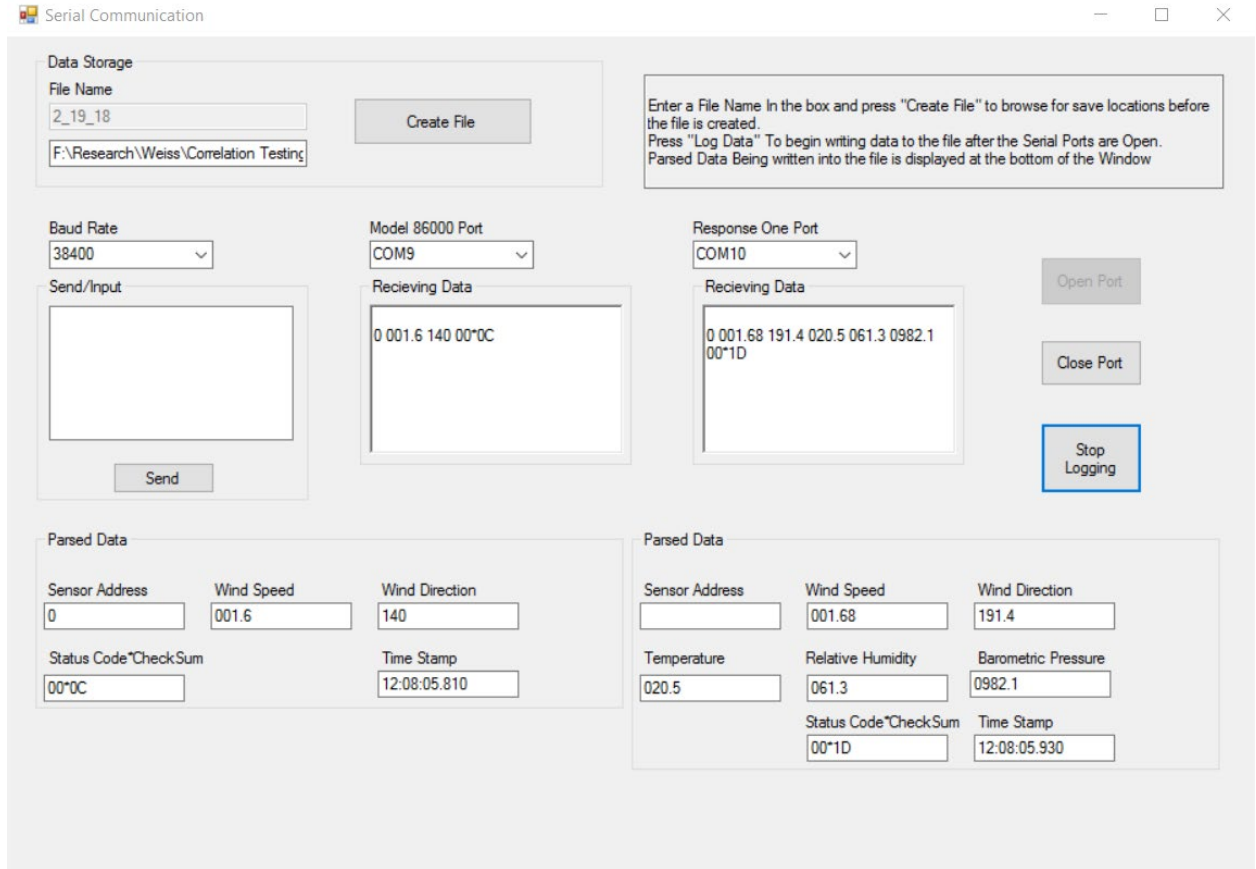


Figure 2.2 The Serial Communication Interface allows control of the data logging procedure and provides graphical display of ultrasonic anemometer sensor parameters

To read data from the serial ports, an existing RS-232.vb class written by Dr. Michael Sama was used to read available COM ports and list them in a ComboBox drop down menu. A button was created to initiate opening the serial ports for reading after being designated to each sensor. Once the ports were opened, data were received for each anemometer using separate subfunctions, with handles that call the subfunction when a full new message was received. A full message was defined in the RS-232 class as all characters between carriage returns (Figure 2.3). The class was also modified to utilize a Boolean logic variable to check for the arrival of a message, and subsequently call `DateTime.Now.ToString` for local timestamping at the arrival of the next bits. Each data

sentence was then parsed after arrival using spaces as the delimiter within Sub statements MessageReceived and MessageReceived2. These statements received and parsed sentences from the 86000 and 92000 models, respectively.

<u>92000 ResponseOne ASCII Polar Format:</u>	<u>86000 ASCII Polar Format:</u>
a www.ww ddd.d ttt.t hhh.h bbbb.b ppppp ss*cc<CR>	a www.ww ddd.d ss*cc<CR>
a	= Sensor address
www.ww	= Wind speed (m/s)
ddd.d	= Wind direction (degrees)
ttt.t	= Temperature (°C)
hhh.h	= Relative Humidity (%)
bbbb.b	= Barometric Pressure (hPa)
ppppp	= Tipping Bucket Count (optional)
ss	= Status code
*	= Asterisk (ASCII 42)
cc	= Checksum
<CR>	= Carriage Return

Figure 2.3: Data sentence format for the 92000 ResponseOne (Young) and 86000 (Young) ultrasonic anemometers from the user manuals. Each sentence is separated by a carriage return <CR>. This is used to separate sentences from each other. The class definition of carriage return in Visual Basic is "vbCr"

A log button with two states was created to start recording data as it is received. By default the button indicated that data were not being logged (inactive) and would cycle between logging (active) and not logging each time it was pressed. When a

message was received by the program, the subfunctions MessageReceived and MessageReceived2 executed, and the status of the logging button was checked. If active, the messages were recorded to the file created by the folder browser subfunction. The user interface was designed such that each functionality occurs independently. Opening and closing the ports, initiating the file generator, “FolderBrowserDialog”, and data recording were contained in separate sub statements so that each process’ success could be verified. An advantage to this arrangement was having the capability to precisely begin data logging at the press of the logging button rather than after the selection of the save location and file generation.

The output of this program was organized data from the 86000 and 92000 model ultrasonic anemometers. The data were labeled by sensor model so it could be separated and processed using MATLAB R2017a. Example data is shown in Table 2.1 from testing the data logging functionality. The checksums were included to validate that data streams transferred successfully. The timestamps are displayed on the right-most column.

Table 2.1 Output data from sensors can be separated in MATLAB by their sensor addresses.

Sensor Address	Wind Speed (m/s)	Wind Direction (°)	Temperature (°C)	Relative Humidity (%)	Barometric Pressure (hPa)	Status Code *Checksum	Time
86000	2.52	161				00*0F	12:07:59.810
92000	1.78	327.3	20.6	60.9	982	00*1D	12:07:59.920
86000	2.60	180				00*03	12:08:00.060
92000	1.75	235.5	20.5	60.9	982	00*17	12:08:00.170
86000	3.32	142				00*09	12:08:00.300
92000	2.11	235.3	20.6	61	982	00*1A	12:08:00.430
86000	2.57	161				00*0F	12:07:59.810

2.1.2 *Hardware Timestamping of Serial Data from Ultrasonic Anemometers*

The PC timestamps recorded along with incoming data were subject to latency. Data were output at regular intervals, whereas the PC timestamps indicated variability by as much as 10 ms. For a more accurate measure of similitude between two distanced stationary anemometers, precise timestamping was needed for collected wind data.

Initial testing utilized the Visual Studio VB.NET function `DateTime.Now.ToString("HH:mm:ss.fff")` to timestamp using a computer's clock synchronized over the internet. This function was triggered by a Boolean variable "TimeTrigger" that armed once a full message was identified. As characters were received, the state of "TimeTrigger" was checked such that when the first character in the next sentence was received, the time was recorded. While this method was effective for timestamping at high baud rates, it is limited by software latency and application in the field. Thus, a hardware method with improved mobility was explored.

A GPS receiver (GPS 18x LVC, Garmin) was used to append a UTC timestamp to each anemometer serial data message. A custom printed circuit board was developed to interface with the pulse-per-second (PPS) and serial data stream from the GPS, shown as Figure 2.4. The PPS signal was connected to an input capture on a microcontroller (dsPIC30F4013, Microchip) and provided the 1 second epoch within 1 microsecond. The serial data stream transmitted the NEMA 0183 GPGGA string, which included a UTC timestamp. PPS events recorded an internal timer value from a 58.58375 kHz clock source that was used to keep track of other events between PPS events. The first start bit of each anemometer serial data message triggered a second input capture to record the local timer value. A UTC timestamp was computed from the local timer value and

appended to the end of each anemometer data string in the same format as used for the previous software method of timestamping. The microcontroller program used to timestamp anemometer serial data is included in Appendix 8.

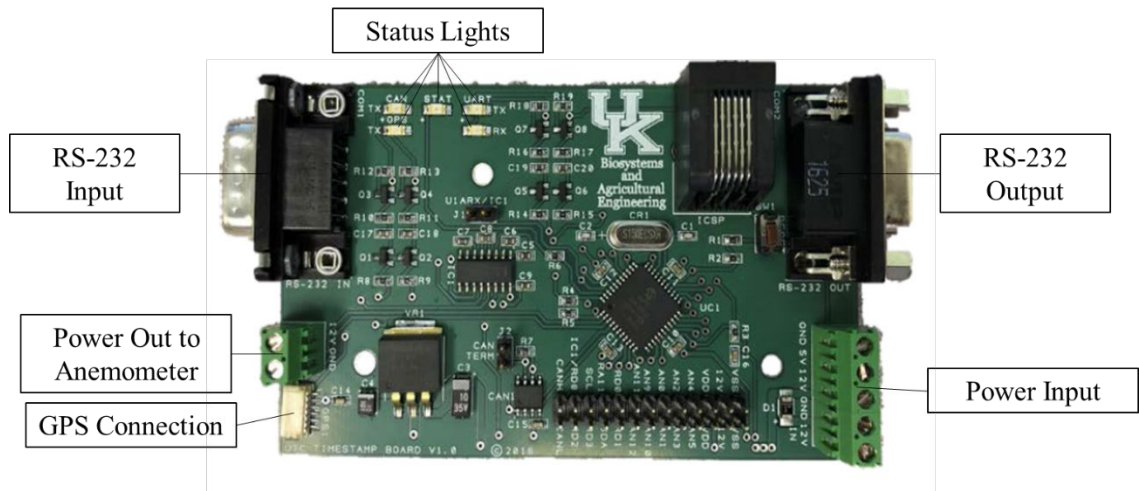


Figure 2.4 The assembled PCB allows a hardware method of timestamping

Adjustments were made to the Windows Form Application to handle hardware timestamped serial data streams. The hardware timestamped serial data streams replaced the carriage return with a carriage return line feed and contained a UTC timestamp concatenated between it and the calculated checksum from the anemometer. A space was maintained as the delimiter for separating data. Occasionally collected data would have a timestamp with 1000 ms after the decimal place. These rare instances were incorrectly read by processing scripts as 100 ms. Data with these incorrect timestamps were removed from the data series to simplify processing.

2.1.3 *Test Fixture Design and Assembly*

Important criteria for the test fixture included rigidity, durability, and ease of assembly. The test fixture (Figure 2.5) consisted of 3 pieces of 2x1 inch t-slotted aluminum extrusion framing, a custom machined aluminum tripod adapter, machined slug mounts for the sensors to tighten onto, as well as brackets and screws to hold the pieces together. The test fixture was designed to mount on a tripod using a standard 5/8"-10 machine screw. Test fixture components were modeled using Autodesk Inventor Professional 2018. The 2x1 inch framing was selected because of its relatively light weight and ease to assemble/disassemble and transport. The custom tripod adapter and support brackets were secured using t-slot fasteners. The frame provided reasonable resistance to torsion caused by long lever arms in the form of the vertical extrusions. The horizontal aluminum beam was 1.83 m long and each vertical beam holding the slug mounts was 0.91 m long. The height at the top of each anemometer with the fixture fully assembled was approximately 2 m. Slug mounts were each 15.24 cm long and 3.40 cm in diameter, allowing the fastening of the anemometers using included brackets by the manufacturer. The slug mounts and top end of the vertical extrusions were tapped to 1/4"-20 threads and secured with hex-head screws for consistency with the framing

brackets. Schematics of machined parts are displayed in Appendix 1.

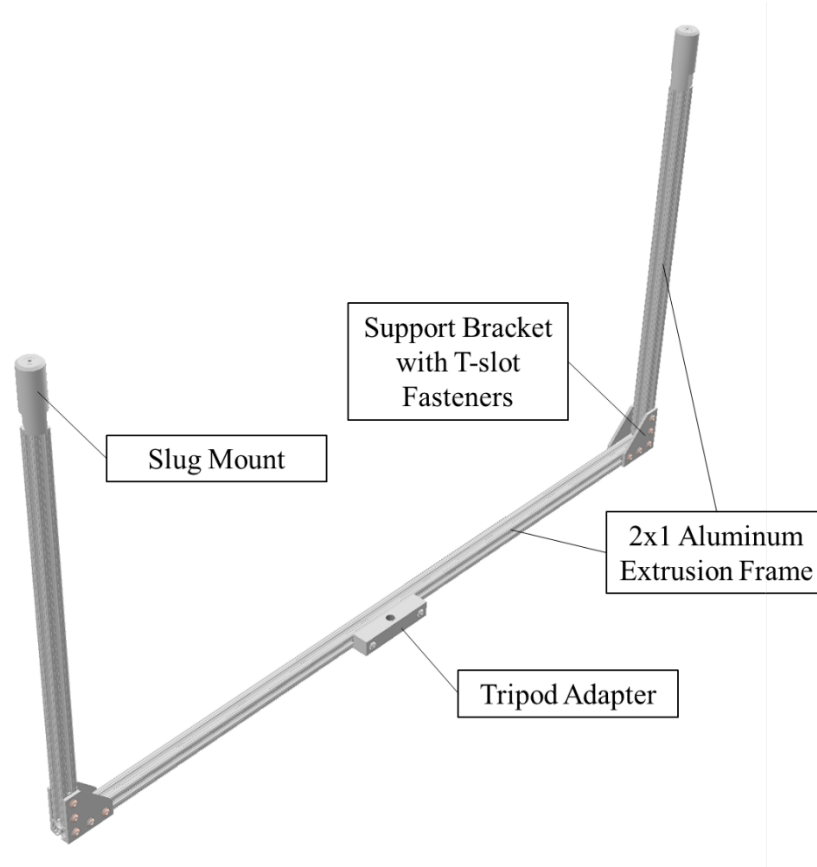


Figure 2.5: Assembly for Preliminary Test Fixture

The PCBs required an enclosure to protect components from inadvertent contact. The design for a 3D printed enclosure needed to provide easy access to connections on the PCB labeled in Figure 2.4 and also allow vision to the board's status lights for confirmation that the system was operating correctly. Aesthetics were also considered: providing indentations for printed labels, rounded edges and walls, and engraving of a University of Kentucky logo. The assembled PCB and case are shown as Figure 2.6. The lid and base contained three holes that were tapped for 1-inch length #4-44 rounded head

machine screws. The screws fastened through the lid, the PCB mounting holes, and into the base. Detailed drawings are presented in Appendix 1.

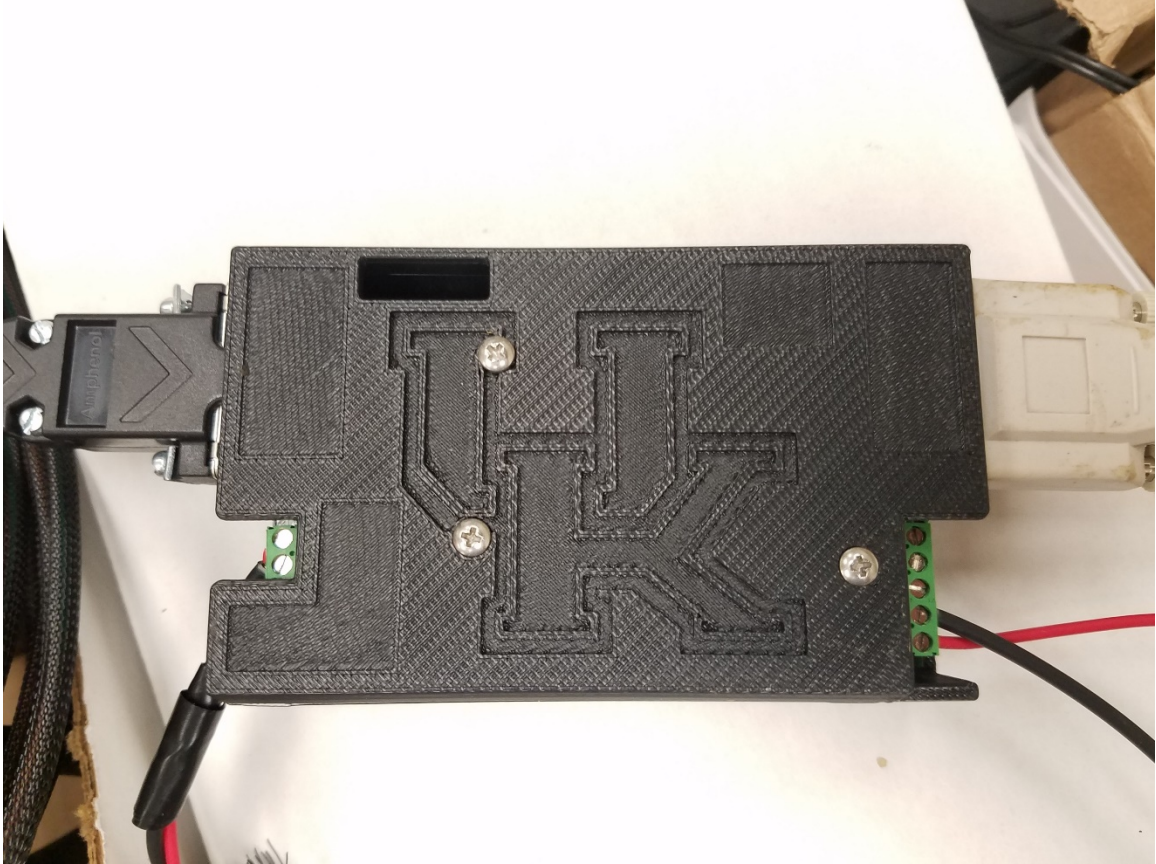


Figure 2.6 The assembled PCB and case

2.1.4 Preliminary Data Collection and Setup Procedure

Data were collected for the static experiment on the roof of the University of Kentucky's Charles E. Barnhart building. Each test was conducted at the same location on the roof at its edge at the coordinate 38.026924° , -84.509623° . A satellite image of the test site is displayed as Figure 2.7. The 12 V DC power supply was used to provide power to the sensors, while the data acquisition system was run on a Microsoft® Surface

3 tablet computer. Data were collected purposely on days with varying temperatures, humidity, and average wind speeds because they are known factors to effect spray drift. There was also no precipitation during any of the testing days and the usual test duration lasted between 1 to 3 hours, determinant on the possibility of rain or snow. The test fixture was installed at the determined testing location next to a cart (seen in Figure 2.8) supporting the laptop, power supply, PCBs, and Trimble GNSS receivers. For the hardware timestamping method, the housed PCBs were each connected to one ultrasonic anemometer and to a Garmin GNSS receiver.

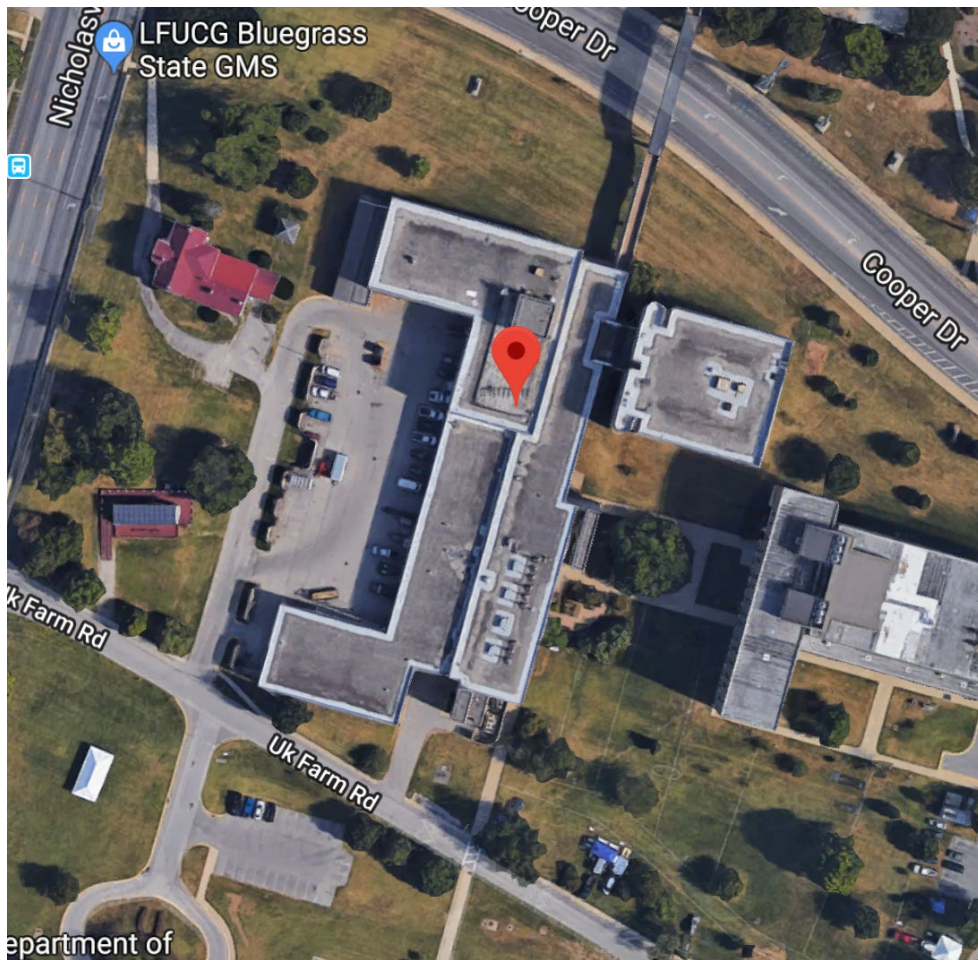


Figure 2.7 A satellite image displays the location of the test site at 38.026924° , -84.509623° (Google, n.d)



Figure 2.8 A cart was used as a surface for the computer, power supply, PCBs, and GPS receivers

The 12V power supply was first turned on to power the ultrasonic anemometers, custom PCBs, and GPS receivers. The status lights on the PCBs were observed to confirm that the equipment was functioning properly prior to logging data.

2.1.5 *Field Testing*

Experimentation at various separation distances between the two anemometers was completed to validate dynamic testing. If sufficient correlation could be achieved for both stationary and dynamic testing at a chosen distance, then a sprayer boom size equal to twice that distance (from the center to edge on both sides added) may have potential for drift mitigation based on weather data recorded at the boom's center. First a pilot test ranging from 3.66-7.32 meters (12-24 feet) was attempted. Due to data logging issues, many points were lost during writing to the .csv file, so correlation analysis was not pursued, however wind velocities appeared similar enough to continue at larger distances. Testing was then conducted at 6.10, 12.20, and 18.29 meter (20, 40, and 60 feet) separations. The maximum distance was chosen based on current spray boom widths of high-clearance self-propelled sprayers and represented one half of the total boom width.

The first field experiment was conducted at the University of Kentucky's C. Oran Little Research Center in Versailles, Kentucky (38.085714°, -84.734869°) on October 5th, 2018 from 14:16:00 to 20:07:00 UTC to examine correlation at various anemometer separation distances. A photo of the equipment setup is shown as Figure 2.9. The experiment was conducted as a completely randomized block design, in which 3 separation distances were tested with 3 replications each for a total of 9 trials. The order of testing for each replication was determined randomly as shown in Table 2.2 to capture wind variability throughout the day while distributing error from weather variability across all trials. The height for both anemometers was set at 194.3 cm checked using a tape measure. Despite dropped readings because of errors in data logging, the remaining wind velocity pairs were close enough to warrant testing at larger separation distances.

Table 2.2: Completely randomized block design for stationary testing

Block 1	Block 2	Block 3
5.49 m	7.32 m	3.66 m
7.32 m	3.66 m	5.49 m
3.66 m	5.49 m	7.32 m



Figure 2.9: Static testing was completed at the University of Kentucky's C. Oran Little Research center in Versailles, KY on October 5th, 2018.

Following this experiment, a second test of different separation distances between the anemometers was conducted at 6.10, 12.20, and 18.29 meter (20, 40, and 60 feet) to emulate larger commercial spray booms. The field experiment was conducted on relatively flat ground at the University of Kentucky's North Farm in Lexington, Kentucky (38.130583°, -84.493944°) on February 13th, 2019 between 18:15:00 to 21:15:00 UTC to examine correlation at various anemometer separation distances. The experiment was conducted as a completely randomized block design, in which 3 separation distances were tested with 3 replications each for a total of 9 trials. The order of testing for each replication was determined randomly as shown in Table 2.3 to capture wind variability throughout the day while distributing error from weather variability across all trials. A photo of the experiment setup is shown as Figure 2.10. Between each replication a digital level was used to check anemometer tilt to ensure they were horizontally level with an error of $\pm 0.2^\circ$ the ground plane. Also, a digital compass was used to align both anemometers facing North. This was completed by holding the compass up to the Northern indicator on each anemometer and is expected to have an error of roughly $\pm 2^\circ$ from North. The height for both anemometers was kept constant at 194.3 cm and was checked using a tape measure. The 6.10, 12.20, and 18.29 meter (20, 40, and 60 feet) trials were chosen to emulate larger commercial spray booms.

Table 2.3: Completely randomized block design for stationary testing

Block 1	Block 2	Block 3
12.2 m	18.29 m	12.2 m
18.29 m	12.2 m	6.10 m
6.10 m	6.10 m	18.29 m



Figure 2.10: Static testing at the University of Kentucky's North Farm was completed on February 13th, 2019.

2.1.6 *Processing Procedures*

After data collection, a MATLAB® script (R2017a) was utilized to filter out data with incorrect timestamps, separate data by sensor address, and interpolate data from the 86000 model to the time interval of the 92000 model anemometer. Interpolation of one dataset onto the time interval of the other was necessary to perform correlation testing because the anemometers were not temporally synchronized. Linear interpolation was chosen due to simplicity. The decision to interpolate the 86000 model's data onto the 92000 data was determined because the 92000 model was assumed to be a superior sensor. This was also the motivation for mounting the 92000 on the moving vehicle later in CHAPTER 3. Statistical analyses were completed using MATLAB, SAS 9.4, and Microsoft Excel.

Various moving-average filters were tested on the dataset to reduce noise, and correlation testing was done to determine the optimal technique yielding acceptable correlation while maintaining accuracy. The filters selected were 3, 5, and 7-point moving averages. At a 200 ms sampling rate, the filtering windows involve data over the time interval of 0.6 seconds to 1.4 seconds. The data were processed to create replications for correlation testing. Replications were defined as a pair of velocity measurements, one from each anemometer, at the same time instant. For example, a replication for testing raw data would include the recorded value of the model 92000 weather station and the interpolated value of the 86000 model for that instant. Additionally, data was arbitrarily categorized by wind velocity as “low” if less than 3 m/s, “medium” if between 3 and 6 m/s, or “high” if higher than 6 m/s. This categorization was completed so that correlation could be compared at different velocity ranges. It also allowed testing of replications of

mixed categories, or large differences in the paired data. Using these tests, the plan for correlation results for each separation distance and filter was to calibrate the results as a reference to a “worse case” result. With the minimal Pearson correlation coefficient known for each separation distance, it could be used as a benchmark for comparison in dynamic testing. After completing Pearson correlation testing, flaws in its appropriateness at long distances were discovered which are discussed in section 2.1.7.

2.1.7 *Assumptions and Validity of Correlation Analysis*

The overlying assumptions for correlation analysis need to be carefully considered. For short time intervals, stationarity of the weather data is assumed because noticeable differences in weather are unlikely to occur. Through the assumption of stationarity (no change in distribution over time), an assumption of homoscedasticity (constant variance) is also made. This can be confirmed by scatterplot visualization of wind speeds for both anemometers but is somewhat arbitrary since the range of wind velocities is expected to be small with no trend over short timespans. The design of the experiment served to better satisfy the concern of nonstationary by collecting data at short time intervals of approximately 15-minutes. The correlation test assumes constant variance for this timespan rather than for all collected data for each anemometer separation distance. In summary, the analysis staves off concerns surrounding weather variability changes throughout the day by a completely randomized block, which breaks up this potential trend among all trials. Normality was tested for anemometer data using R-Studio (Version 1.1.456) using installed packages (xlsx, gstat, sp, and lattice) and was determined sufficient based on calculated residuals, skewness, and kurtosis.

A Pearson correlation test serves to test for linear dependence between two variables. After testing for Pearson correlation, it was determined that further analysis was needed to validate correlation as a means for determining similitude between the anemometers, especially for longer physical separation distances. The reason for this additional validation becoming necessary was because of small Pearson correlation coefficients at increasing distances, even though differences in measurements were generally not large and likely acceptable for making droplet spectrum decisions. Considering velocity measurements are taken at separate locations, it is expected that wind velocity will be different. If regression were to be calculated for each data series, poor models (low r value) would be expected. Therefore, a perfect linear dependence between them is not expected either. Alternatively, one strategy for determining similitude is to compute covariance between wind velocities at both locations however, the combined effects of unreduced noise and uncertain variability at different vehicle velocities and distances could yield incomparable variance during dynamic testing. For comparison purposes, the normalized cross-correlation coefficient between the two time-series data was computed, and the resulting cross-correlograms examined. The advantage to this correlation analysis is that focus is shifted towards detecting agreements in wind fluctuation at both locations rather than whether the velocity was the same. An analysis on the actual differences in wind velocities was carried out separately to the correlation analysis.

For cross-correlation testing to be meaningful, there must be both an individual structured auto-correlogram for each anemometer, and a structured cross-correlogram representing both sensor's data. Although the relationship between sequential

measurements is not of interest here, if there is a structured cross-correlation, and it is strongest with both series in phase (lag=0) with decay into increasing lag times, then it suggests merit in the cross-correlation analysis on paired measurements, because comparisons at the same instant yield better similitude than with measurements separated by short time lags. A high cross-correlation coefficient would suggest that wind velocity at both locations fluctuate synchronously. Additionally, by observing the cross-correlogram, the variability of wind velocity can also be observed by noting the rate of correlation decay across lag times. A steeper drop in cross-correlation can suggest distinct matching wind variability while a flatter slope suggests less similarity over the time period tested.

2.1.8 *Quantifying Uncertainty in Processed Wind Velocities*

Two sources of uncertainty were considered regarding wind velocity measurements between the two sensors. The first source was systematic uncertainty, defined in this work as the manufacturer provided 2% tolerance given by each anemometer's specification. The second was inspired by Contini et al. (2006). The study looked at quantifying random uncertainties related to sensor orientation and prevalence of obstacles. For this study, nozzle control per wind velocity data will require calibration to translate velocity readings to appropriate droplet spectra. Therefore, predicting random uncertainties to obtain statistically very precise estimates of true wind velocity (subtracting error caused by obstacles) is not necessary because ultimately the data will be used to make decisions at a different location. Instead, Contini's random uncertainty component γ was repurposed to encompass uncertainty derived by the filtering method.

Rather than using standard error, the standard deviation of measurements for each filtered measurement was used for uncertainty because it is better suited for describing variability amongst those measurements, as opposed to incorrectly describing results as a statistically derived estimate. The modified form of equation 2.1, expressing actual wind velocity as the sum of output data, filtering uncertainty, and systematic uncertainty is shown as equation 2.2.

$$V_{actual} = V_{output} \pm (\gamma + \omega) \quad (2.1)$$

Where:

V_{actual} = Actual measurement (m/s)

V_{output} = Measurement output by anemometer (m/s)

γ = Uncertainty in measurement derived from filtering ($\gamma = \sigma$ (m/s))

ω = Systematic uncertainty (Manufacturer provided 2% of output wind velocity (m/s))

Differences between measurements were calculated using equation 2.2 to evaluate the accuracy and similitude between two anemometers at varying separation distances. The filtering uncertainties for each sensor were added rather than subtracted to obtain uncertainty limits for each data pair's difference. For this portion of the analysis, the manufacturer's systematic uncertainty was excluded because it was the same for both sensors, and we are more concerned with filtering effects on accuracy. The resulting uncertainty equation is the spatial uncertainty (differences in velocity) added to the filtering uncertainty (standard deviation over filter's window). These differenced velocity

values α were placed into bins of 0.1 m/s in width, and cumulative probability plots were generated for each trial to supplement the correlation analysis. Additionally, the filtering uncertainty was organized into bins to compare filtering window size contribution to the measurement uncertainty.

$$\text{Uncertainty in Measurement} = (\alpha_1 - \alpha_2) \pm (\gamma_1 + \gamma_2) \quad (2.2)$$

Where:

α_i = Measurement value for anemometer i (m/s)

γ_i = Filtering uncertainty for anemometer i (m/s)

$(\alpha_1 - \alpha_2)$ = Spatial Uncertainty (m/s)

$(\gamma_1 + \gamma_2)$ = Filter Uncertainty (m/s)

2.2 Results:

2.2.1 *Visualizing Data and Filter Uncertainties*

The processed wind velocity and directional data was plotted in Excel with uncertainty bars representing filtering uncertainty and systematic uncertainty. An example dataset is visualized in Figure 2.11 and Figure 2.12, showing data from block 1, 20 ft separation filtered using a 3-point moving average. Filter uncertainty or standard deviation for used points in the filtering window is shown as error bars in the velocity

data. Uncertainty for wind direction was not derived because it was used for comparison and validation sake, and not for statistical analysis.

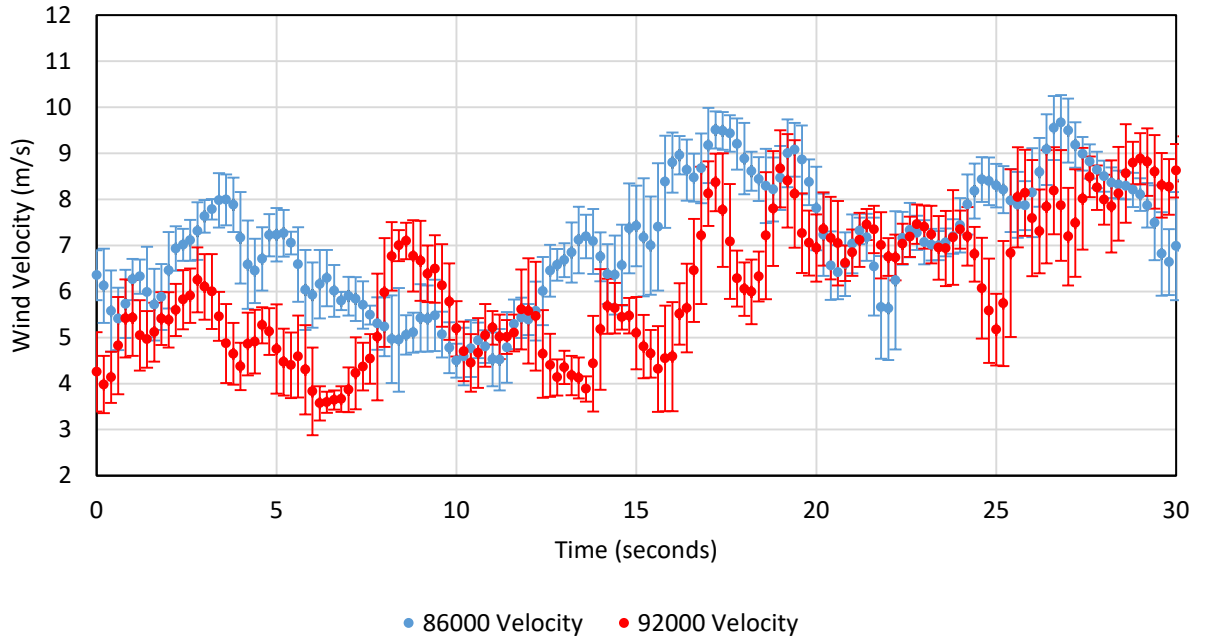


Figure 2.11: Wind velocity data is shown for the first 30 seconds of data at 20 ft separation, block 1, filtered using a 3-point moving average

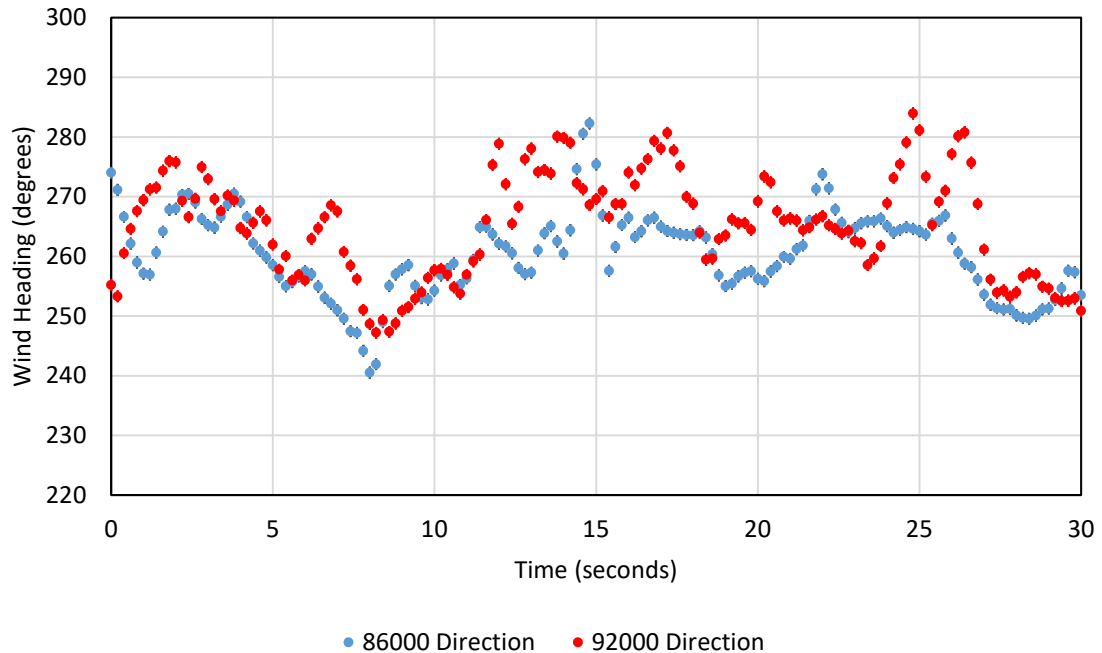


Figure 2.12: Wind direction is shown for the first 30 seconds of data at 20 ft separation, block 1, filtered using a 3-point moving average

2.2.2 Correlation Results:

The results of an initial Pearson correlation test using SAS 9.4 showed decreasing correlation with increasing distance between anemometers. This result was expected however R-values seemed mediocre averaging about 0.512 for 20ft, 0.353 for 40ft (the 3rd trial was 0.18 and very noisy), and 0.327 for 60ft. To examine whether these results were indicative of the degree of similitude between the sensors, cross-correlation among lag times were tested. Each lag step is equal to the sampling interval of 200 milliseconds, and plots were generated to 50 lags (10 seconds). This time interval was chosen because this work aims to examine feasibility of real-time adjustments of a nozzle, so timespans longer than a few seconds are irrelevant for droplet control.

Auto-correlograms such as the example shown in Figure 2.13 were generated for the 20ft, 40ft, and 60ft trials to validate structure before computing cross-correlograms such as in Figure 2.14. The results of these tests indicate structured correlation between the two anemometers that flatten with increasing lag times. The lag time is the offset which the data is temporally shifted out of phase and tested for correlation. Correlation decayed at different intensities depending on distance. These structures were examined for each of the 3 separations distances with various moving average filters as well. As expected, the correlogram structures are smoothed with increasing averaging time because velocity measurements are smoothed while also improving correlation among lag times. For all trials and filters, generated auto-correlograms showed structure that warranted analysis by cross-correlation.

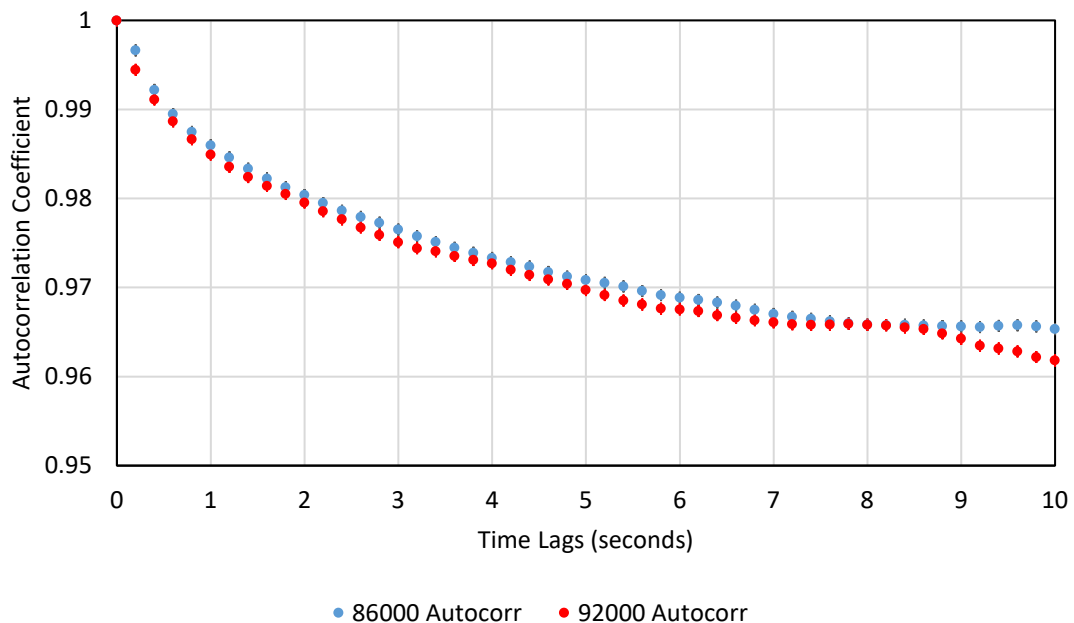


Figure 2.13: Autocorrelograms for 20ft separation, trial 1, show structured autocorrelation warranting a cross-correlation analysis.

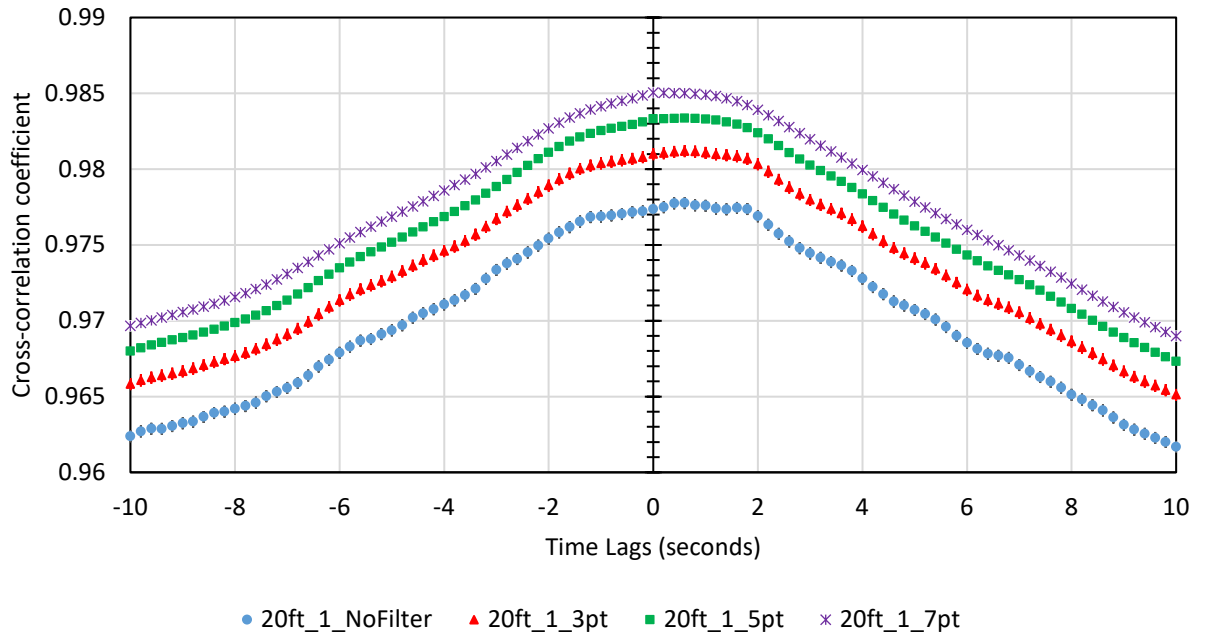


Figure 2.14: Cross-Correlogram for 20ft separation, trial 1, shows structured decay in correlation with increasing lag

Figure 2.14 displays the cross-correlogram for data collected with a 20ft separation between anemometers for multiple filters. The figure shows a clear decay in correlation with lag time, suggesting that both anemometers experienced fluctuations in wind velocity in-phase. At larger separation distances this phenomenon diminished. Table 2.4 displays a summary of cross-correlograms and a descriptor of their structure. “YES” indicates structured decay, while “Flat” indicates small slop of decay. In Figure 2.15, cross-correlograms for block 1 trials is displayed. The cross-correlation coefficient at zero lag decreases with growing distance, but also decays at slower rates. This is likely explained by an increased occurrence of larger differences between the two sensors overall, thus causing smaller slopes in correlation when the series are out of phase. These

trends were apparent among all blocks as shown in Figure 2.16 and Figure 2.17 however, in the later it's worth noting that the 40 ft trial is almost completely flat. When considering the poor 0.18 Pearson coefficient for this trial mentioned at the beginning of this section, this trial supports the methodology of identifying noisy or largely different data series by their flat cross-correlograms. Such phenomenon was considered during further analysis of velocity differences between the anemometers.

Table 2.4: Unfiltered trials generally all had structured decay in cross-correlograms except for one trial.

Trial	Structure?
20 ft, block 1	YES
20 ft, block 2	YES
20 ft, block 3	YES
40 ft, block 1	YES
40 ft, block 2	YES
40 ft, block 3	Flat
60 ft, block 1	YES
60 ft, block 2	YES
60 ft, block 3	YES

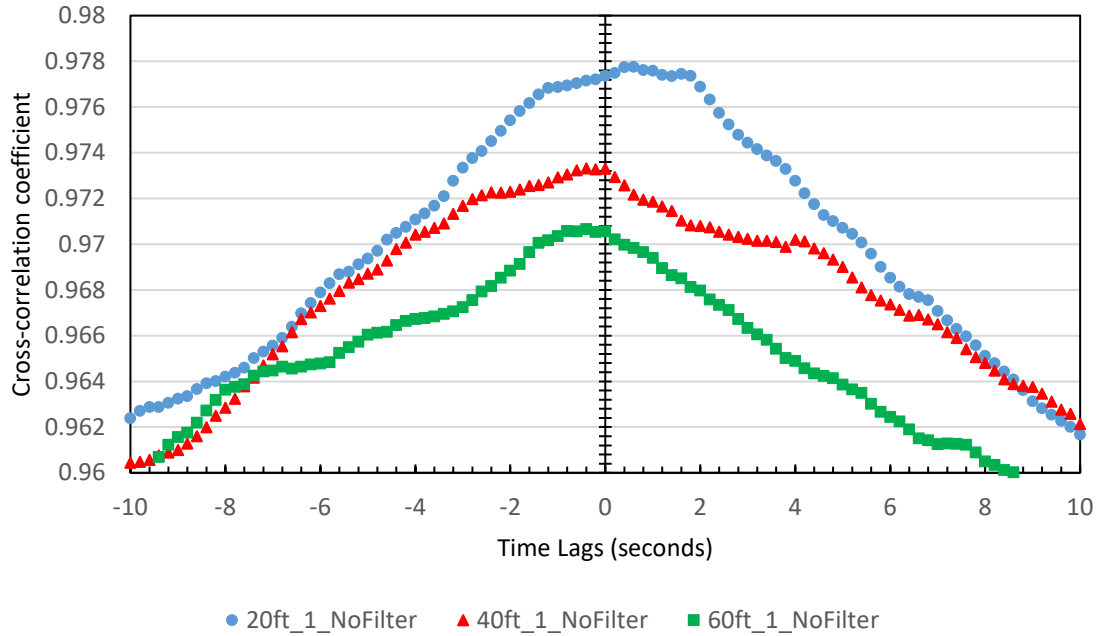


Figure 2.15: Cross-Correlogram for all separation distances in block 1 shows decreasing correlation with increasing distance and flatter structure, suggesting less similarity in wind velocity with increasing distance on the scale of spray boom size

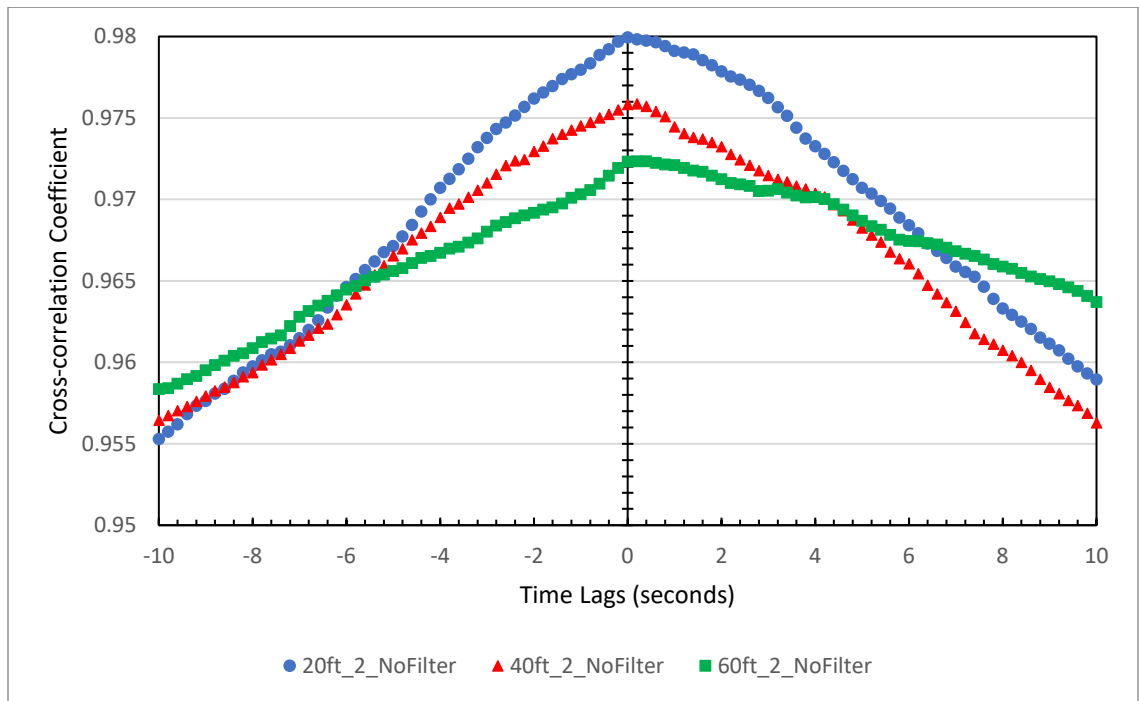


Figure 2.16: Cross-correlogram for all separation distances in block 2 shows decreasing correlation with increasing distance and flatter structure, suggesting less similarity in wind velocity with increasing distance

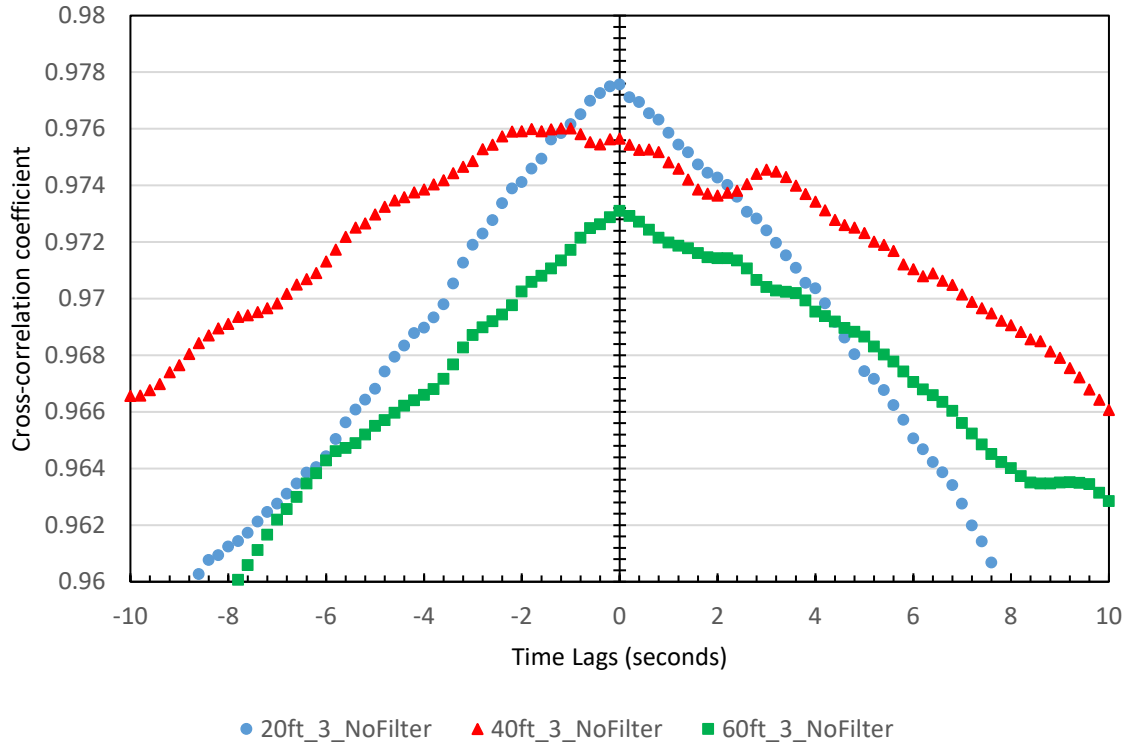


Figure 2.17: Cross-correlogram for 60 ft separation, trial 3, shows similar trends as Figure 2.17 and Figure 2.18, but displays the highly varied 40 ft trial.

2.2.3 Distribution of measurement differences between the two anemometers:

To further quantify similitude between paired measurements, differences between measurements at the same instant were calculated and categorized into bins for cumulative probability analysis. Each bin was 0.1 m/s wide, and the frequency of velocity differences were calculated for each bin. From this data, cumulative probability data was calculated representing the percentage of values equal to or less than a given difference in velocity. An example chart is displayed as Figure 2.18. In this example, approximately 90% of velocity pairs were less than or equal to 2 m/s. Increasing filtering size lessened the differences between measurements for all trials, acting as a smoother over time. The

trend is easily recognizable by the positioning of each filter's series relative to each other. The filter with the steepest initial slope was a 7-point moving average, indicating that the distribution of differenced values was more right-tailed and had a higher percentage of small differences, and less large differences between wind velocity series compared to smaller filtering windows.

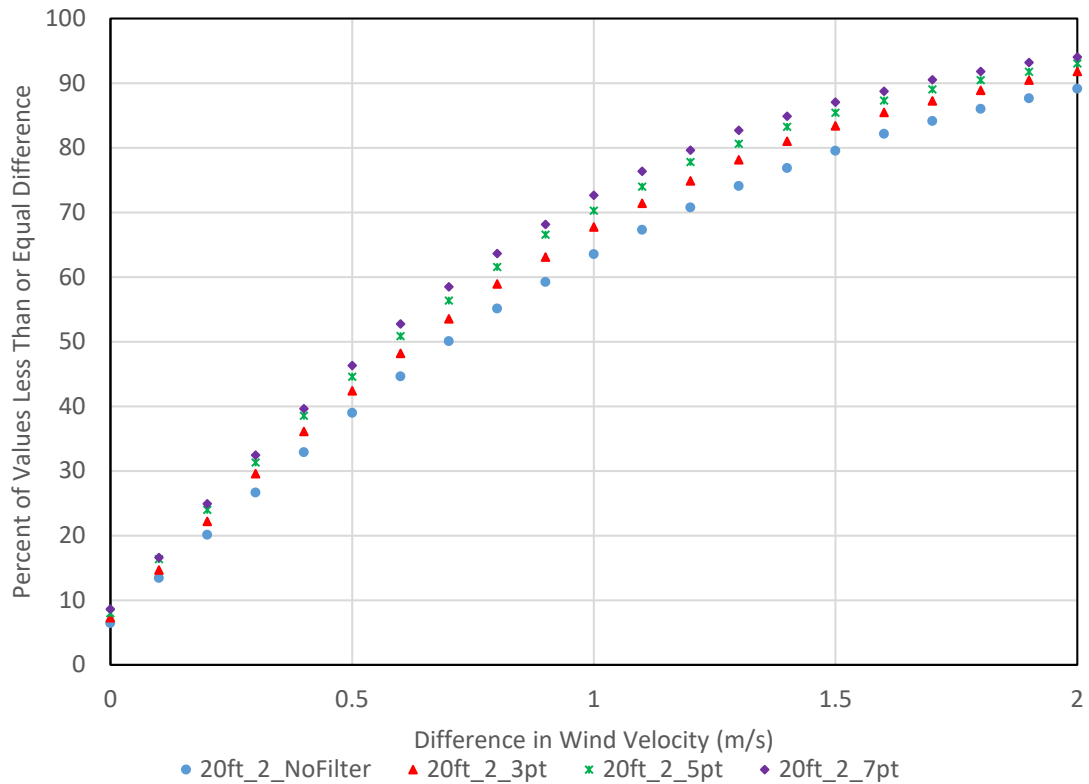


Figure 2.18: Cumulative probability chart for 20ft separation distance, trial 1

The cumulative probability plots for unfiltered data for all trials is displayed as Figure 2.19, Figure 2.20, and Figure 2.21. Each of the three trial blocks had increasing

amounts of larger velocity differences with increasing distance between sensors, except for block 3 where the before-mentioned noisy 40ft trial occurred. This trend of decreasing similarity at larger distances between the anemometers was considered for comparison to dynamic testing in Chapter 3.

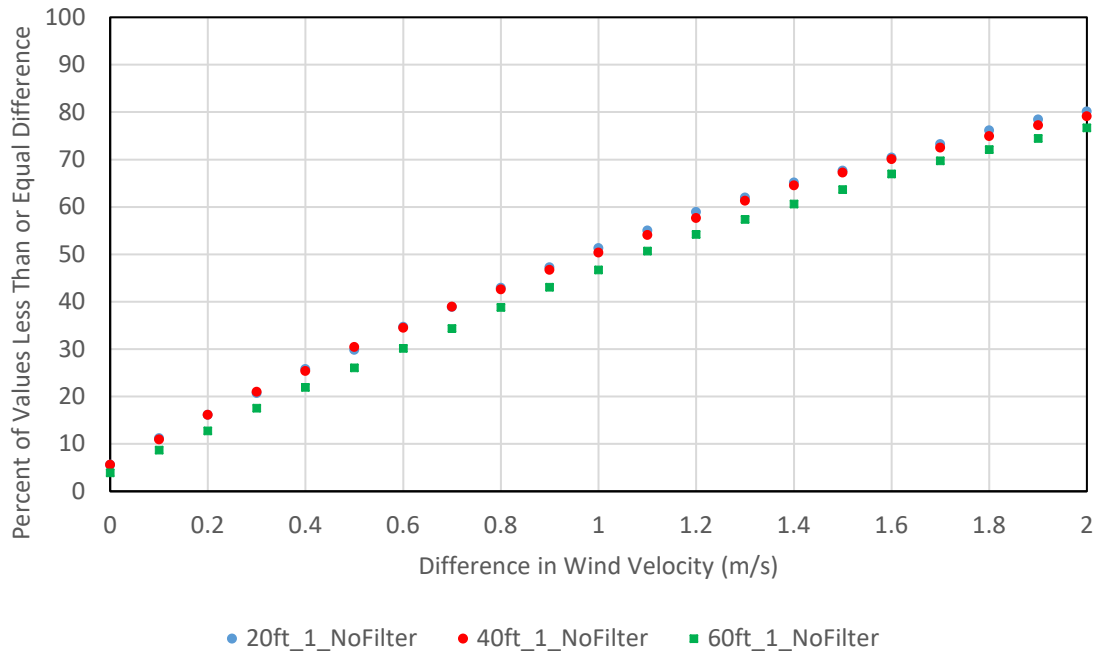


Figure 2.19: Cumulative probability plot for all trials in block 1

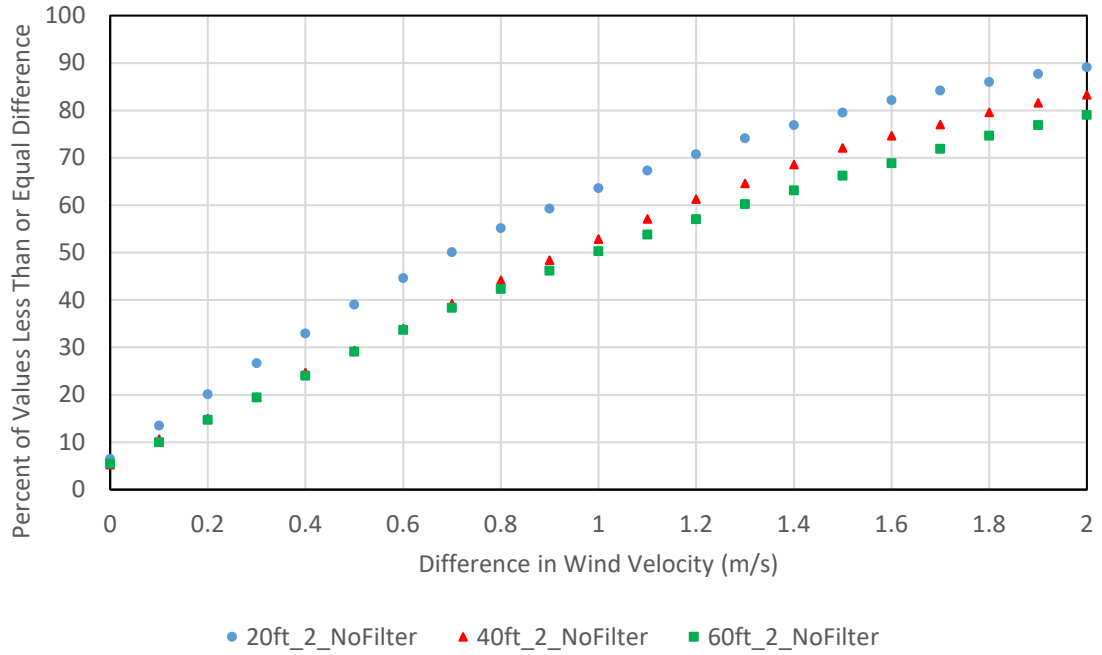


Figure 2.20: Cumulative probability plot for all trials in block 2

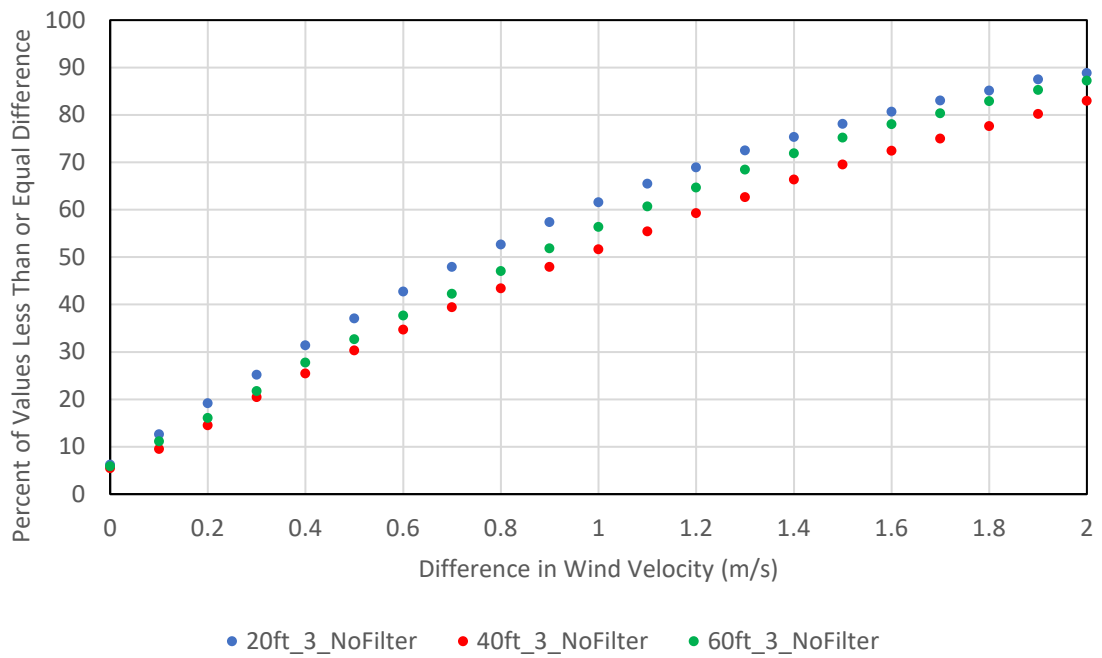


Figure 2.21: Cumulative probability plot for all trials in block 3

2.2.4 *A Note on Filtering Uncertainty Analysis by Probability Distribution*

An ideal filter is one that reduces noise while preserving the desired signal. Improved cross-correlation results for increasing filter size was expected because each dataset is smoothed by the filter, and noise reduction improves similitude. The calculated filtering uncertainty or standard deviation of points included in each filter's window was organized into histograms for comparison among filters as shown by example in Figure 2.22. Generally, among the three filters as filter size increased, the uncertainty in each processed result seemed to increase, but was determined too close, and with too many inconsistencies to determine as a definite trend. This was not surprising because larger windows allow more fluctuations to be included in the calculation of each filtered point, but in less noisy datasets could deceptively not be present where the range of values in the window were smaller. If replicated, this method of analysis should be completed with caution and while consulting a visual of the dataset. It was not replicated in this work for dynamic testing.

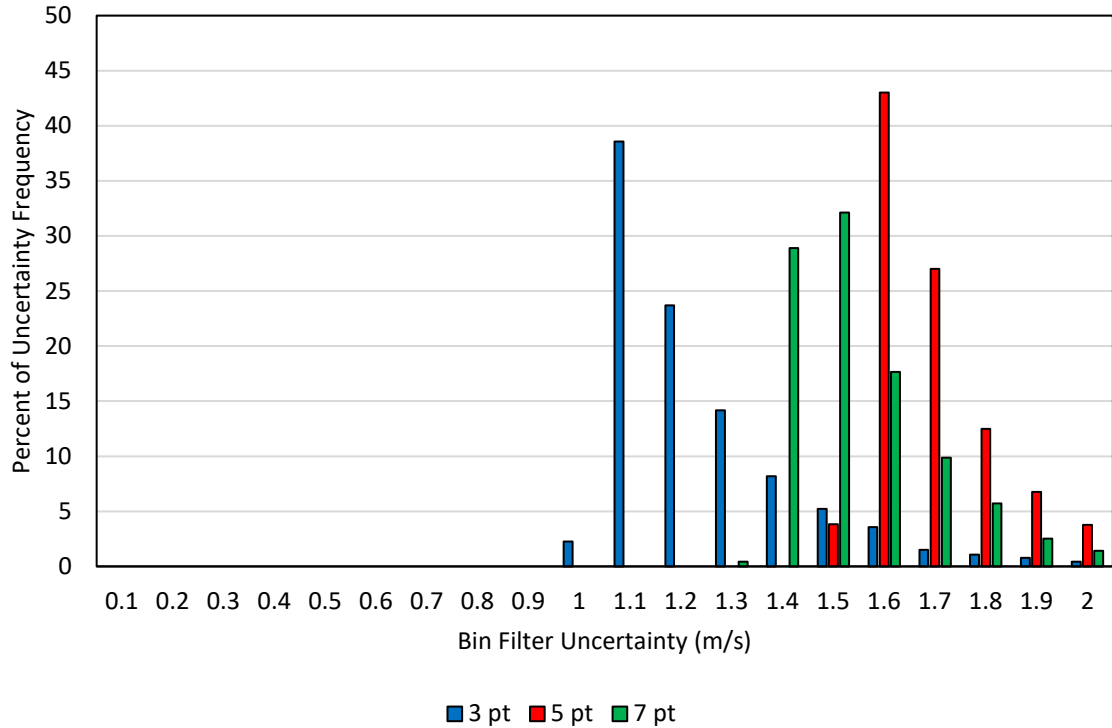


Figure 2.22: Sample histogram of filtering uncertainties distribution for block 1, 20 ft separation

2.2.5 Conclusions from Static Testing

The goal for static testing for similarity between to wind velocity at separate locations was to determine feasibility for dynamic testing. Before a processing algorithm for removing vehicle dynamics could be developed, and the resulting data compared to a static anemometer, it was important to first check that wind velocities at the distances of the same scale were similar. The resulting analysis of this chapter served as a form of benchmarking for what was to be expected in the dynamic experiment’s results.

At the conclusion of the static data analysis, it was clear that a Pearson correlation coefficient was insufficient alone for determining similitude, especially at larger

separation distances. Intuition says that this measure becomes increasingly more inappropriate because differences in wind velocity are expected at longer distances. Instead, focus was shifted toward quantifying whether wind velocity fluctuated together, as opposed to comparing precise values at each location. Cross-correlograms suggested that filtering improved similarity between the two series, and an analysis of differenced velocities confirmed it. Increasing distance between the anemometers resulted in smaller cross-correlation at zero lags as the wind varied more. Flattening of cross-correlograms was explained by both higher differences in wind velocities and increased prevalence of non-matching fluctuations. Especially noisy datasets masked some matching fluctuations between the two sensors.

CHAPTER 3. CORRELATION TESTING BETWEEN ONE DYNAMIC AND ONE STATIONARY ANEMOMETER

3.1 Methods and Materials:

Once similitude between static sensors was confirmed, dynamic testing was completed in comparison. The location of these tests was at the University of Kentucky's North Farm in Lexington, KY (38.130583°, -84.493944°) on an open parcel of land with no trees or structures within several hundred meters. For three separation distances and three ground speeds, a vehicle mounted weather station was driven in circles around a stationary anemometer to test whether this work's processing algorithm could successfully back-out vehicle dynamics from the apparent wind velocity measurements and obtain good agreement with the stationary data. The purpose for using multiple vehicle velocities was to examine the feasibility and differences in the algorithm's performance in addition to varying separation distance. Since cruise control was not available on the vehicle, vehicle speed was not constant for the duration of each trial. For the remainder of this work, the intended velocity of vehicle is referred to as the target velocity what was indicated by the vehicle's speedometer and monitored by the driver in effort to maintain the target velocity. The design was a completely randomized block, in which the order of testing for the radii of the vehicles path were completely random for a total of 3 trials each. Within each radius, targeted velocities were randomized as well. The advantage to this design allowed any temporal weather variability throughout the data to be distributed randomly among all trials. The tested radii for the vehicle's path were 6.10 m, 12.2 m, and 18.29 m, and the target velocities were 4.83 km/h, 9.66 km/h,

and 14.48 km/h. Two of the 6.10 m, 14.48 km/h trials were cancelled during experimentation because of increasing difficulty driving at that speed without sliding and digging ruts in the field. The order of testing is shown in Table 3.1, and an aerial photo of the experiment captured with a drone is displayed as Figure 3.1.

Table 3.1: The completely randomized block design is shown. The procedure moved down each block in order, completing separations from top to bottom, and vehicle velocities from left to right. Two of the 14.48 km/h trials were omitted because of difficulty turning the vehicle at that speed.

	Block 1				Block 2				Block 3			
6.10	4.83	14.48	9.66	18.29	9.66	4.83	14.48	12.19	4.83	14.48	9.66	
m	km/h	km/h	km/h	m	km/h	km/h	km/h	m	km/h	km/h	km/h	
18.29	14.48	9.66	4.83	12.19	4.83	9.66	14.48	6.10	9.66	4.83	14.48	
m	km/h	km/h	km/h	m	km/h	km/h	km/h	m	km/h	km/h	km/h	
12.19	4.83	14.48	9.66	6.10	14.48	9.66	4.83	18.29	14.48	4.83	9.66	
m	km/h	km/h	km/h	m	km/h	km/h	km/h	m	km/h	km/h	km/h	



Figure 3.1: Aerial photo visualizing the three radii for dynamic testing

A goal for this experiment was to remove components of the anemometer's output wind velocity caused by the motion of the sensor itself and quantify the degree of similitude with a stationary anemometer. To accomplish this, the vehicle's velocity and heading were calculated using three methods for comparison, from the output of a survey grade GNSS receiver operating in RTK mode. This calculated velocity was interpolated to the sampling interval of the 92000 model anemometer and were subtracted from the apparent wind velocity vector. The direction of wind data relative to the vehicle's heading was also recalculated using the newly solved wind vectors. The determined

vehicle heading was used to rotate the wind data relative to “True North” to match a stationary anemometer for statistical analysis.

3.1.1 *Equipment Setup and Vehicle Mounting*

A Trimble R10 GNSS receiver was used to record the vehicles position. NMEA GPGGA and GPRMC were output at 20 Hz and included UTC time, UTC date, latitude, longitude, velocity, and course over ground, among other parameters. The GNSS receiver was mounted on the roof of a utility vehicle (XUV 855D, John Deere) using a bracket mounted to a t-slotted aluminum framing cross member. The 92000 model ResponseOne weather station was also mounted to the cross member. The forward and sideways offset between the devices were measured for velocity transformation from the GNSS location to the weather station’s location. To reduce potential effects of tilt or vibration on measurement accuracy of the moving anemometer, testing was done on flat land. Applied moving average filters also hoped to reduce noise that may be caused by noticeable bumps on the vehicle’s path.

3.1.2 *Methods for Calculation of Weather Station Velocity*

Even though the GNSS receiver and weather station were mounted closely on the vehicle used, efforts were made in this study to derive an algorithm to transform position and velocity data from the GNSS location to the location of the weather station. At a separation of a few inches, the difference in velocity between the GNSS receiver and weather station may be considered negligible however, at larger distances or high turning rates this would not be the case. The velocity of the vehicle compared to at the end of a large spray boom however, may be notably different especially while turning. To provide

feasibility for estimating anemometer velocity located at larger distances from the GNSS receiver (i.e.: at the end of a large spray boom) an algorithm for transforming GNSS velocity to the position of the weather station was necessary to obtain the velocity of the weather station.

Three methods for calculating the velocity of the weather station were investigated using GNSS data and compared to each other. The justification for this approach was to provide alternative methods of velocity calculation that are dependent on different GNSS parameters available in the GPGLA and GPRMC messages. The first method, dependent on the GNSS fix precision, involved using GPS coordinates and the known offset between devices to calculate new coordinates at the weather station's position. The Haversine distance formula (Robusto, 1957) was used to calculate the distance between the two coordinates, and velocity was obtained by dividing by the sampling interval of the GNSS receiver. Vincenty's formula was also used to calculate new coordinates at the 92000 model's position and to calculate velocity for comparison. The second method used the GNSS receiver's "Speed Over Ground" and "Course Over Ground" outputs. The velocity at the location of the weather station was determined by modeling vehicle motion as a curved path and solving for weather station by multiplying GNSS velocity by a scalar multiple equal to a ratio of turn radii at each location. The third method utilizes the Haversine/Vincenty methods to calculate the GNSS velocity, however, uses the relative motion algorithm in method 2 to transform velocity data to the weather station's location. A comparison of the three methods is discussed in section 3.1.8.

3.1.3 Method 1: Successive Coordinates

Perhaps the most intuitive method of calculating velocity from GNSS data is to calculate the distance between successive coordinates and divide by the known sampling interval. It is important to recognize that GPS coordinates are spherical coordinates, and that the coordinate system they are based on has a non-uniform grid size therefore, Pythagorean distance is an incorrect estimator of distance unless coordinates are transformed. Two methods of solving for the distance between coordinates were explored.

The Haversine “Great Circle” distance was first solved instead. Before finding the Haversine distance, new coordinates at the weather station’s position needed to be calculated. The following equations represent a clockwise path, but the opposite direction (used in this study) was easily derived geometrically. First, the azimuth forward bearing was calculated according to equation 3.1. Next, the bearing facing the weather station from the GNSS receiver is solved using trigonometry in Equations 3.2 and 3.3. A legend defining the terms equations 3.1-3.10 is displayed as Figure 3.3, and the parameters are visualized in Figure 3.2.

$$\theta = \tan^{-1}(\sin(K_i - K_{i-1}) * \cos(L_i) , \cos(L_{i-1}) * \sin(L_i)) \quad (3.1)$$

$$- \sin(L_{i-1}) * \cos(L_i) * \cos(K_i - K_{i-1})$$

$$\varphi = \tan^{-1}\left(\frac{Y}{X}\right) \quad (3.2)$$

$$\theta_{G \rightarrow w} = \theta - (90 - \varphi) \quad (3.3)$$

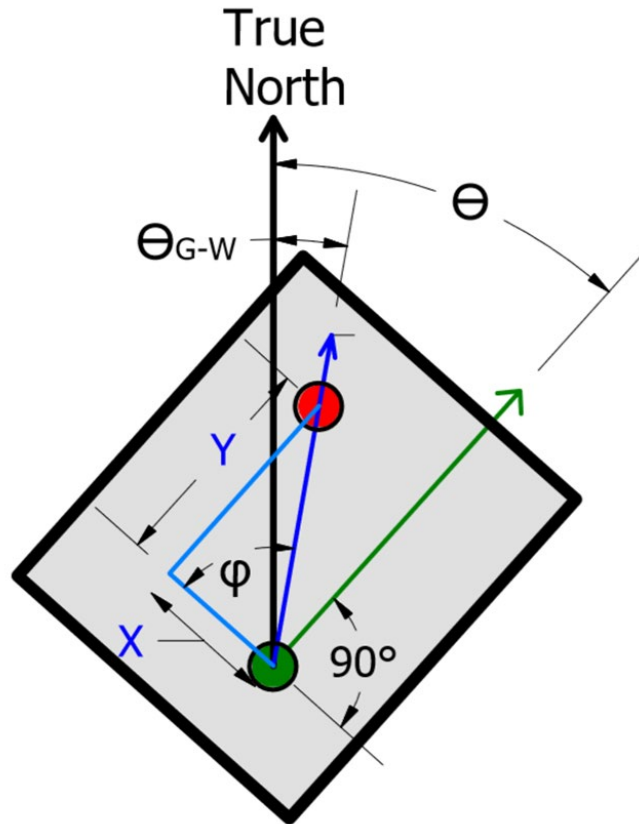


Figure 3.2: Diagram shows the definition of azimuth forward bearing θ , and bearing facing the weather station $\theta_{G \rightarrow W}$

Once the bearing facing the weather station was calculated, ("Calculate distance, bearing and more between Latitude/Longitude points,") the angular distance δ from the GNSS to weather station and was calculated, shown as equation 3.4. The new latitude and longitude coordinates are solved using equations 3.5 and 3.6 respectively.

$$\delta = \frac{\text{Offset Distance}}{r} \quad (3.4)$$

$$L_T = \sin^{-1}(\sin(L_i) * \cos(\delta) + \cos(L_i) * \sin(\delta) * \cos(\theta_{G \rightarrow w})) \quad (3.5)$$

$$K_T = K_i + \tan^{-1}(\sin(\theta_{G \rightarrow w}) * \sin(\delta) * \cos(L_i), \cos(\delta) - \sin(L_i) * \sin(L_T)) \quad (3.6)$$

The Haversine distance calculation is completed using equations 3.7, 3.8, and 3.9. The resulting velocity at the weather station's location is the Haversine distance divided by the sampling interval t (equation 3.10). Finally, the heading at the weather station's location can be calculated using the same method as at the GNSS location using equation 1.

$$a = \sin\left(\frac{L_i - L_{i-1}}{2}\right)^2 + \cos(L_{i-1}) * \cos(L_i) * \sin\left(\frac{K_i - K_{i-1}}{2}\right)^2 \quad (3.7)$$

$$c = 2 \tan^{-1}(\sqrt{a}, \sqrt{1 - a}) \quad (3.8)$$

$$d = r * c \quad (3.9)$$

$$v = \frac{d}{t} \quad (3.10)$$

Where:

δ = angular distance

Offset Distance = $\sqrt{((31.25 * .0254)^2 + (0.5 * 0.354)^2}$ meters from GNSS receiver

r = Radius of the Earth ($6361 * 10^3 m$)

L_T = Transformed Latitude Coordinate in decimal degrees

K_T = Transformed Longitude Coordinate in decimal degrees

L_i = Latitude coordinate at position i in decimal degrees

K_i = Longitude coordinate at position i in decimal degrees

d = Haversine distance in meter

v = Average velocity between coordinates

t = Sampling interval of GNSS (0.05 seconds)

Figure 3.3: Legend for equations 3.1-3.10

Although seemingly effective by comparison to other methods during preliminary testing, the resulting Haversine velocity calculations were noisy and yielded poor similarity as compared to other methods (Figure 3.4). This was because equations 3.1-3.10 assume a spherical Earth model. An alternative method of solving for coordinates at the weather station's location involved using Thaddeus Vincenty's formula for a WGS-84 geodetic ellipsoid (Vincenty, 1975). A modified form of Vincenty's formula was also used to solve for distance between transformed points. This algorithm is significantly longer but, claims higher accuracy up to a few millimeters.

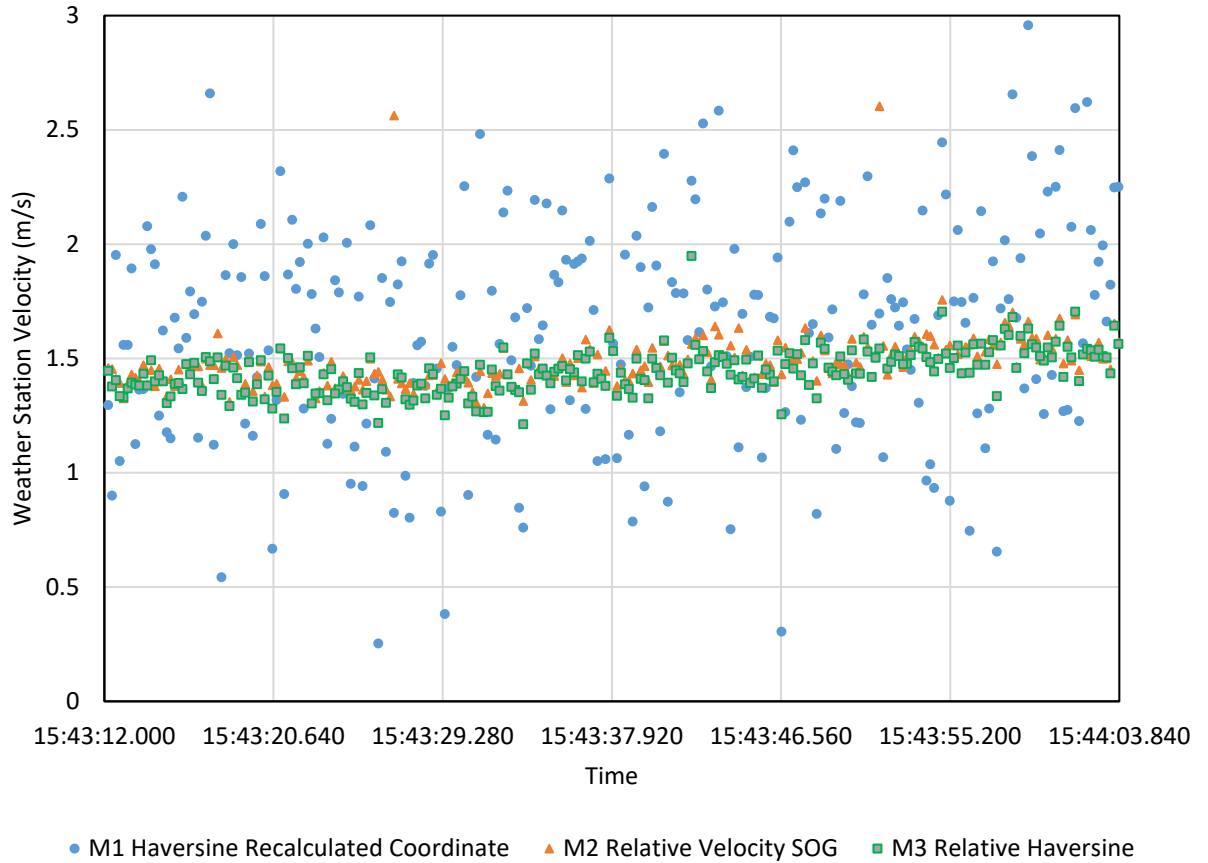


Figure 3.4: The Haversine calculated form of method 1 yielded poor similitude when using equations 4-6 for new coordinate calculation

The accuracy of this method of calculation is determinant on the accuracy of the GPS fix at the location of the GNSS receiver, which can vary greatly based on the current satellite constellation, location on Earth, interference, and differential correction method. For collected data in this study, the expected error of the fix is small because of the Real-Time Kinematic RTK functionality included in the Trimble R10 model GNSS, which received data from the Kentucky Transportation Cabinet virtual reference station (VRS). Also, the Haversine distance formula assumes a spherical model of the Earth, and a constant known Earth radius at all locations involved in the calculation, which has been

assumed equal to $6371 * 10^3$ meters. The Vincenty formula differs from this by assuming a WGS-84 oblate ellipsoid. In both formulas, the transform headings were calculated by averaging sequential GNSS courses so that the transform could be completed relative to the course at each coordinate rather than courses in route towards them. A sample plot of original GNSS coordinates and calculated coordinates at the weather station's location is shown as Figure 3.5.

Distances between the GNSS coordinates and the transformed coordinates using equations 1-6 (spherical Earth model) and the Vincenty method (WGS-84 Earth model) were calculated using Vincenty's distance formula (`vreckon.m` in Appendix 5) to compare accuracy between the methods of coordinate transform. The expected distance was the measured 0.7939 meters between the GNSS receiver and the 92000 model anemometer. The Vincenty WGS-84 method yielded better accuracy compared to the spherical method however, the spherical method of transform was still accurate within about ± 0.06 meters. The distances are both displayed as Figure 3.6. To reduce noise in the velocity calculations, filtering was applied to match the timespan of filtering for the anemometers. These results are displayed and compared to other methods in section 3.2.1.

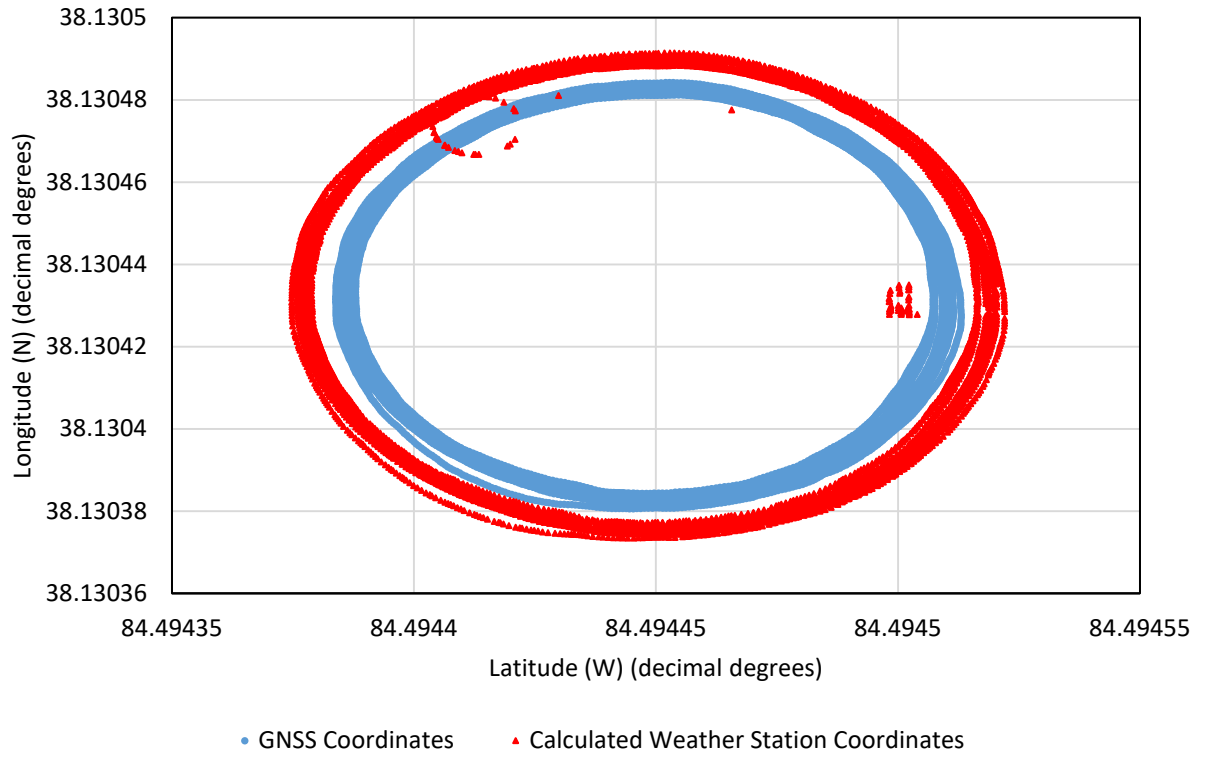


Figure 3.5: Weather stations coordinates solved using Vincenty's formula were effective at transforming at a set distance.

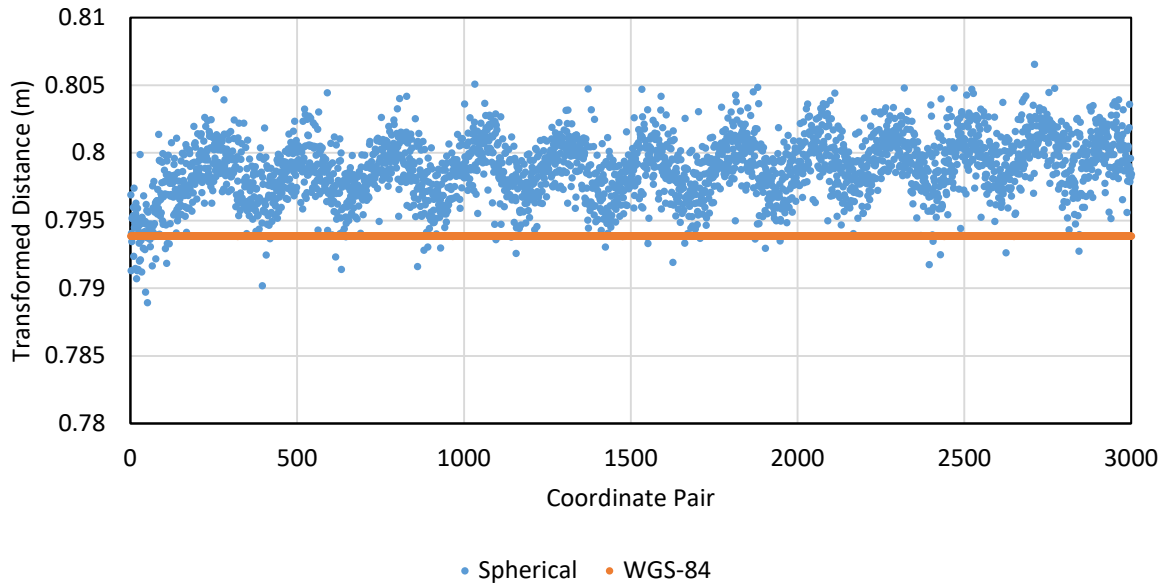


Figure 3.6 Distances between GNSS coordinates and transformed coordinates (coordinate pairs) show better accuracy using the WGS-84 Earth model

3.1.4 Method 2: Relative Velocity to GNSS

Higher-end GNSS receivers such as the Trimble RTK R10 may feature high precision velocity tracking that can be used in substitute to the successive coordinate method previously described. The featured GNSS receiver uses the Doppler effect to calculate velocity relative to satellite motion, measuring the difference in a message's frequency at emission and arrival from multiple satellites to solve for velocity. The principle of this calculation is explained in depth by an article from InsideGNSS (Gaglione, 2015). Similar to the successive coordinate method, the velocity at the location of the GNSS receiver is insufficient for large separation distances from the weather station, thus a transform algorithm is described, relating the velocities at each location by a scalar multiple equal to the ratio of the turn radii of both locations. This

method 2 and the subsequent section's method 3 were examined as a comparison to the calculated weather station dynamics solved using method 1.

Before implementing this method, a few assumptions need considered. First, the precision of the transformed velocity is dependent on the precision of the output GNSS velocity. The precision of the GNSS velocity is dependent on the number of available satellites involved in the Doppler effect calculations. Additionally, the principle of this method relies on an assumption that angular velocity of the vehicle and attached bodies is equal and acceleration is negligible between measurements. This is a reasonable assumption at high sampling rates where accelerative force does not significantly change velocity between sampled points.

3.1.5 *Calculating Velocity at Weather Station's Location relative to GNSS Location*

The velocity transform from the GNSS receiver location to the location of the weather station that was initially used was completed by computing a scalar multiple equal to the ratio of turn radii at both locations (GNSS and weather station) that can be applied to the output GNSS velocity. The principle of calculation is described in this section.

Equations 3.11 and 3.12 express the velocities at the GNSS and weather station locations as circular motion by relating them to an equal angular velocity ω multiplied by their respective turn radii. Combining these two equations yield equation 3.13 for the velocity at the location of the weather station. Figure 3.7 visualizes the geometry for equations 3.14 and 3.15 solving for the turn radius at the weather station for right-hand

and left-hand turns respectively. Once this relation is derived, the turn radii of the GNSS needs to be calculated.

$$V_{GNSS} = \omega * r_{GNSS} \quad (3.11)$$

$$V_{Weather} = \omega * r_{Weather} \quad (3.12)$$

$$V_{Weather} = V_{GNSS} * \left(\frac{r_{Weather}}{r_{GNSS}} \right) \quad (3.13)$$

$$r_{Weather} = \sqrt{(r_{GNSS} + X)^2 + Y^2} \quad (3.14)$$

$$r_{Weather} = \sqrt{(r_{GNSS} - X)^2 + Y^2} \quad (3.15)$$

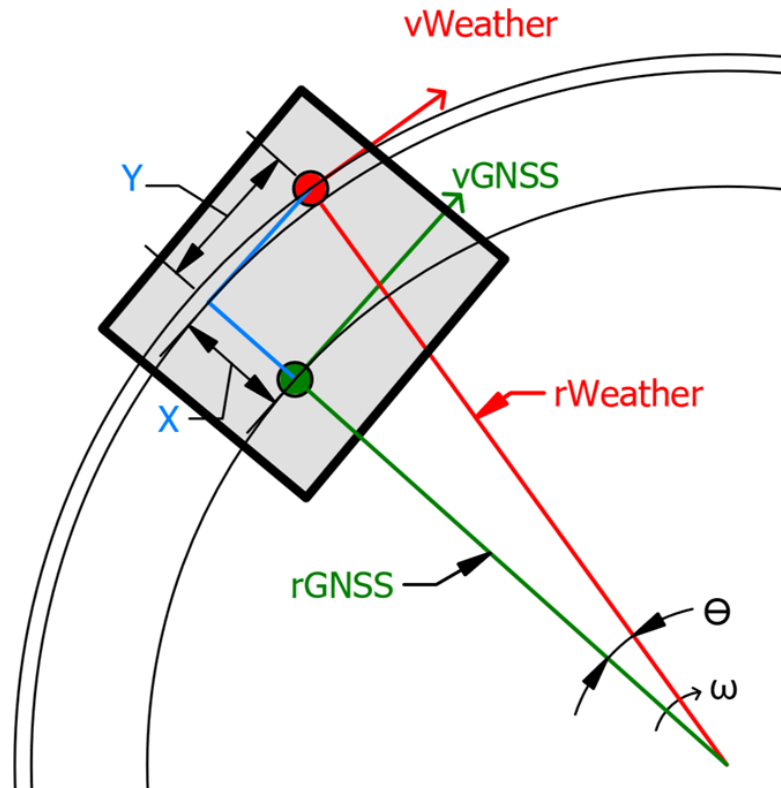


Figure 3.7: Diagram displaying turn radii and velocity vectors at the GNSS and weather station locations. The direction of the velocity vectors are pointing to the rear of the vehicle for visibility.

To solve for the turn radius of the GNSS receiver, two velocity measurements were used in an attempt to solve for GNSS turn radii without physical measurements. The theoretical distance traveled between data samples was solved by averaging successive velocities multiplied by the sampling time interval (equation 3.16). Equation 3.17 expresses the distance traveled as the arc length of the path traveled. By using these two relations, the turn radius r_{GNSS} was solved using equation 3.18 and was used to solve for the turn radii at the weather station location. With all required parameters solved, the velocity at the weather station was attempted. A legend for the terms of equations 3.16-

3.18 is displayed as Figure 3.9, and a visualization of variables as Figure 3.8. This method was unsuccessful however, because the calculated turn radii is better described as the radius of curvature. The offset distance between the GNSS receiver positions was consistently much larger than the calculated radius, which could not be scaled to the actual turn radius of the receiver for transformation to the weather station's position. Instead, the known turn radii defined by the experiment's plan was used. This is described further in 3.1.7.

$$distance = V_{GNSS} * \Delta t \quad (3.16)$$

$$distance = 2\pi r_{GNSS} * \frac{d\theta}{360} \quad (3.17)$$

$$r_{GNSS} = \frac{V_{GNSS} * t * 360}{2\pi * d\theta} \quad (3.18)$$

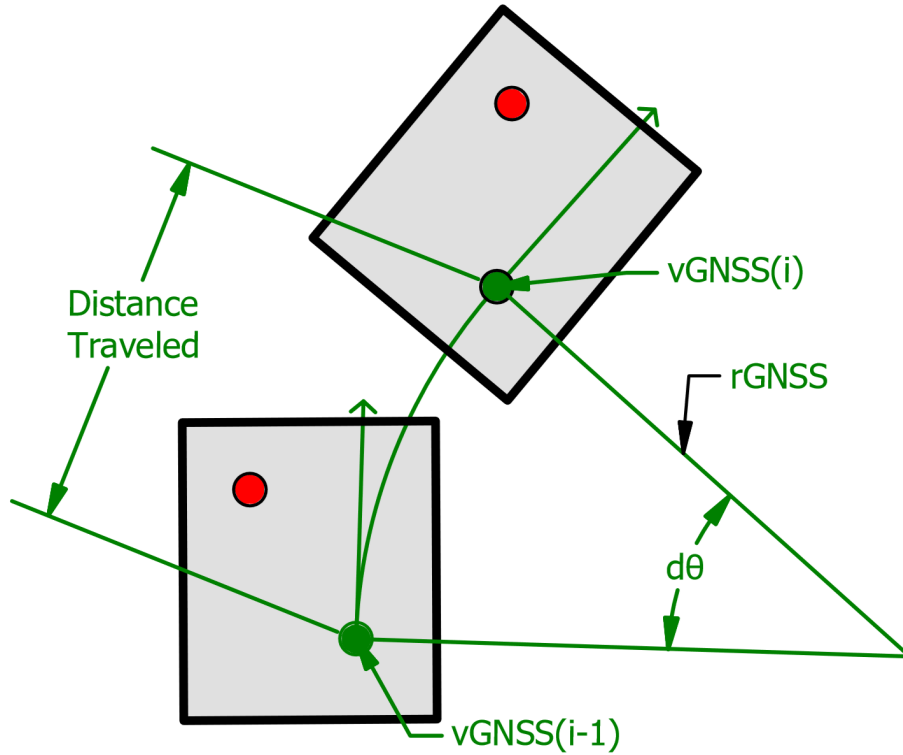


Figure 3.8: Turn radius visualization at the GNSS location requires two velocity measurements. Velocity vectors are pointing in the opposite direction for improved visibility.

Where:

r_{GNSS} = Turn radius of GNSS receiver

$r_{Weather}$ = Turn radius of weather station

X = Sideways offset

Y = Forward offset

V_{GNSS} = Recorded velocity of GNSS receiver

$V_{Weather}$ = Calculated velocity of weather station

Figure 3.9: Legend for parameters in equations 16-18

Finally, to obtain the heading of the weather station, first θ in Figure 3.7 can be calculated using equation 3.19, and can be added to the GNSS “Course Over Ground” output for left turns and subtracted for right turns.

$$\theta = \sin^{-1}\left(\frac{Y}{r_{Weather}}\right) \quad (3.19)$$

3.1.6 *Method 3: Calculating Haversine velocity at GNSS location and using the relative velocity method to transform velocity to the weather station’s location*

Alternative to the previous methods, a third method of calculating GNSS velocity using successive coordinates and transforming them to the weather station’s location using the relative velocity method was attempted. This could be a more elegant option for using successive coordinates if using a GNSS receiver that does not output velocity, because it yields a shorter algorithm that likely would solve faster compared to Method 1. This is because it eliminates the need to calculate azimuth forward bearing and coordinates at the location of the weather station. First, Vincenty or Haversine distance and velocity can be calculated, and then transformation completed using the relative velocity method.

3.1.7 *Adjustments and Limitations for Methods for A Circular Path*

After collecting pilot GPS data to test vehicle velocity calculations, all methods appeared to work well when driving along a straight path, but later testing revealed flaws in the algorithm on a circular path. For method 1, the issue was identified by incorrectly

transformed coordinates caused by misalignment of the bearings in time with GPS coordinate data. Initially the transform angle towards the weather station was calculated locally as a constant (90 degrees added to the counter-clockwise form of equation 3), which was relative to the front of the vehicle however, the course angles used were the average course between points rather than at each coordinate. The course before and after each location were averaged to represent the angle at the point. After calculating the weather station's velocity, filtering was conducted over the timespan of anemometer data filtering.

For methods 2 and 3, flaws were discovered in the underlying equations for calculating the scalar multiple for velocity transform. In equations 3.14 and 3.15, by adding the physical offset dimension to the radius of curvature, the scalar was incorrectly calculated. The mistake isn't obvious for straight paths because the radius of curvature approaches infinity, so small physical offsets make a negligible difference in the calculation. To correct this, the physical offset between the weather station and GNSS receiver needed to be scaled to match the radius of curvature by dividing by the offset by the actual turn radius, to add to the GNSS radius of curvature (equation 3.20). Alternatively, simply calculating the ratio of physical turn radii as the scalar multiple has similar results and has the advantage of not needing the previous heading to calculate GNSS turn radii. Applying this adjustment to these methods automatically limit their application because the physical turn radius must be estimated. In this study, trials were completed at known separation distance however, were not precise for the duration of each trial because of inevitable human errors while driving. Implementing these methods

in an actual system would require knowing the turn radius and may be better suited for center-pivot irrigation systems than self-propelled sprayers.

$$r_{Weather} = r_{GNSS} + \frac{\text{offset} + \text{separation}}{\text{separation}} \quad (3.20)$$

3.1.8 *Comparing Methods of Anemometer Velocity*

The methods of velocity calculations are compared. Calculation time from data collection through the calculation of the weather station's velocity were an important consideration for processing the observed wind data. Here a discussion of the reaction time is introduced as the time period in which data processing and nozzle orifice actuation can be completed. This work focuses on the processing portion of this reaction time and involves the time to collect and filter weather data, collect GNSS data, calculate vehicle dynamics, and back out those vehicle dynamics from apparent wind velocities. Since the three described methods of velocity calculation have different inputs and principles of calculation, the following should be considered.

Flow diagrams displaying the number of steps for each of the three methods are displayed as Figure 3.10, Figure 3.11, and Figure 3.12. Transforming position coordinates to different coordinates were much more time consuming than velocity calculations. All methods of calculating weather station velocity from GNSS measurements require at least two data points. Using the successive coordinate method 1 requires three position coordinates for the Vincenty formulas. Two course bearings are obtained from the 3 positions and are averaged to get the instantaneous bearing at the

middle coordinate. If a GNSS receiver that calculates velocity and “course over ground” from the doppler effect is used, then only two position coordinates are needed (with matching output bearings) to solve for velocity using Vincenty’s formula. Alternatively, the relative velocity methods 2 and 3 using this output data require two GNSS velocity measurements. As mentioned in the previous section, the latter two methods may alternatively be calculated with one GNSS velocity measurement but still depend on a known or estimated instantaneous turn radius with good precision. A disadvantage to calculating velocity using successive GNSS coordinates (method 1) is an increased calculation time, as GNSS velocity must be calculated after data collection, whereas using a GNSS receiver utilizing doppler effect calculations (methods 2 and 3) provides this information as an output. This is easily visible by the difference in calculation lengths between the methods, where the successive coordinate method 1 requires several computationally time expensive trigonometric functions for both Vincenty and Haversine calculations. Method 3 requires the Haversine calculation, while method 2 only requires 1 trig function for the heading. Depending on the GNSS sampling interval, method 2 could have a significant advantage in potential reaction time however its effectiveness and that of method 3 effectiveness is again limited by the precision of the input turn radii.

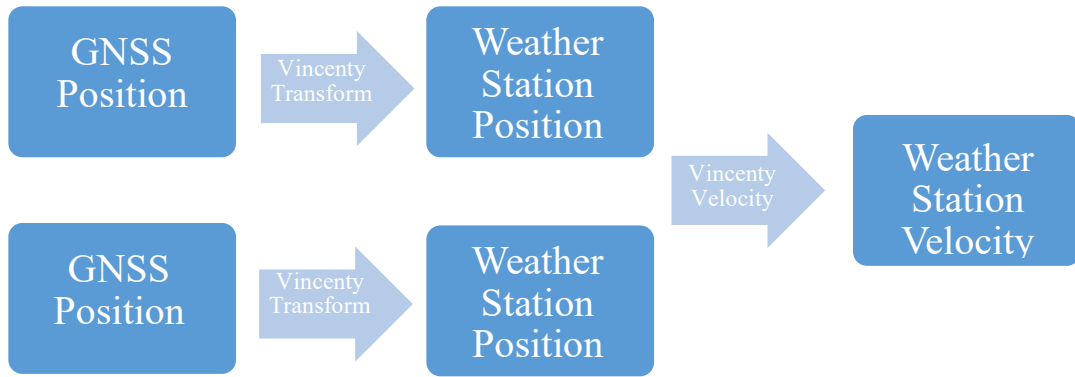


Figure 3.10: Flow chart visualizing Method 1 weather station velocity calculation



Figure 3.11: Flow chart visualizing Method 2 weather station velocity calculation for known turn radii

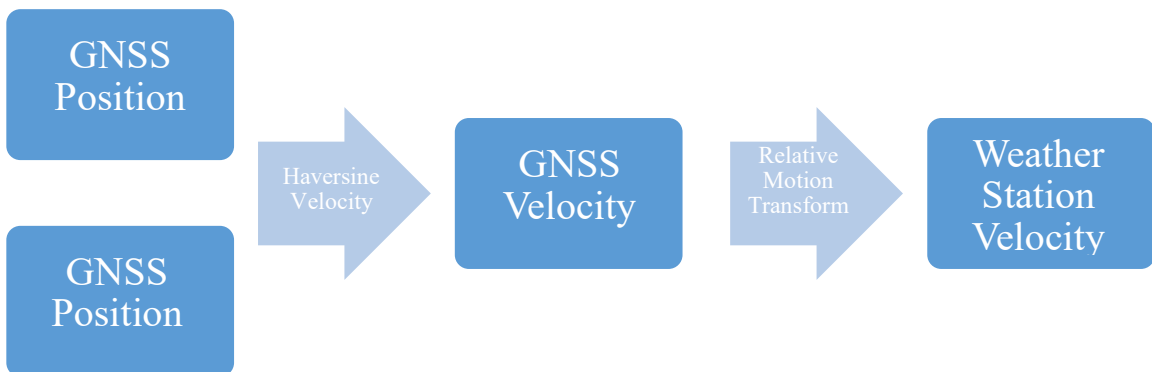


Figure 3.12: Flow chart visualizing method 3 weather station velocity calculation

Considering the weather station's temporal sampling interval of 200 ms, and the GNSS receiver sampling every 50 ms, whenever GNSS measured or sampled velocities did not line up with weather data, all three methods could yield either three or four velocity calculations between weather data samples. With perfect alignment, up to five GPS coordinates can be recorded over 200 ms. The vehicle velocity data solved using method 1 was filtered using a moving average matching the sampling interval/filtering window of the weather station to reduce noise, simulating the amount of data available in real-time. For example, if the weather data were to be filtered with a 3-point window (600ms), then the GNSS velocity was filtered using a 12-point window (600ms). These velocity calculations were linearly interpolated to the time interval of the weather data, so a correction of weather data by subtracting vehicle dynamics can be achieved. Since interpolation accuracy is dependent on the sampling interval, the correction of wind data will be as well.

3.1.9 *Removing Vehicle Velocity from Apparent Wind Data*

Once the determined vehicle dynamics have been linearly interpolated to the time interval of collected weather data, a correction of the observed weather station can be applied. First, wind data collected as polar vectors needed transformed into component values before the correction could be applied. Alternatively the anemometer may be configured to output wind vectors in polar format. Conditional statements for decomposing the 360-degree compass coordinates were used before applying trigonometric functions to calculate wind velocity components. Once wind vectors were decomposed into "X" and "Y" components, the interpolated dynamics at the weather

station's location were subtracted from the forward "Y" component of the wind data as shown in Figure 3.13. Again, a conditional statement was required to handle negative Y component values corresponding to gusts blowing from the front end of the vehicle. Next, the magnitude of the wind velocity was solved using equation 3.21, and the direction solved using conditional statements in equations 3.22-3.25, and visualized in Figure 3.14.

$$|V_{wind\ corrected}| = \sqrt{(Y')^2 + (X)^2} \quad (3.21)$$

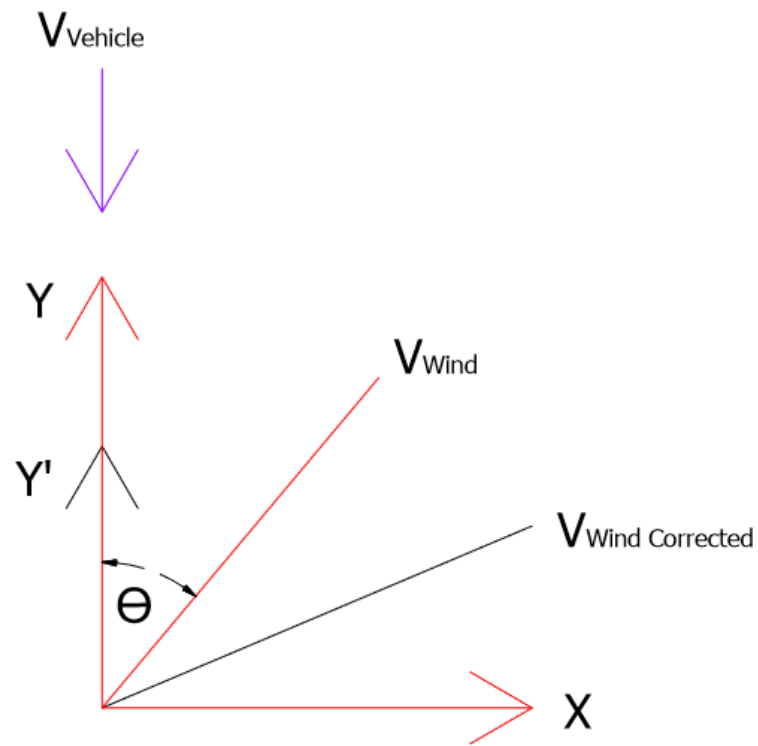
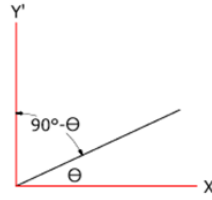
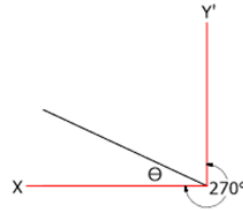


Figure 3.13: A diagram visualizing wind components and resultant vectors after subtraction of vehicle velocity from the "Y" component. $V_{\text{Wind Corrected}}$ is the resultant vector of Y' and X components.

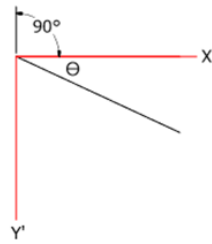
a.)



b.)



c.)



d.)

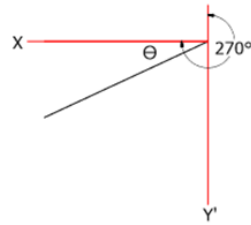


Figure 3.14: Visualization of wind direction calculation according to equations 21-24

a.) If $Y' < 0$ and $X < 0$

$$\theta_{Wind\ Corrected} = 90^\circ - \left| \tan^{-1} \left(\frac{Y'}{X} \right) \right| \quad (3.22)$$

b.) If $Y' < 0$ and $X > 0$

$$\theta_{Wind\ Corrected} = 270^\circ + \left| \tan^{-1} \left(\frac{Y'}{X} \right) \right| \quad (3.23)$$

c.) If $Y' > 0$ and $X < 0$

$$\theta_{Wind\ Corrected} = 90^\circ + \left| \tan^{-1} \left(\frac{Y'}{X} \right) \right| \quad (3.24)$$

d.) If $Y' > 0$ and $X > 0$

$$\theta_{Wind\ Corrected} = 270^\circ - \left| \tan^{-1} \left(\frac{Y'}{X} \right) \right| \quad (3.25)$$

Once vehicle dynamics are effectively removed from the apparent wind data, a final correction rotating the wind vectors relative to True North is necessary for correlation testing with a static anemometer. In Figure 3.15, “False North” refers to the heading of the weather station, while $|V_{Wind}|$ is the corrected wind velocity after removing vehicle dynamics, and $\theta_{Wind\ Corrected}$ is the recalculated direction of wind gusts relative to “False North” or the weather station’s heading. To obtain the correct heading of wind data, the apparent wind heading is simply added to its relative vehicle heading as shown in equation 3.26. A modulus function is used to handle resulting headings greater than $359.\bar{9}$ to keep results in the range of compass bearings.

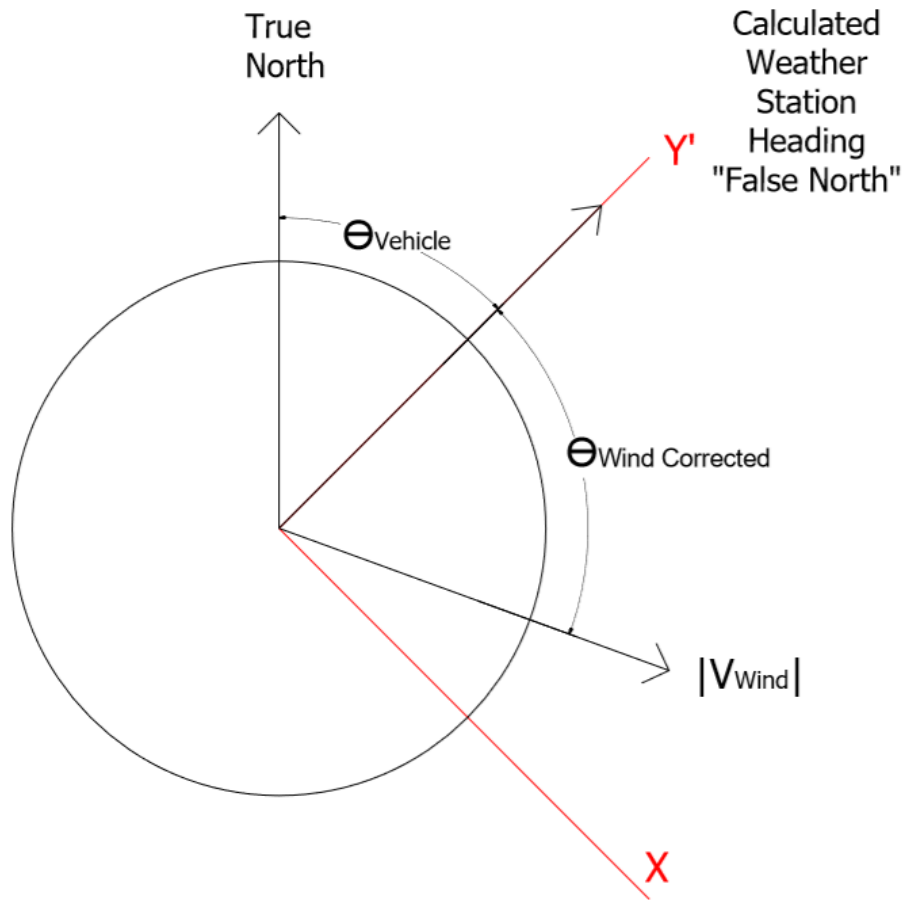


Figure 3.15: Diagram displays principle of correction of wind directions to be relative to True North

$$\theta_{True} = \theta_{Vehicle} + \theta_{Wind\ Corrected} \quad (3.26)$$

3.1.10 *Quantifying Similitude Between Anemometer and Dynamic Weather Station*

The analysis of output data for this experiment was conducted similarly as the stationary experiment in Chapter 2. Identical to the analysis of the stationary experiment, the Pearson correlation coefficient was determined as an unsuitable indicator of

similitude between the two data series at the distances between sensors because it tests for linear dependence. Cross-correlograms were generated and examined at zero lags, and their shapes were examined for structured decay at increasing lags as indicators of variability. An analysis of differenced wind velocities was also studied to indicate the percentage of differences under increasing thresholds. Filtering uncertainty described in section 2.1.8 was calculated but used only for visualization purposes.

3.2 Results:

3.2.1 *Vehicle Velocity and Course Comparisons*

In this section the results for the various methods of vehicle velocity calculations are discussed. Among the three methods of calculation for the velocity of the weather station, the results are similar. The targeted vehicle velocity (3mph, 6mph, 9mph) seemed to have had little effect on the accuracy of any method however, 3 mph trials seemed to produce the least noise. Figure 3.16, Figure 3.17, and Figure 3.18 display weather station velocity calculations using the three methods at a 6.10 m separation distance. In some trials an oscillating trend exists, caused by the driver pushing the gas pedal to accelerate and letting off to slow to the trial's target velocity. The three methods were similar to each other however, method 1 had the most noise. Since this method relied on course calculations to determine the heading where new coordinates were to be projected to, inconsistencies in the spacing of these coordinates caused more variation in the velocity calculation. Since coordinates were projected outward from a circular path with the course changing rapidly, coordinate spacing was larger as expected, and often smaller where the vehicle's course changed more gradually. It also may have been caused by imprecision with the adjustment method discussed in section 3.1.7. Although the

transformed points had a somewhat sporadic spacing, they remained the correct distance from the GNSS receiver's location. Figure 3.19 shows how applying filters to the velocity data over the same time span of wind data helped to alleviate these inconsistencies.

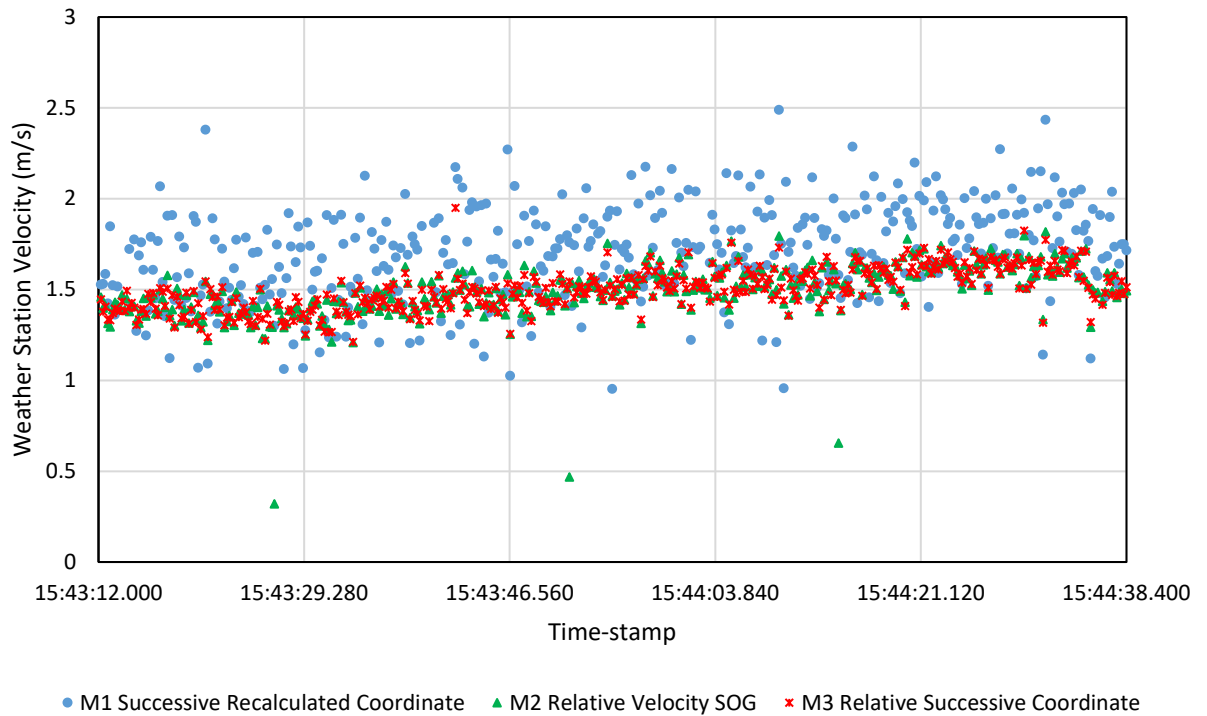


Figure 3.16: Three methods of calculation for weather station velocity yield close results for dynamic testing at 20 ft anemometer separation, approximately 3 mph vehicle speed, trial 1. Method 1 (M1) results were noisier when unfiltered.

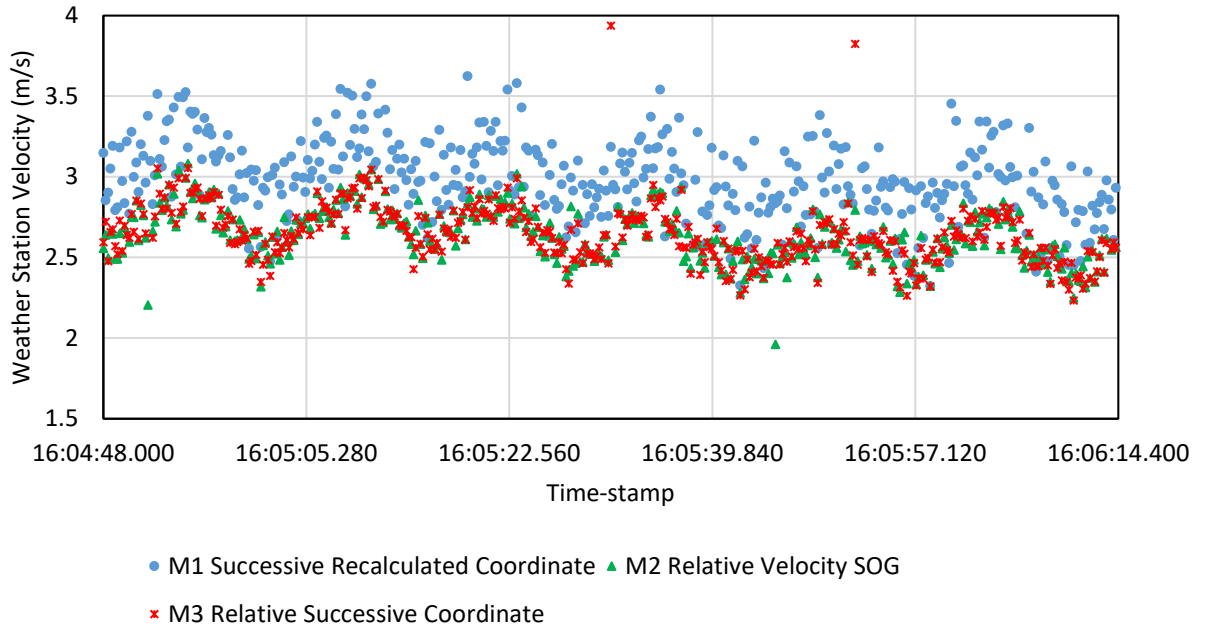


Figure 3.17: Three methods of calculation for weather station velocity yield close results for dynamic testing at 20 ft anemometer separation, approximately 6 mph vehicle speed, trial 1.

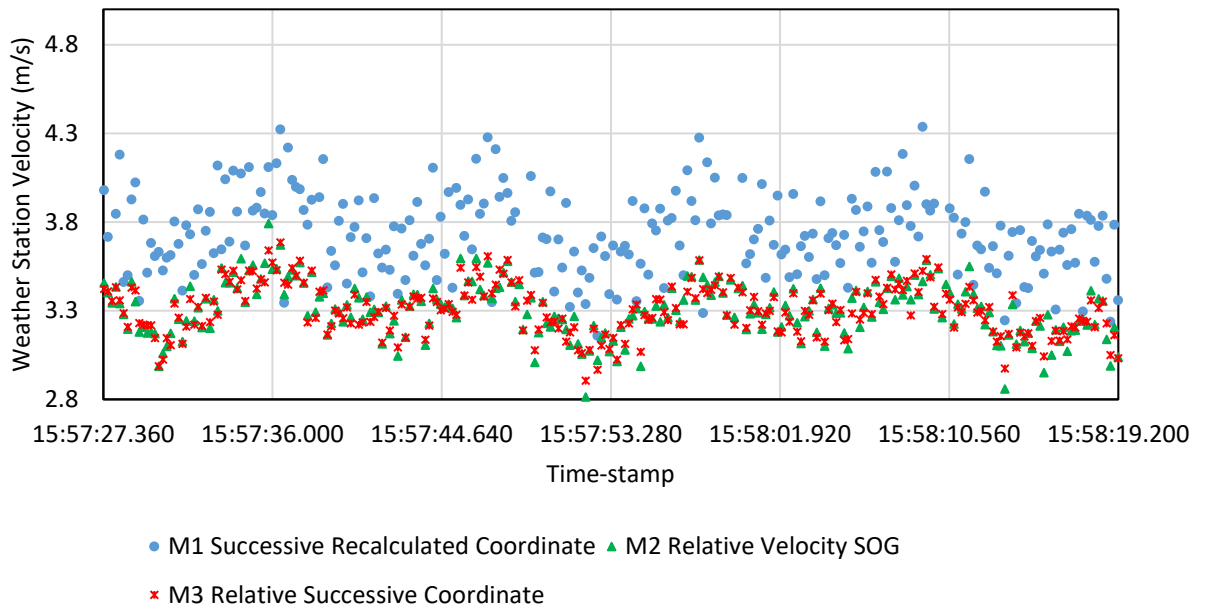
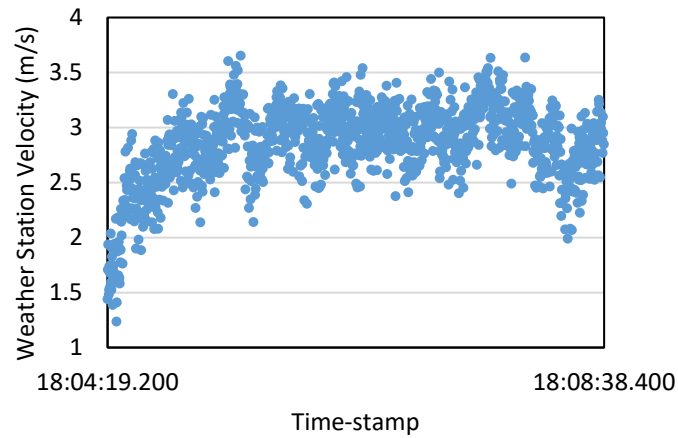
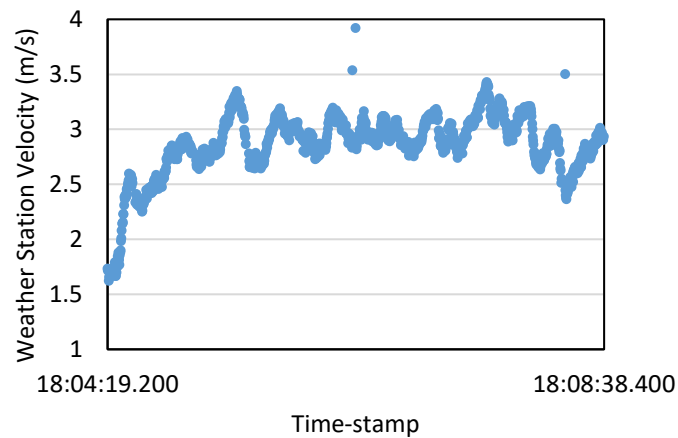


Figure 3.18: Three methods of calculation for weather station velocity yield close results for dynamic testing at 20 ft anemometer separation, approximately 9 mph vehicle speed, trial 1.



• M1 Successive Recalculated Coordinate



• M1 Successive Recalculated Coordinate

Figure 3.19: The data for block 3, at a 40 ft separation, 6 mph, no filter (top) and 7-point moving average (bottom) show filtering’s effect of smoothing weather station dynamics on method 1 for reducing noise before backing it out from wind data.

For the weather station’s course calculations, all three methods yield the same results. Matching course calculations for the velocities in Figure 3.16, Figure 3.17, and Figure 3.18 are plotted in Figure 3.20, Figure 3.21, and Figure 3.22 respectively. As

expected, the slope of the weather station's (vehicle course) plot increases with increasing velocity because it travels around the circular path faster.

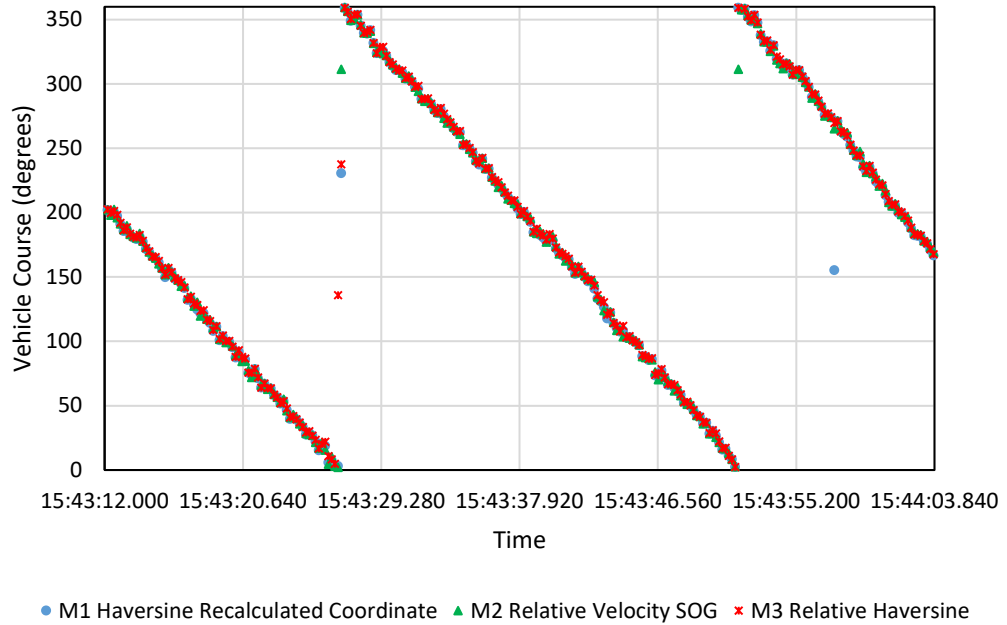


Figure 3.20: Course calculations for all three methods have good alignment with minimal noise. This sampled data is from Block 1, with an anemometer separation distance of 20 ft and 3 mph vehicle velocity target.

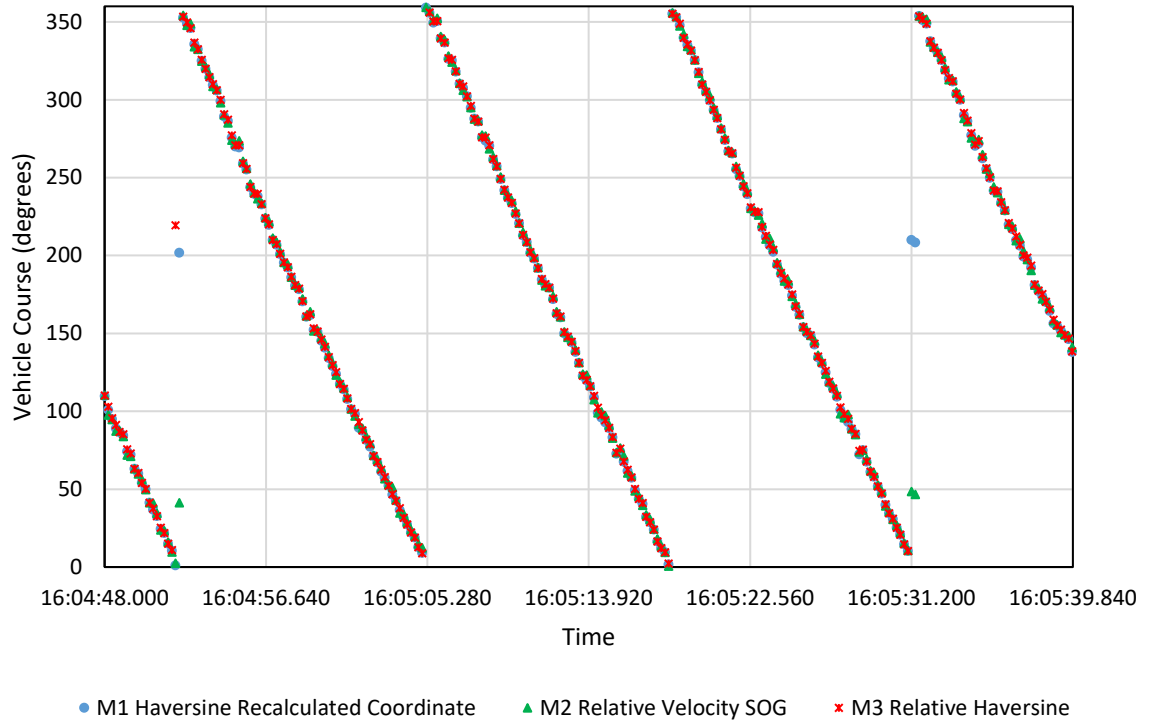


Figure 3.21: Course calculations for all three methods from sampled data from Block 1, with an anemometer separation distance of 20 ft and 6 mph vehicle velocity target.

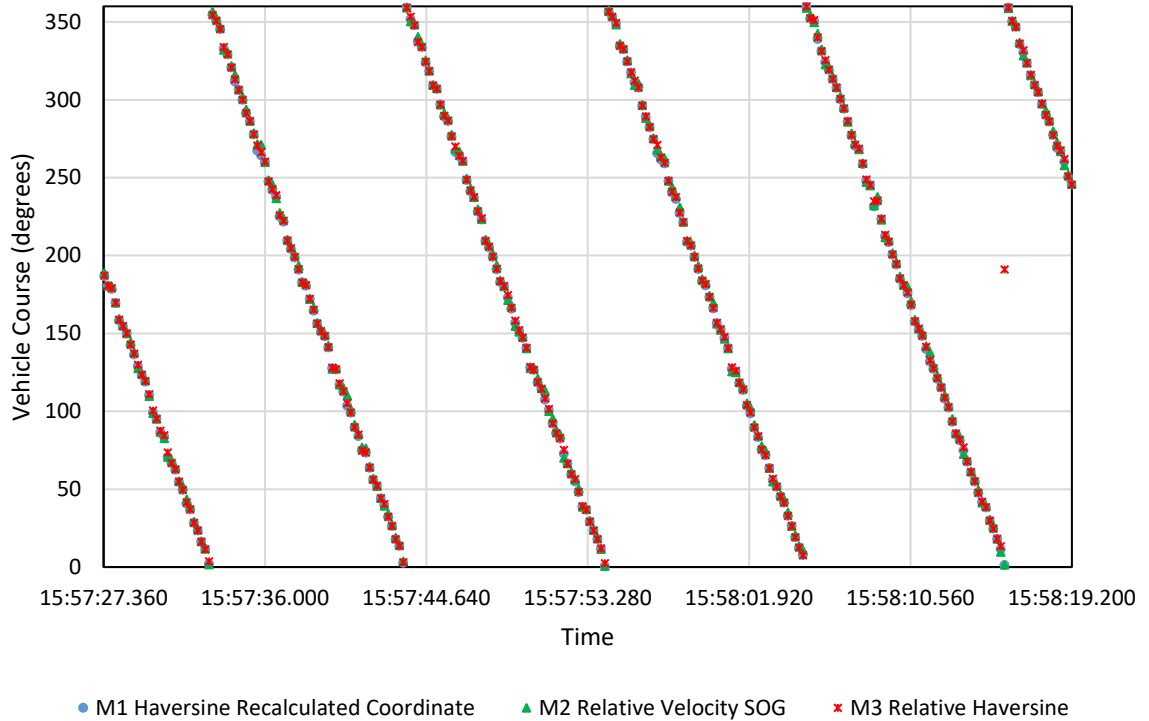


Figure 3.22: Course calculations for all three methods from sampled data from Block 1, with an anemometer separation distance of 20 ft and 9 mph vehicle velocity target.

3.2.2 Visualizing Filter Uncertainty

Filtering's effect on the dynamic weather station was first examined by plotting the data with calculated standard deviation for each data point over its filtering window. This standard deviation was considered the filtering uncertainty because the variance is descriptive of the range of wind velocities used for the calculation of each filtered point. These uncertainties were shown graphically as error bars for each resulting data point post-filtering. Figure 3.23 is the plotted filtered wind velocities and uncertainties for the stationary and dynamic sensors. Since the dynamic wind data is the result of subtracting vehicle dynamics, the uncertainty for each of its data points include both filtering

window's variance and uncertainty regarding the precision of the vehicle velocity calculations. Regardless, the initial plotting of the data appeared to show good similitude between the two series, especially where uncertainties overlap.

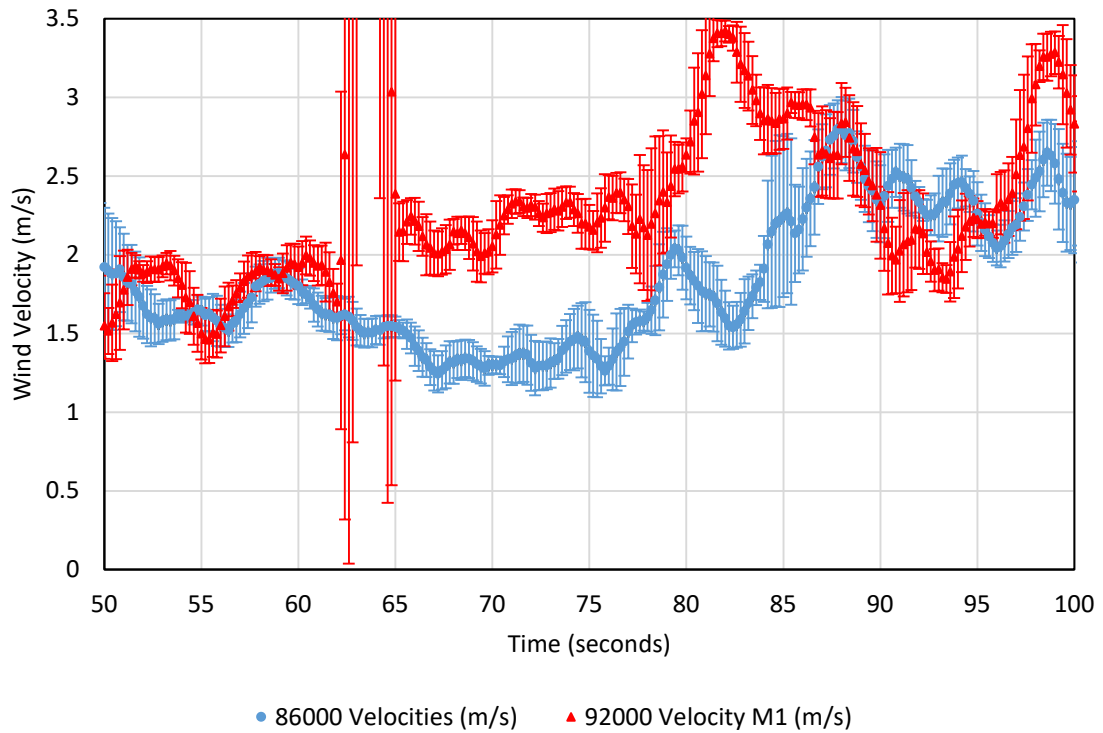


Figure 3.23 Scatter plot with filter uncertainties defined as the standard deviation over the filtering window.

3.2.3 *Validating Wind Gusts by Comparing Direction*

The directional data of raw stationary measurements and processed dynamic measurements were compared to validate the origin of wind gusts. If processed gusts were found to have similar velocities and direction, then confidence was improved in the processing algorithm's accuracy. Overall, matching wind directions were observed for both anemometers. Noise identified in the weather station's velocity calculations carried over to the processed wind speeds and direction as shown in Figure 3.24. The increasing filtering window's effects on noise reduction on the sensors dynamics also carried through to the wind dynamics as well and is visualized in Figure 3.25. Disregarding noise, the directional data appeared to fluctuate in sync between the two series, suggesting that the processing method for reorienting dynamic sensor data was effective. Increasing distance between the sensors and higher vehicle velocities seemed to result in greater differences between the wind data. This observation was validated by comparison to the wind velocity data.

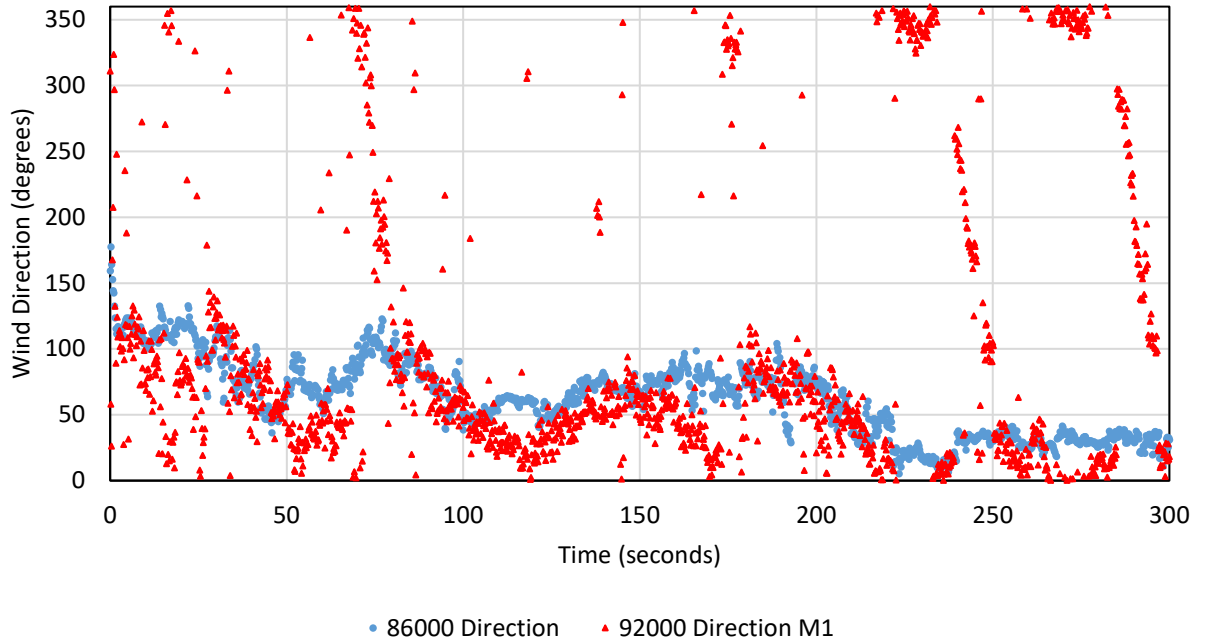


Figure 3.24: Noise from calculating vehicle dynamics carry into wind calculations when unfiltered. This data is from block 1, 40ft separation, 3 mph speed.

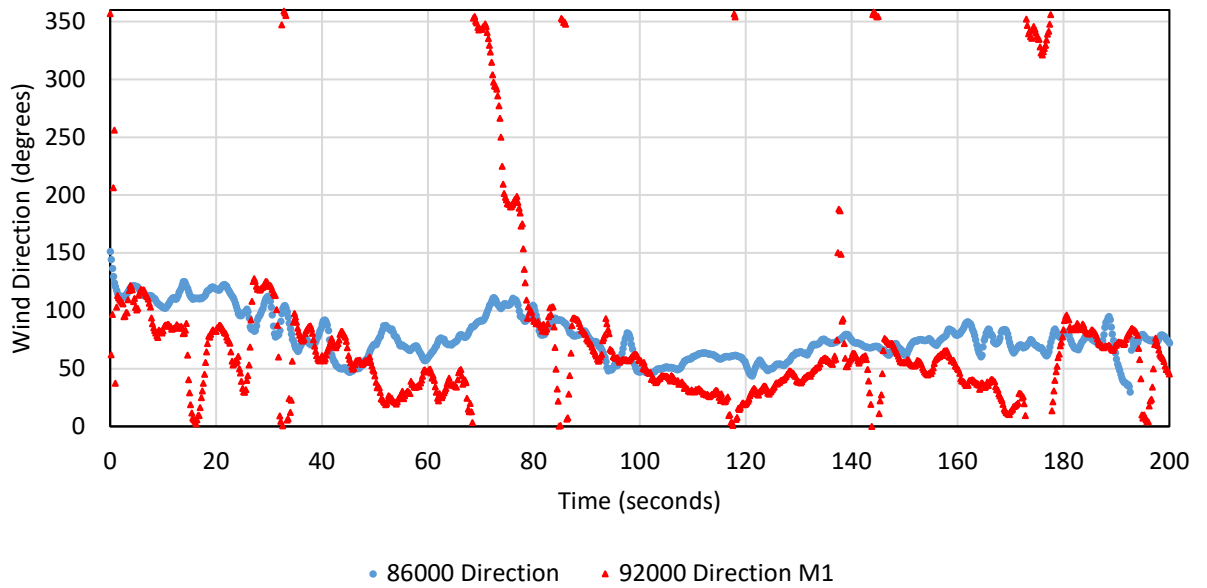


Figure 3.25: Noise in wind data were reduced using moving average filtering. This dataset was from block 1, 40 ft separation, 3 mph, with a 7-point moving average applied.

3.2.4 Cross-correlation Results

Cross-correlation testing was completed on raw data for each trial and for each selected filter (3, 5, and 7 point moving averages) for each method of weather station velocity calculation. In each dataset, the calculated cross-correlation coefficient as well as its decay over lag times was examined. As with stationary testing, a structured decay indicated similitude in time-series data because it suggests that paired wind velocities (at the same time) may experience the same gusts, as wind velocity at both locations fluctuate together.

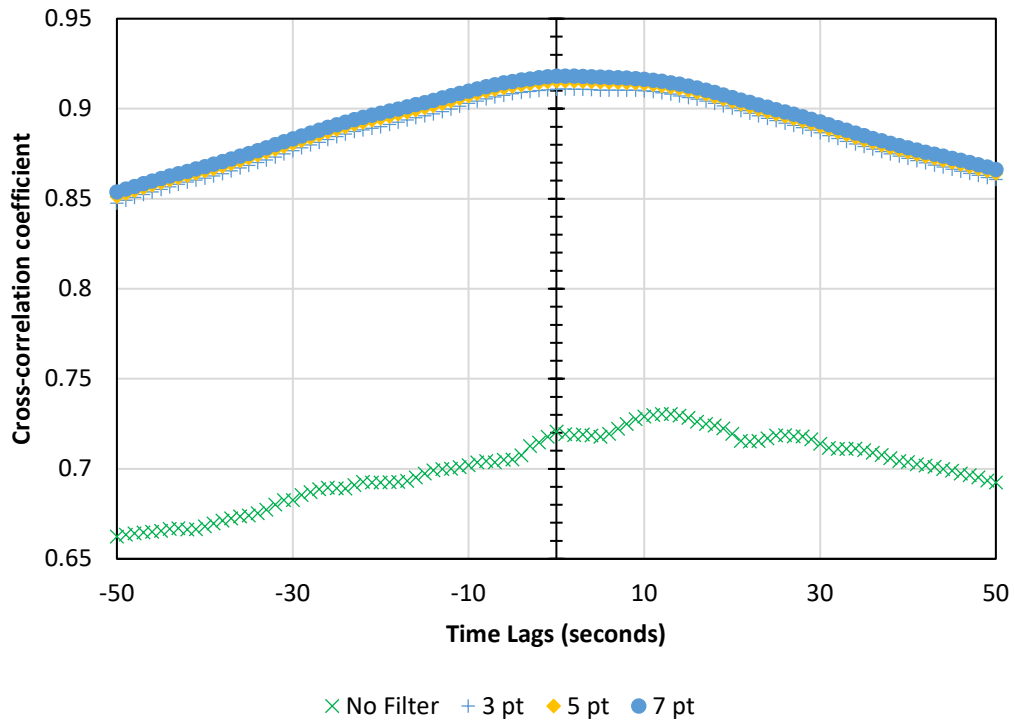
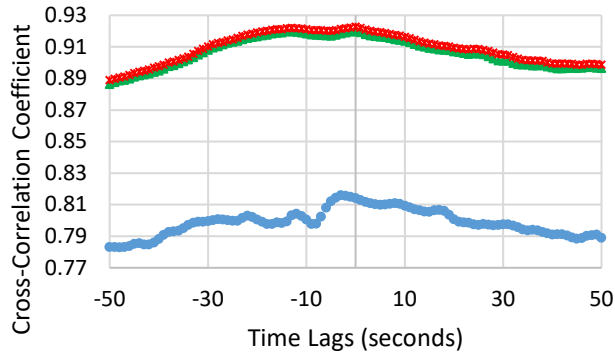
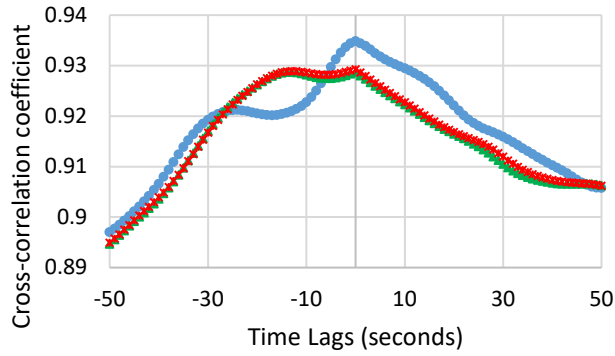


Figure 3.26: Increasing filtering window size improves correlation Block 1, 60 ft, 9 mph



- M1 Successive Coordinate
- ▲ M2 Relative Velocity
- × M3 Relative Successive Coordinate



- M1 Successive Coordinate
- ▲ M2 Relative Velocity
- × M3 Relative Successive Coordinate

Figure 3.27: Increasing filter size results in method 1 having the highest cross-correlation and similarity. Data from block 2, 20 ft separation distance, 3 mph is shown with no filtering (a. top) and 7-point moving average (b. bottom)

Similar to stationary experimentation, filtering using a moving average removed noise and yielded improved cross-correlation. Method 1 of calculating coordinates at the location of the weather station and then subsequently calculating the velocity yielded the lowest cross-correlation coefficient for all trials when left unfiltered. An example cross-

correlogram is displayed as Figure 3.26. This was caused by an abundance of noise in the calculation of the weather station's dynamics. When filtering was applied, method 1 sometimes yielded the highest cross-correlation, as shown in Figure 3.27 where method 1 has drastic improvement in correlation after filtering. In all trials the unfiltered data had very poor cross-correlation and improved significantly with filtering as the vehicle dynamics were smoothed. Methods 2 and 3 yielded inconsistencies in correlation, and it is unclear which of the two methods are more ideal from correlograms alone. At a 20 ft separation between the anemometer and weather station, cross-correlograms generally had structured decay across increasing lag times, shown in Figure 3.28a. Figure 3.28b is an example of structured decay over shorter lag distance (about 30 lags or 6 seconds) in which the increase in cross-correlation at longer times indicate a cyclical trend over the domain. These occurrences don't necessarily indicate any lack of similarity but instead may suggest higher variability over the test's duration.

With increasing separation distances (40 ft and 60 ft), the cross-correlograms generally become less structured. Figure 3.29 is an example of a 40 ft trial with good structure and steep decay. Table 3.2 shows indication of each cross-correlogram structure for all dynamic trials. "YES" indicates a symmetrical decay in cross-correlation over positive and negative lags, while "NO" indicates a flat structure. On some trials, structured decay occurred mainly across positive (Right side) or negative (Left side) lags. For most trials, an applied 3-point moving average filter significantly improved structure of the correlograms because of reductions in vehicle dynamic noise, while 5 and 7-point filters mainly increased the correlation while maintaining structure. In Table 3.2, indicators summarize the results for all filters tested.

Table 3.2: A summary of cross-correlogram structure for unfiltered dynamic trials. “YES” indicates a symmetrical decay in cross-correlation coefficient, “NO” indicates no structured decay. Descriptive notes are included for special cases.

Trial	3 mph	6 mph	9 mph
20 ft_1	YES	YES	Weak
20 ft_2	YES	YES	N/A
20 ft_3	YES	YES	N/A
40 ft_1	Weak	Weak	YES
40 ft_2	YES	NO	Yes, Left
40 ft_3	YES	YES	YES
60 ft_1	YES, Left side	YES, Left until Lag = 10	YES
60 ft_2	YES	YES	YES, until lag = 20
60ft_3	YES	YES, Left Side	Weak

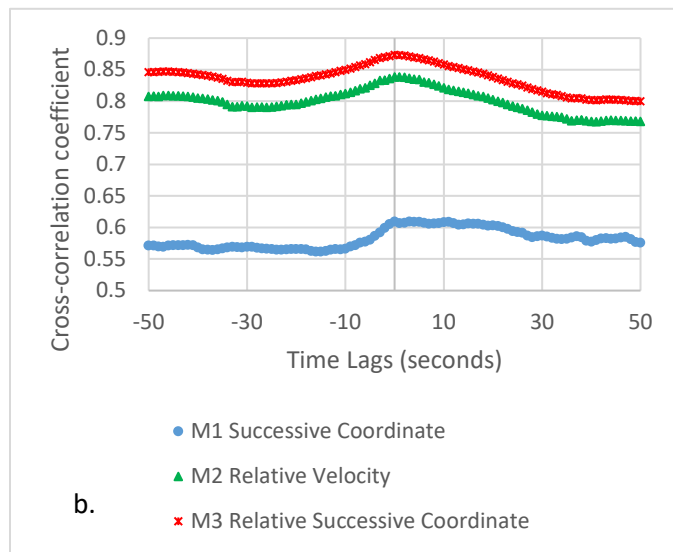
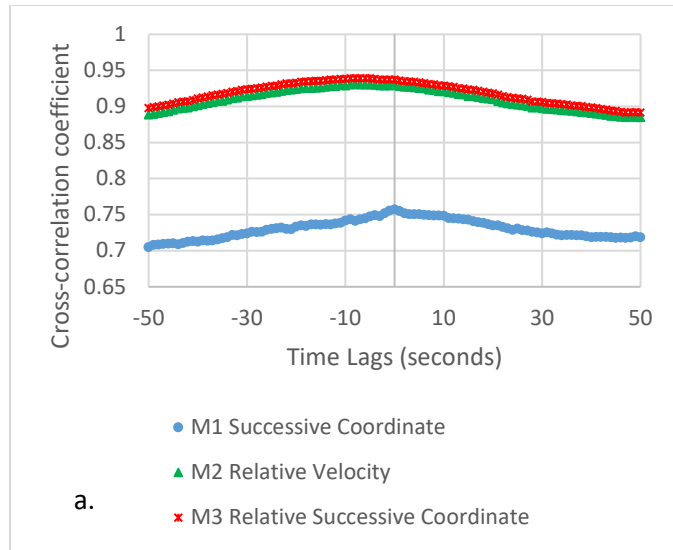


Figure 3.28: Block 1, 20 ft, 3 mph (a. top), 6 mph (b. bottom)

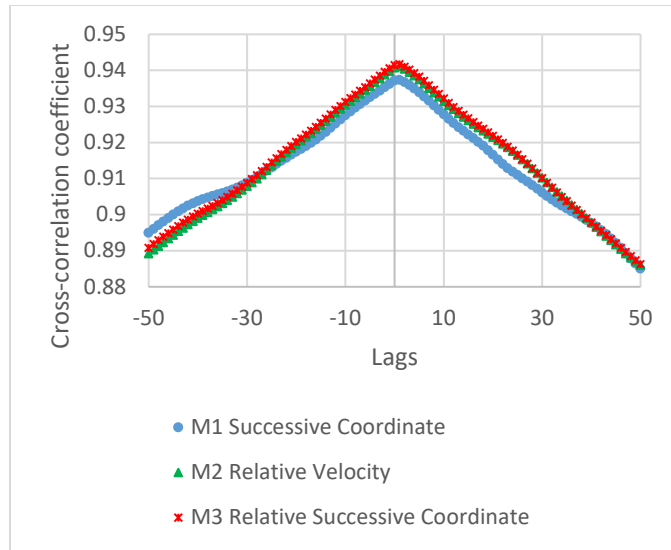


Figure 3.29: Block 1, 40ft, 9 mph, 7-point moving average

3.2.5 *Special Cases in Cross-Correlation Analysis*

For most trials, acceptable structure in cross-correlation was observed once a filter was applied. In the analysis of these trials, a steeper decay in cross-correlation was attributed to either agreement between fluctuations in wind velocity at the two locations, or a long-term trend over the sampled domain for the trial (10 minutes) existent at both locations. In either case, the results were interpreted as evidence of similitude at the two locations. Conversely, a flat shape in the cross-correlogram may indicate an abundance of noise where wind velocity at one location fluctuates so frequently that all lags yield the same cross-correlation. An example of this phenomenon was discovered from Block 2, 40 ft of separation, with a target vehicle velocity of 6 mph. This trial still yielded a good correlation of 0.795 at zero lags but, the correlogram's flat shape suggests that the wind velocity had more random variability throughout the trial (Figure 3.30). This was confirmed after the dataset was visualized, shown in Figure 3.31. Filtering the data

resulted in a slight reduction of noise and an increase in cross-correlation coefficient however, the structure of the resulting cross-correlogram still remained relatively flat. Next, vehicle velocity calculations were consulted to investigate whether there was an abundance of noise skewing the data (Figure 3.32) but it was found to be minimal. In this situation, it's clear that wind velocity over the time period was more variable than other observed times at that distance. By following this procedure, determinations of similitude among wind data for other trials were made.

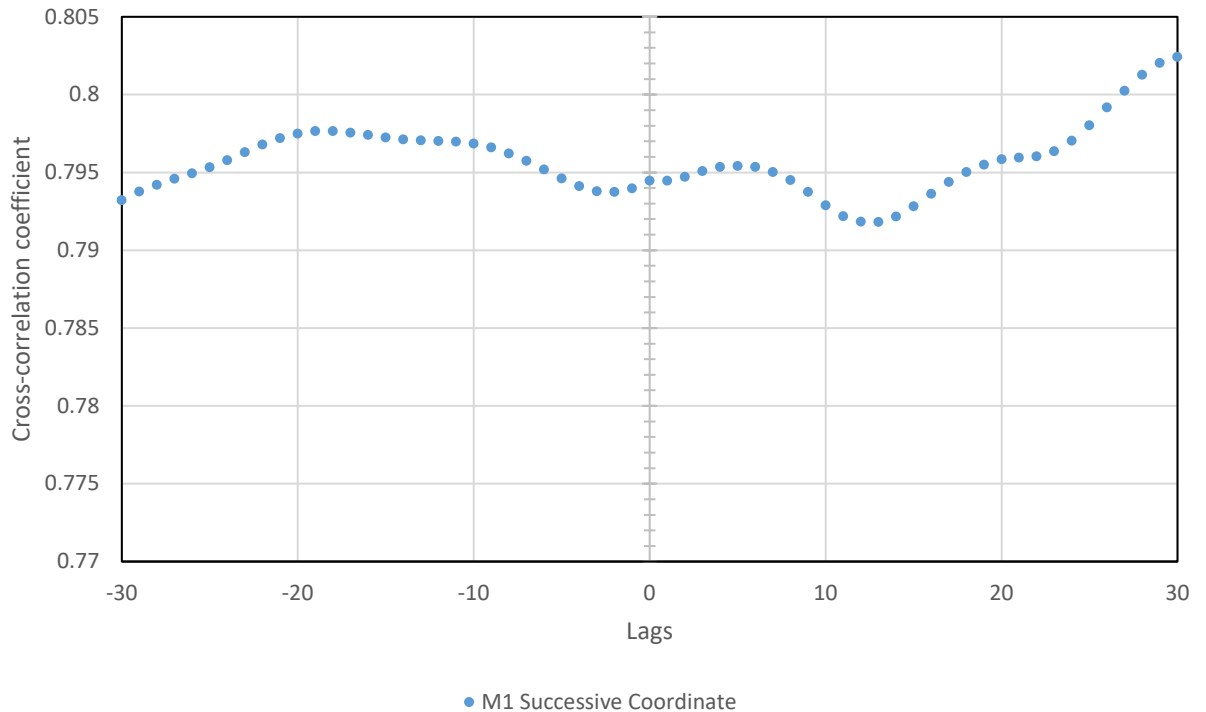


Figure 3.30: The cross-correlogram for the trial in Block 2, 40ft separation, 6 mph target vehicle velocity exhibits a flat shape even with a 7-point moving average applied, suggesting more random variability among one of the time-series datasets

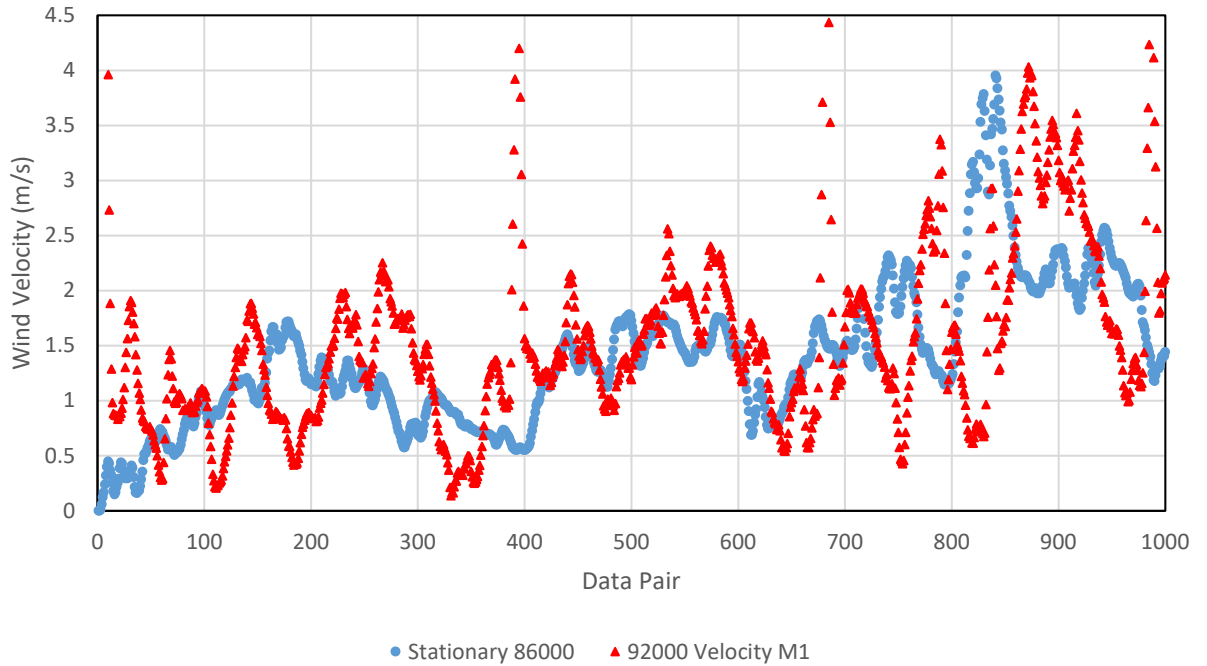


Figure 3.31: Plotted data for Block 2, 40 ft separation, 6 mph targeted vehicle velocity still displays an abundance of noise after filtering with a 7-point moving average

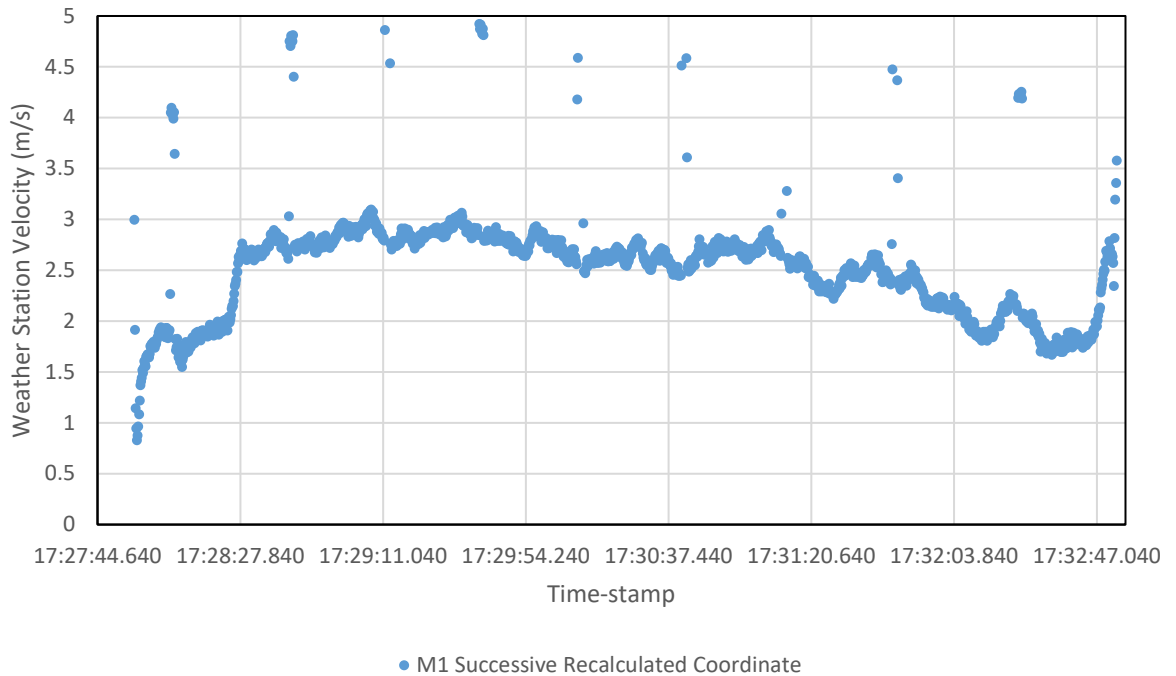


Figure 3.32: Vehicle velocity calculations (Method 1) show minimal noise, eliminating it as a source for irregularities in processed wind data

3.2.6 *Distribution of Measurement Differences*

In addition to previous methods, a more intuitive approach analyzes the differences between stationary measurements and processed dynamic wind velocities. These differences were calculated and placed into bins of width 0.1 m/s. Cumulative probability density plots were generated for each of the three dynamic processing methods. The intended goals for this analysis were to help determine which method of calculating weather station dynamics yielded the closest wind velocities to the stationary anemometer, and to visualize the extent of each filter's smoothing effect on the datasets. Those datasets with larger proportions of small wind velocity differences are easily recognized by their position on the chart above other trials. Figure 3.33 is the cumulative probability chart for the first 20 ft separation trial with a targeted 3 mph vehicle velocity. This observation however was not obvious for all trials. When looking at visualizations of the data and calculated uncertainty, increasing filter size was shown to minimize large differences between wind velocities at the two locations for all trials, targeted velocities, separation distances, and was in agreement to improved cross-correlation when comparing increasing filtering window size. Histograms of the distribution of measurement differences also confirmed this but were not included in this work. The cumulative data masks filtering's effect of minimizing small differences in method 1 because of the prevalence of noise in the dynamic series. Although not shown in plots, there was a decrease in the number of bins (of equal width), as more wind velocity differences were smaller with increasing filtering window.

Comparing target vehicle velocity, the cumulative probability data were ranked for each separation distance to visualize its effect on wind velocity differences and were

displayed as Table 3.3. All 9 mph trials experienced the largest differences in wind velocities at the two locations, suggesting at a glance that faster vehicle speeds may negatively impact the precision of processed dynamic measurements. Considering distance, there was a noticeable pattern of decreasing similarity with increasing vehicle speed at a 20 ft separation distance. For the 40 ft and 60 ft trials however, the 3 mph and 6 mph vehicle speeds become increasing inconsistent in ranking compared to the 9 mph trials. Overall, it appeared that slower speeds yielded the smallest differences between wind data. An example of this trend is displayed at Figure 3.34. When looking at increasing distances alone, there wasn't an apparent trend from this method of analysis. All potential trends aside, every trial experienced roughly at least 75% of wind velocity differences less than or equal 1 m/s, showing great similarity between them.

Table 3.3: Ranking for each trial's cumulative probability data is shown. A rank equal to 1 indicates closer wind velocities between sensors, as a higher percent of velocity pairs have smaller differences.

Trial	3 mph	6 mph	9 mph
20 ft_1	1	2	3
40 ft_1	1	2	3
60 ft_1	1	2	3
20 ft_2	2	1	N/A
40 ft_2	1	2	3
60 ft_2	1	2	3
20 ft_3	1	2	N/A
40 ft_3	2	1	3
60 ft_3	2	1	3

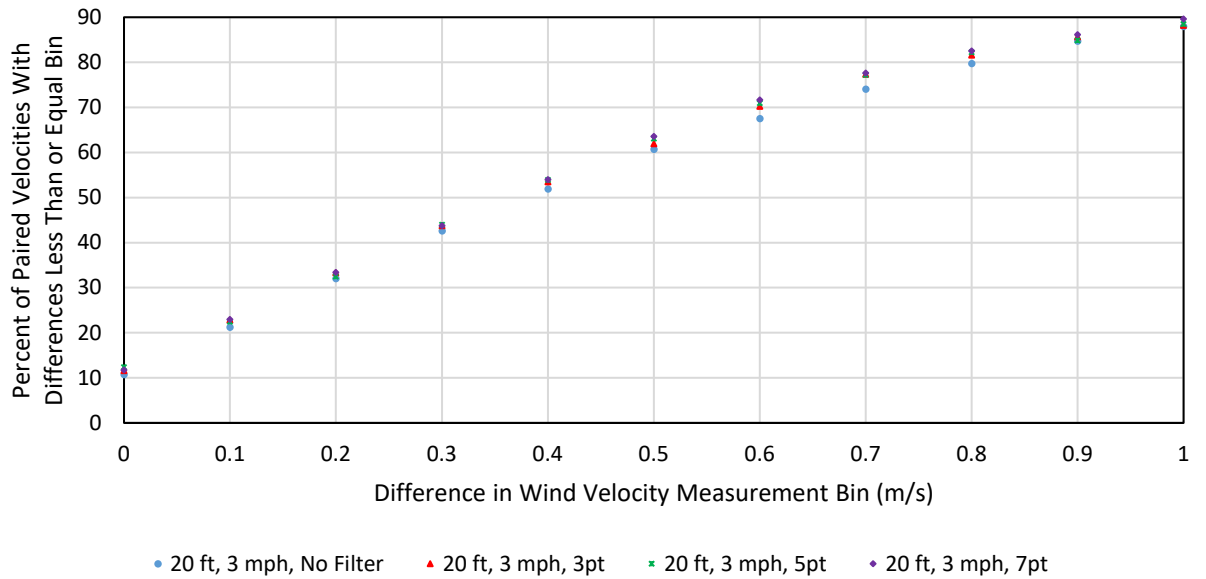


Figure 3.33: Cumulative probability distribution shows increasing the moving average filtering window smooths velocity data and minimizes differences in values

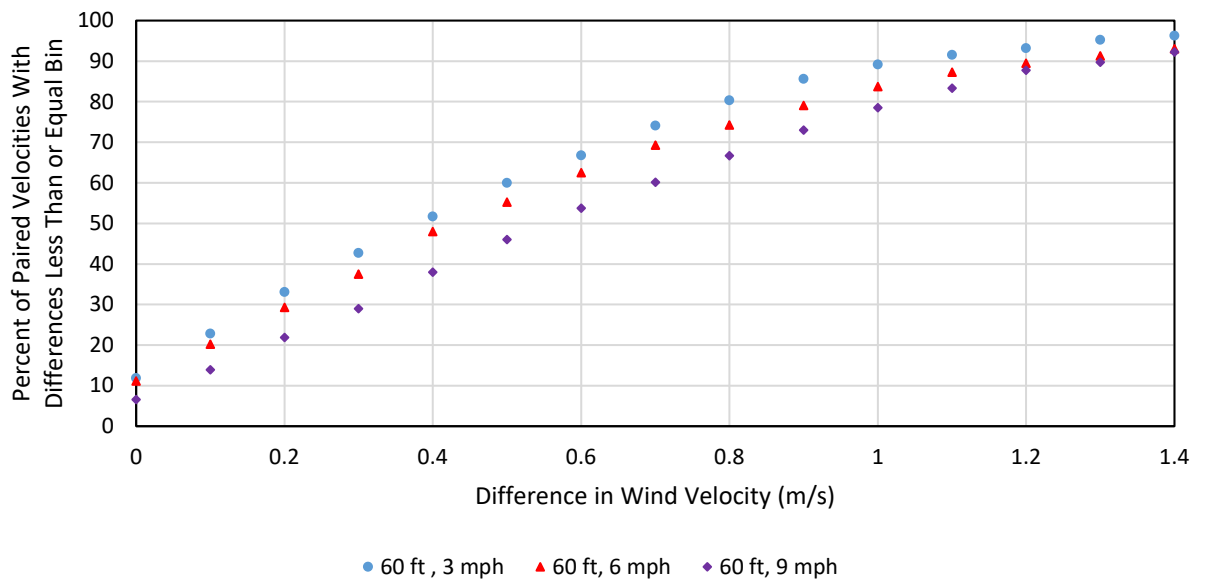


Figure 3.34: Cumulative probability chart shows higher percent of small wind velocity differences at slower vehicle speeds

3.3 Discussion:

3.3.1 *Weather Station Velocity Transformation*

Three methods of calculation/transform were determined feasible to solve for weather station velocity. With similar results for each method at a small offset between the weather station and GNSS receiver, testing of each algorithm may be continued at larger offsets. Method 1, which uses Vincenty's formula for calculating new coordinates using current location, destination distance, heading, then solving for the distance between coordinates divided by the sampling interval for velocity, was the most robust and seemingly accurate method of solving for weather station velocity. The second method, which used the GNSS receiver's velocity and course calculations solved by the Doppler effect between satellite signals and transforming them by relative motion to the weather station's location was a "quick and dirty" method of transform that appeared effective during straight path experimentation but required adjustment to be used correctly. The resulting algorithm required a known physical turn radius, limiting it to either center-pivot irrigation systems or highly parameterized paths with complex conditional statements in the algorithm. The third method used Vincenty/Haversine distance calculations on the original GNSS coordinates, but also suffered the limitations of method 2 for relative motion transformation. To determine which method may be suitable for actuating a variable-rate sprayer system, each method's uncertainty, calculation time, and device sampling rate should be considered.

3.3.1.1 *Uncertainty in Velocity Calculations' Precision*

The resulting velocity calculations at the location of the weather station for the three methods were similar. Vincenty's formula claims highly precise coordinate transformations, but its application in this study had limiting factors. The main limitation was the derivation of headings in the direction of the weather station from the GNSS receiver, where two bearings between coordinates need to be averaged to obtain an approximated instantaneous bearing at each coordinate.

Although three methods of calculation for the weather station velocity were investigated, only similitudes between the results were studied. If the actual precision of each method is desired, a separate experiment may be conducted in which the velocity at two separate locations could be solved and compared using two GNSS receivers. This proposed experiment should record geographical coordinate data while traveling in circular motion as was completed in this chapter however, with a larger offset between the receivers comparable to a commercial spray boom. Using the three methods described in this work, velocities at each GNSS receiver can be calculated by transformation of the data originating from the other receiver so that the predicted velocities can be validated by the GNSS output at the location of the prediction.

Additionally, future work should validate the methods of velocity calculation in this work by investigating transform of GNSS coordinates into a local or regional coordinate system such as Universal Transverse Mercator (UTM) and subsequently calculating the position of the anemometer. The velocity of the anemometer can then be calculated from those successive coordinates. These calculations should be compared with methods in this work for accuracy and calculation time.

In most cases a velocity transform algorithm is probably not needed. When the vehicle is traveling a straight path, the velocities at all points on the moving body are equal. For implements that need to travel curved paths, a more complicated algorithm could be developed that switches transforming portions on and off when traveling across previously defined locations. Alternatively, radius of curvature calculations described could detect turns and trigger a transformation method. The filtering technique could also be refined by implementing band-pass filters to remove obvious outliers from the velocity transform.

3.3.1.2 Considering Calculation Time

An important consideration when implementing the processing algorithms is the calculation time. Since the processing algorithm for dynamic data wasn't completed in real-time, the fastest sampling interval was used, and the results interpolated to the weather station's data series. In this study, the GNSS receiver sampling interval was 50 ms, while the anemometers sampled at 200 ms. To implement either of the three algorithms, the calculation time must not exceed the sampling interval/timespan of filtering to prevent either a backlog of calculations from building or an increasing number of threads. For example, in this study's configuration a 3-point moving average required 600 milliseconds of data recording. The corresponding maximum number of recorded GNSS velocities to be used in vehicle dynamics transformation was 12 points ($600 \text{ ms} / 50 \text{ ms} = 12 \text{ points}$). If the vehicle dynamics or wind data processing algorithms take longer than that time (i.e 600 ms) to complete, the processor will fall behind on calculations it needs to complete and could cause latency in the results or potential program failure. In section 3.1.8 Comparing Methods of Anemometer Velocity, it was

mentioned that method 1 requires the most time to calculate because of an abundance of trigonometric functions however, the difference in calculation time to the Haversine method is often considered negligible especially weighing in the benefits of improved precision. Meanwhile, methods 2 and 3 are notably shorter but sacrifice precision if implemented on paths not well parameterized.

3.3.2 *Feasibility at Varied Distances*

At the beginning of the study, it was expected that the variability between anemometers would increase with distance. The analysis of cross-correlograms generally supported this hypothesis by displaying sharper decays in the correlograms for smaller distances, and flatter structures at farther distances. Despite this, visualizing the data series and an analysis of differenced values showed no apparent trend of increasing differences in wind velocity with distance however, cross-correlograms suggest that the variability between locations differed more, evident in the visuals by less synchronized fluctuations. Even considering non-synchronized fluctuations, both anemometers saw a majority of wind velocity differences less than or equal to 1 m/s. The wind direction was mostly always in sync because the main heading of wind gusts was in the same direction, but even small variations in direction were seen in both series. This validated that the same gusts were present at both sensors, but the intensity was not the same nor expected but was close enough.

CHAPTER 4. CONCLUSIONS AND FUTURE WORK

The objectives of this work aimed to help examine the feasibility of using wind velocity data for controlling droplet spectra in sprayer systems. First, two ultrasonic anemometers were interfaced with GNSS receivers for timestamping with millisecond accuracy. Next, stationary testing looked at similitude between wind velocities at varied separation distances with various moving average filtering window sizes. Finally, a dynamic test was conducted in which a one anemometer was mounted to a utility vehicle, and vehicle dynamics were subtracted from apparent wind velocity for comparison to a stationary anemometer. Various filtering window sizes were tested for noise reduction and their effects on improving similitude between wind velocities in both experiments.

The anemometers were successfully interfaced for accurate timestamping of wind data. From the results of experiments in this study, there seems to be promise for making nozzle decisions in the field at distances many meters away from a recording position for mitigating drift. As expected, wind intensity varies with increasing distance, but there is certainly potential for actuating a nozzle orifice to obtain larger droplets in reaction to gusts, while still maintaining a minimal size to achieve optimal coverage. By applying filters over the timespan of about one second, there was a notable reduction in noise with minimal changes to differences between locations.

APPENDICES

Appendix 1. Schematics for Machined and 3D Printed Parts

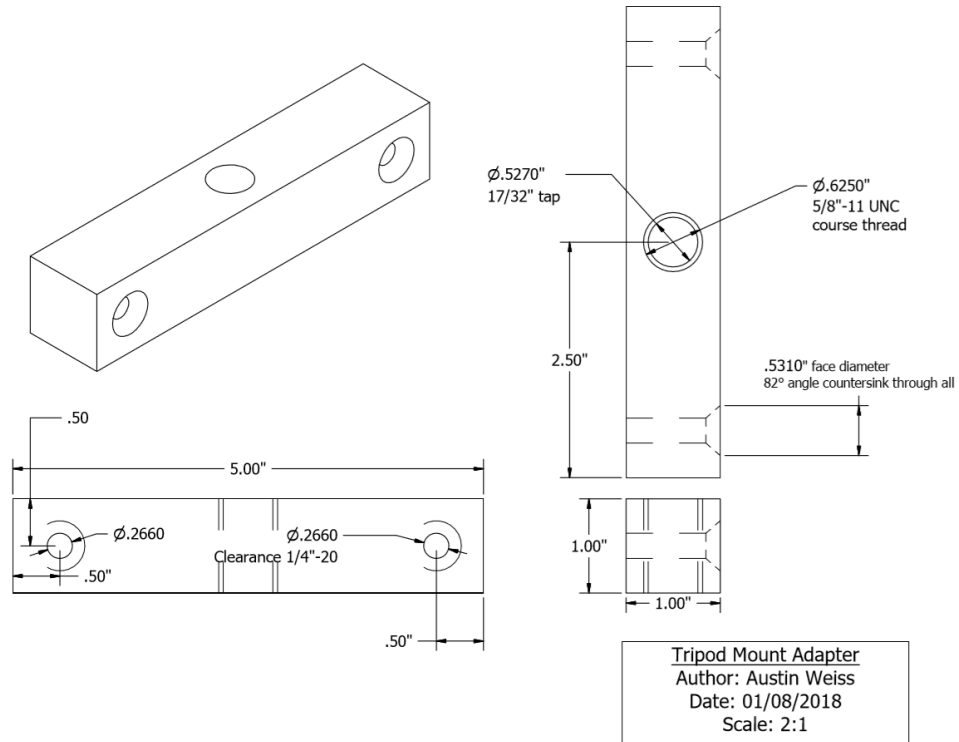


Figure A.1.1: Custom aluminum tripod adapter for 6 ft horizontal 8020 aluminum extrusion attachment.

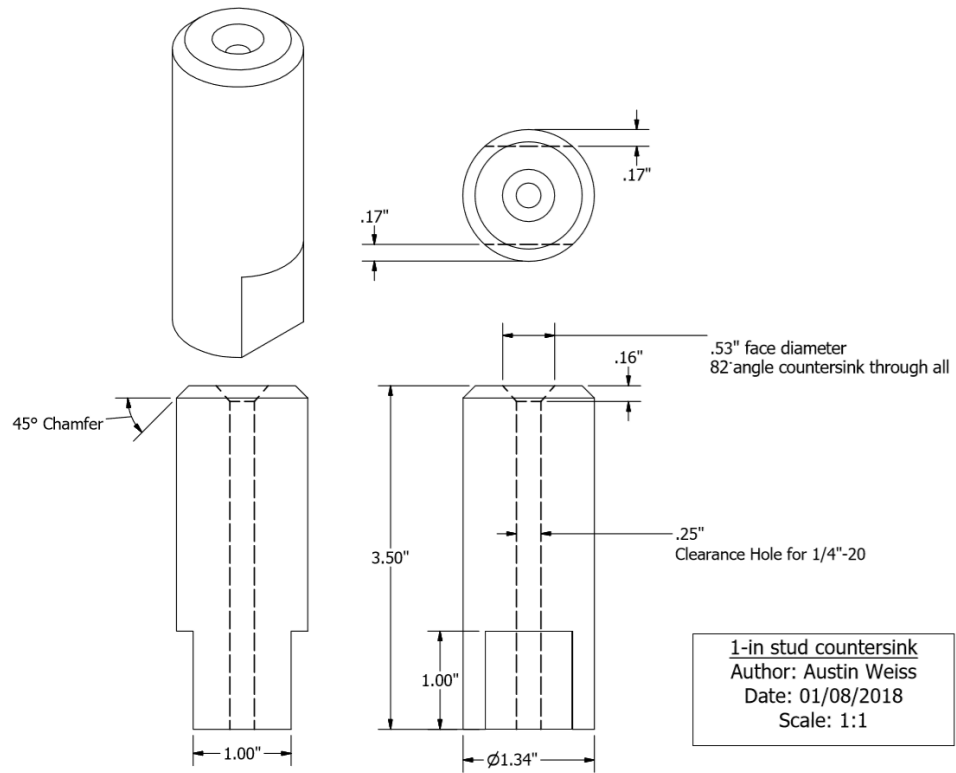


Figure A.1.2: Custom aluminum slug mount for fastening anemometers. Chamfers were cut for ease of mounting and filets so that a wrench could grip the slug for tightening.

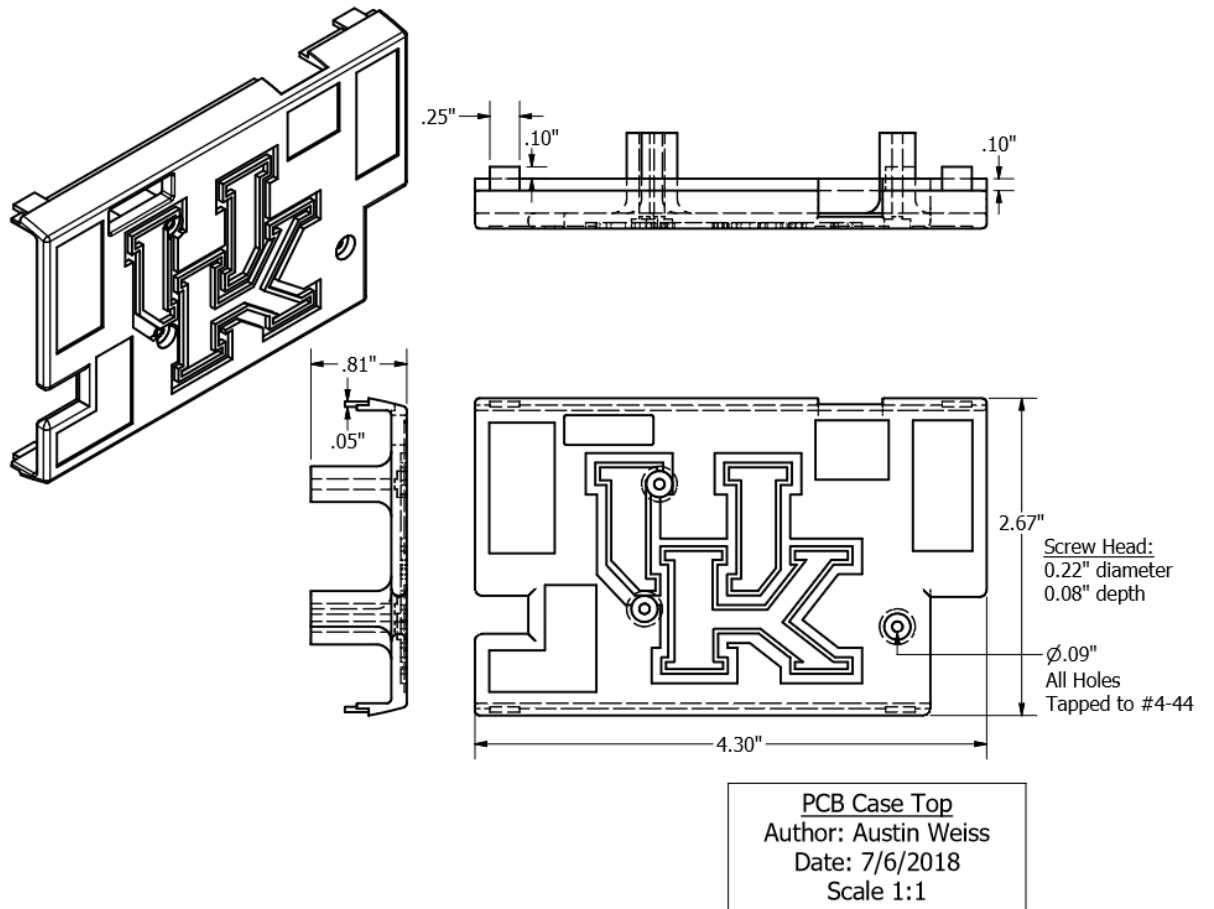


Figure A.1.2: 3D printed lid for PCB enclosure fastens to the bottom piece with 3 #4 machine screws.

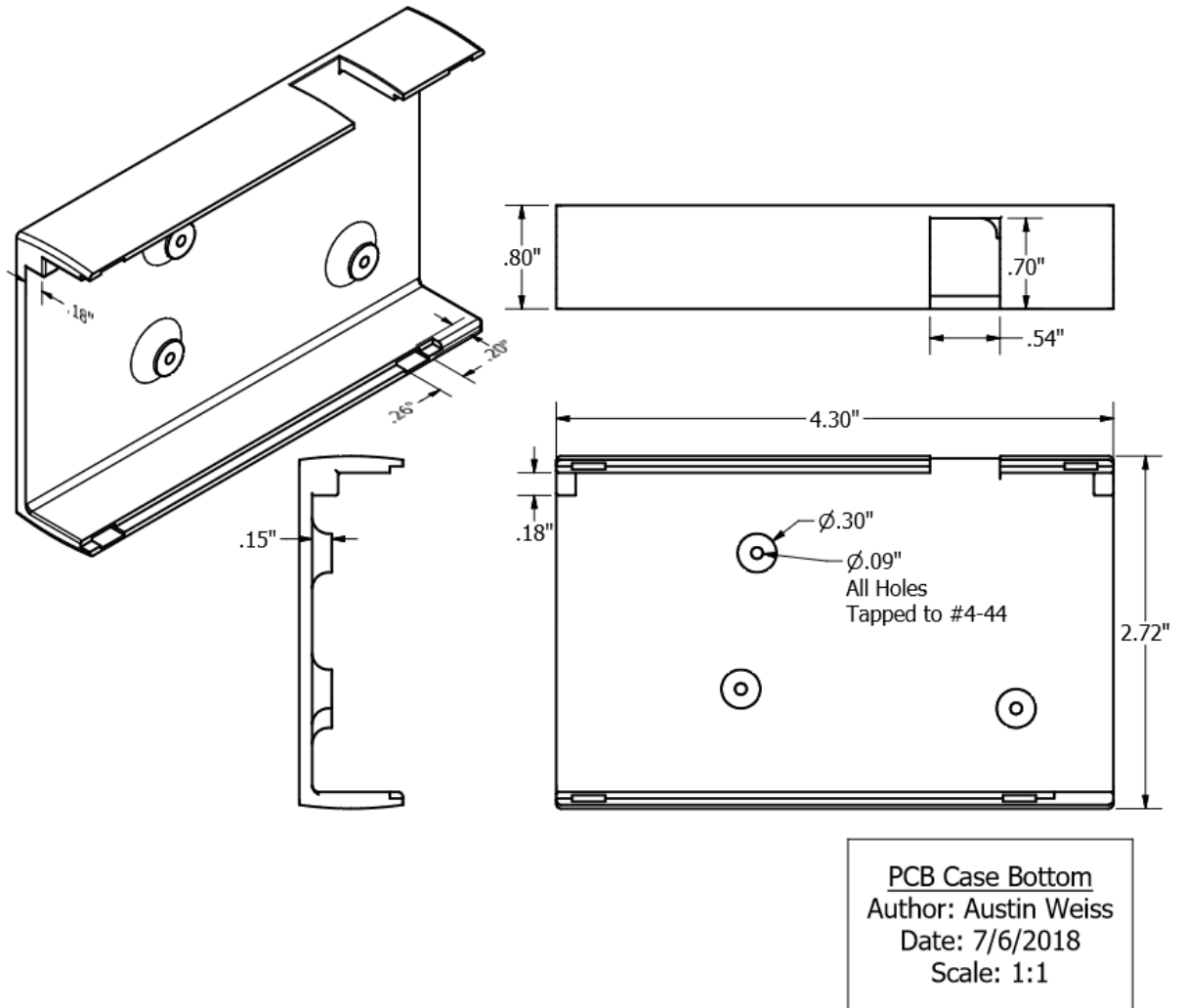


Figure A.1.3: 3D printed base for PCB enclosure rise and secure the board sufficiently to the base.

Appendix 2. MATLAB Code for Stationary Testing

Caller Script:

```
%% stationaryPrompt.m
% Author: Austin Weiss
% Date: 2/19/19
% Description:
%     Loops through raw data files to create repetition data for
stats
%     analysis.
%     Passes filename, PrintSuffix, distance, and filter to
%     Anemometer_process function
%
clear; clc;
colonfix=1; % Fix colon for millisecond timestamp (1=YES)
direction_ask=0; % Directional filtering (0=NO)
velocity_ask=0; % Velocity filtering (0=NO)
repetition_ask={1,3,5,7};

filename={{'Stationary_20ft_1','Stationary_20ft_2','Stationary_20ft_3'}
,...
{'Stationary_40ft_1','Stationary_40ft_2','Stationary_40ft_3'},...
{'Stationary_60ft_1','Stationary_60ft_2','Stationary_60ft_3'}};

%filename={{'Stationary_12ft_1cont'}};

% filename is 3 cells (1 for each distance) with 3 trials inside
each
PrintSuffix={'_NoFilter','_3pt','_5pt','_7pt'};
distance={20,40,60};
%distance={12};
count=1; % For printing row placement
for x=1:numel(distance)
    for f=1:numel(filename(Xie & Wang)) % For each distance
        readFilename{f}=sprintf('%s',filename(Xie & Wang){f},'.csv');
    end
    for i=1:numel(readFilename)
        for j=1:numel(repetition_ask)
            Printfilename=sprintf('%s',filename(Xie &
Wang){i},PrintSuffix{j},'.xlsx');

            cumul=Anemometer_process(readFilename{i},repetition_ask{j},Printfilenam
e,distance(Xie & Wang),colonfix,direction_ask,velocity_ask);
            velthres(1,1)={Printfilename};
            velthres(2,1)={'Difference in Wind Velocity (m/s)'};
            velthres(3,1)={'Percent of Measurements <='};

            velthres(2,2:numel(cumul(:,1)))=cumul(2:numel(cumul(:,1)),3);

            velthres(3,2:numel(cumul(:,1)))=cumul(2:numel(cumul(:,1)),4);

            velthresprint(count,1:numel(velthres(1,:))=velthres(1,:);
```

```

velthresprint(count+1,1:numel(velthres(1,:))=velthres(2,:);

velthresprint(count+2,1:numel(velthres(1,:))=velthres(3,:);
    velthresprint(count+3,1:numel(velthres(1,:))={[]});
    clear velthres;
    count=count+4; % Counter for printing
        end
    end
end
xlswrite('DifferenceAnalysis.xlsx',velthresprint,1)

```

Anemometer Processing:

```

%% Anemometer_process.m
% UltraSonic Anemometer Processing Script
% Author: Austin Weiss
% Last updated: 3/26/2018
% Description:
%     Filtering out errors
%     Separating of sensor data by address
%     Print results in Excel
%     To visualize time in Excel: NEED TO CHANGE NUMBER FORMAT
FOR
%     TIME COLUMNS BACK TO 'HH:MM:SS.000'

function [cumul]=
Anemometer_process(filename,repetition_ask,Printfilename,distance,colon
fix,direction_ask,velocity_ask)
%% Main Code:
TimeANDFiltering=0; % Disable time data (old analysis)

% Inactive variables for TimeANDFiltering=1
%print_ask=input('Do you want to print to Excel? 1 = yes, 0 = no ');
%if print_ask==1
%summary_ask=input('Generate Summary Sheet? 1 = yes, 0 = no ');
%else
%end
% Input window size to filter
%% Sheet Name Generator
% Directional and velocity filtering
if direction_ask==1 && velocity_ask==1
    window=input('Enter filtering window for orthogonal window to
remove in degrees ');
    min_velocity=input('Enter minimum velocity (m/s) to include ');
    max_velocity=input('Enter maximum velocity (m/s) to include (0 will
choose dataset maximum) ');
    windowstring=string(window);
    windowsuffix=' deg orthog rem';

    min_velstring=string(min_velocity);
    velstringsuffix=' m_s vel,';
    % Converting Max velocity to actual if not specified
    if max_velocity==0

```

```

        max_velstring='Max';
    else
        max_velstring=string(max_velocity);
    end
    sheet=sprintf('%s',min_velstring,'-
',max_velstring,velstringsuffix,windowstring,windowsuffix);    %
Creates sheet name
    % Direction only filtering
elseif direction_ask==1 && velocity_ask==0
    window=input('Enter filtering window for both directions in degrees
');
    windowstring=string(window);
    windowsuffix=' deg orth rem';
    sheet=sprintf('%s',windowstring,windowsuffix);    % Creates sheet
name
    % Velocity only filtering
elseif direction_ask==0 && velocity_ask==1
    min_velocity=input('Enter minimum velocity (m/s) to include ');
    max_velocity=input('Enter maximum velocity (m/s) to include (0 will
choose dataset maximum) ');
    min_velstring=string(min_velocity);
    velstringsuffix=' m_s velocity, ';
    % Converting Max velocity to actual if not specified
    if max_velocity==0
        max_velstring='Max';

    else
        max_velstring=string(max_velocity);
    end
    sheet=sprintf('%s',min_velstring,'-',
max_velstring,velstringsuffix);
else
    sheet='Raw';
end

%tabledataa=readtable('2_15_18_raw.csv');
[num,txt,xlsreaddata]=xlsread(filename); % Read in data from file

sensor1=86000;    % Sensor addresses as string (as displayed in
sheet)
sensor2=96000;
k=1;    % Start writing counter for sensor1
m=1;    % Start writing counter for sensor2
%% Scan for Errors and Delete
% For Table read:
%datafilter=rmmissing(tabledataa,'DataVariables',{'Time'});

% For xlsread:
data=xlsreaddata;
n=numel(data(:,1));
j=1;
count=1;
while n>j
    if isnan(data{j,8}) % if NaN
        data(j,:)=[]; % Deleting errors
        count=count+1; % Counting error that are removed.
    end
end

```



```

    for d=1:numel(dataout1(:,1))    %86000 60-120 deg
        if dataout1{d,3}>=90-(window/2) && dataout1{d,3}<=90+(window/2)
            dir(d)=1;
        elseif dataout1{d,3}>=270-(window/2) &&
dataout1{d,3}<=270+(window/2)
            dir(d)=1;
        else
            dir(d)=0;
        end
    end
    dataout1(dir(:)==1,:)=[];

    % 92000:
    dir=zeros(numel(dataout2(:,1)),1);    %preallocate check array
    for d=1:numel(dataout2(:,1))    %92000 240-200 deg
        if dataout2{d,3}>=90-(window/2) && dataout2{d,3}<=90+(window/2)
% IF direction is inside the window, remove!
            dir(d)=1;
        elseif dataout2{d,3}>=270-(window/2) &&
dataout2{d,3}<=270+(window/2)
            dir(d)=1;
        else
            dir(d)=0;
        end
    end
    dataout2(dir(:)==1,:)=[];
else
end

%% For velocity filtering:
if velocity_ask==1
    if max_velocity==0    % Sets maximum check to actual if not
specified by prompt (entered 0)
        max_velocity1=max([dataout1{:,2}]);
        max_velocity2=max([dataout2{:,2}]);
    else
        % Otherwise set maximum for each sensor as chosen:
        max_velocity1=max_velocity;
        max_velocity2=max_velocity;
    end

    if velocity_ask==1    % Prompted at start
        % 86000:
        vel=zeros(numel(dataout1(:,1)),1);    %preallocate check array
        for d=1:numel(dataout1(:,1))    %86000 60-120 deg
            if dataout1{d,2}>max_velocity1 ||
dataout1{d,3}<min_velocity
                vel(d)=1;
            else
                vel(d)=0;
            end
        end
        dataout1(vel(:)==1,:)=[];

        % 92000:
        vel=zeros(numel(dataout2(:,1)),1);    %preallocate check array
        for d=1:numel(dataout2(:,1))    %92000 240-200 deg

```

```

        if dataout2{d,2}>max_velocity2 ||
dataout2{d,3}<min_velocity
            vel(d)=1;
        else
            vel(d)=0;
        end
    end
    dataout2(vel(:)==1,:)=[];
else
end
else
end
end

%% Grabbing data for interpolation
time1=cell2mat(dataout1(:,8));
% Check for duplicates and remove
[newtime1,index,index2]=unique(time1);
time1=time1(index);
dataout1=dataout1(index,:);

velocity1=cell2mat(dataout1(:,2));
direction1=cell2mat(dataout1(:,3));

time2=cell2mat(dataout2(:,8));
velocity2=cell2mat(dataout2(:,2));
direction2=cell2mat(dataout2(:,3));
temperature=cell2mat(dataout2(:,4));
humidity=cell2mat(dataout2(:,5));
pressure=cell2mat(dataout2(:,6));

%% Interpolating 86000 to the time series of 92000
% DECOMPOSING 86000 for interpolation
for i=1:numel(direction1)

    if direction1(i)<=90
        x86(i)=velocity1(i)*sin(direction1(i)*pi/180);
        y86(i)=velocity1(i)*cos(direction1(i)*pi/180);
    elseif direction1(i)<=180 && direction1(i)>90
        x86(i)=velocity1(i)*cos((direction1(i)-90)*pi/180);
        y86(i)=velocity1(i)*sin((direction1(i)-90)*pi/180)*(-1);
    elseif direction1(i)<=270 && direction1(i)>180
        x86(i)=velocity1(i)*sin((direction1(i)-180)*pi/180)*(-1);
        y86(i)=velocity1(i)*cos((direction1(i)-180)*pi/180)*(-1);
    else
        x86(i)=velocity1(i)*cos((direction1(i)-270)*pi/180)*(-1);
        y86(i)=velocity1(i)*sin((direction1(i)-270)*pi/180);
    end
end
end
%% INTERPOLATING COMPONENTS
interp86000_Xvel=interp1(time1,x86,time2,'linear');    % Interpolating
86000 X Velcocity data onto 92000 time series
interp86000_Yvel=interp1(time1,y86,time2,'linear');    % Y data interp
interp86000vel=sqrt(interp86000_Xvel.^2+interp86000_Yvel.^2); %
Magnitude USED FOR COUNTING NUMBER OF POINTS FOR RECALC DIR
%% Recalculating direction: NOT USED, Just for reference:
%(if moving average=1 point, this direction should equal it.)
for i=1:numel(interp86000vel)

```

```

    if interp86000_Xvel(i) >=0 & interp86000_Yvel(i)>=0

interp86000dir(i)=180/pi*atan(abs(interp86000_Xvel(i)/interp86000_Yvel(
i))); % Switch X and Y!?!
    % =180/pi*atan(x/y) I think this is correct
    elseif interp86000_Xvel(i)>=0 & interp86000_Yvel(i)<0

interp86000dir(i)=90+(180/pi*atan(abs(interp86000_Yvel(i)/interp86000_X
vel(i))));
    elseif interp86000_Xvel(i)<0 & interp86000_Yvel(i)<0

interp86000dir(i)=180+(180/pi*atan(abs(interp86000_Xvel(i)/interp86000_
Yvel(i))));
    else

interp86000dir(i)=270+(180/pi*atan(abs(interp86000_Yvel(i)/interp86000_
Xvel(i))));
    end
end

%% DECOMPOSING 92000
for i=1:numel(direction2)
    if direction2(i)<=90
        x92(i)=velocity2(i)*sin(direction2(i)*pi/180);
        y92(i)=velocity2(i)*cos(direction2(i)*pi/180);
    elseif direction2(i)<=180 && direction2(i)>90
        x92(i)=velocity2(i)*cos((direction2(i)-90)*pi/180);
        y92(i)=velocity2(i)*sin((direction2(i)-90)*pi/180)*(-1);
    elseif direction2(i)<=270 && direction2(i)>180
        x92(i)=velocity2(i)*sin((direction2(i)-180)*pi/180)*(-1);
        y92(i)=velocity2(i)*cos((direction2(i)-180)*pi/180)*(-1);
    else
        x92(i)=velocity2(i)*cos((direction2(i)-270)*pi/180)*(-1);
        y92(i)=velocity2(i)*sin((direction2(i)-270)*pi/180);
    end
end
end
%% Creating repetitions:
repetition_ask; % Number of points

%% Moving Average Analysis:
% Moving average on components.. Already interpolated
mag86intx=movmean(abs(interp86000_Xvel),repetition_ask,'Endpoints','dis
card'); % For magnitude velocity
mag86inty=movmean(abs(interp86000_Yvel),repetition_ask,'Endpoints','dis
card');
% Variance for uncertainty
mag86varx=movvar(abs(interp86000_Xvel),repetition_ask,'Endpoints','disc
ard');
mag86vary=movvar(abs(interp86000_Yvel),repetition_ask,'Endpoints','disc
ard');
var86=sqrt(mag86varx.^2+mag86vary.^2);
% For direction:
Mov_avg_86X =
movmean(interp86000_Xvel,repetition_ask,'Endpoints','discard'); %
Interpolated 86000 velocity

```

```

Mov_avg_86Y =
movmean(interp86000_Yvel, repetition_ask, 'Endpoints', 'discard'); %
Interpolated 86000 velocity
% 92000
mag92intx=movmean(abs(x92), repetition_ask, 'Endpoints', 'discard');
mag92inty=movmean(abs(y92), repetition_ask, 'Endpoints', 'discard');
% Variance for uncertainty
mag92varx=movvar(abs(x92), repetition_ask, 'Endpoints', 'discard');
mag92vary=movvar(abs(y92), repetition_ask, 'Endpoints', 'discard');
var92=sqrt(mag92varx.^2+mag92vary.^2);
% For direction:
Mov_avg_92X = movmean(x92, repetition_ask, 'Endpoints', 'discard');
% 92000 velocity
Mov_avg_92Y = movmean(y92, repetition_ask, 'Endpoints', 'discard');
% 92000 velocity
%
%Mov_corr = corrcoef(Mov_avg_86, Mov_avg_92); % Pearson Correlation
calculation
%% Recalculating 86000 direction for filter:
for i=1:numel(Mov_avg_86X)
    if Mov_avg_86X(i) >=0 & Mov_avg_86Y(i) >=0

Mov_avg_dir86(i)=(180/pi*atan(abs(Mov_avg_86X(i)/Mov_avg_86Y(i))));
    elseif Mov_avg_86X(i) >=0 & Mov_avg_86Y(i) <0

Mov_avg_dir86(i)=90+(180/pi*atan(abs(Mov_avg_86Y(i)/Mov_avg_86X(i))));
    elseif Mov_avg_86X(i) <0 & Mov_avg_86Y(i) <0

Mov_avg_dir86(i)=180+(180/pi*atan(abs(Mov_avg_86X(i)/Mov_avg_86Y(i))));
    else

Mov_avg_dir86(i)=270+(180/pi*atan(abs(Mov_avg_86Y(i)/Mov_avg_86X(i))));
    end
end

%% Recalculating 92000 direction for filter
for i=1:numel(Mov_avg_92X)
    if Mov_avg_92X(i) >=0 & Mov_avg_92Y(i) >=0

Mov_avg_dir92(i)=(180/pi*atan(abs(Mov_avg_92X(i)/Mov_avg_92Y(i))));
    elseif Mov_avg_92X(i) >=0 & Mov_avg_92Y(i) <0

Mov_avg_dir92(i)=90+(180/pi*atan(abs(Mov_avg_92Y(i)/Mov_avg_92X(i))));
    elseif Mov_avg_92X(i) <0 & Mov_avg_92Y(i) <0

Mov_avg_dir92(i)=180+(180/pi*atan(abs(Mov_avg_92X(i)/Mov_avg_92Y(i))));
    else

Mov_avg_dir92(i)=270+(180/pi*atan(abs(Mov_avg_92Y(i)/Mov_avg_92X(i))));
    end
end

%% Recombine Components
Mov_avg_86=sqrt(mag86intx.^2+mag86inty.^2); % Use abs for magnitude
average
Mov_avg_92=sqrt(mag92intx.^2+mag92inty.^2);
% Adding Manufacture tolerance
Manufactol92=Mov_avg_92*.02;

```

```

Manufactol86=Mov_avg_86*.02;

Mov_avg_temp =
movmean(temperature,repetition_ask,'Endpoints','discard');
Mov_avg_humid = movmean(humidity,repetition_ask,'Endpoints','discard');
Mov_avg_press = movmean(pressure,repetition_ask,'Endpoints','discard');

num_reps=numel(Mov_avg_86(:,1)); % Number of data points
OUTPUT=cell(num_reps+1,9); % preallocate for speed
OUTPUT(1,:)={'Repetition','Anemometer','Distance (ft)','Wind Velocity
(m/s)','Wind Direction (deg)','Temperature (deg F)','Humidity
(%)','Barometric Pressure (hPa)','Category'};
instant=2; % Used for pairing data points
Randomindex=randperm(num_reps,num_reps); % Non-repeating
randomization
% Preallocating for speed:
Repetition86000=zeros(num_reps,3); % Preallocating size,
repetitions=totalpts/Pointsperrep
Repetition92000=zeros(num_reps,6);

% Writing distance into first column:
Repetition86000(:,1)=distance;
Repetition86000(:,2)=Mov_avg_86(:,1);
Repetition86000(:,3)=Mov_avg_dir86(1,:);

Repetition92000(:,1)=distance;
Repetition92000(:,2)=Mov_avg_92(1,:);
Repetition92000(:,3)=Mov_avg_dir92(1,:);
Repetition92000(:,4)=Mov_avg_temp;
Repetition92000(:,5)=Mov_avg_humid;
Repetition92000(:,6)=Mov_avg_press;

Repetition86000=num2cell(Repetition86000);
Repetition92000=num2cell(Repetition92000);

%% FORMING OUTPUT TABLE:
for i=1:numel(Randomindex)
    Rep_grab=Randomindex(i); % Grabbing randomized repetition
    OUTPUT(instant,1)={i}; % Naming repetition
    OUTPUT(instant,2)={'86000'}; % Writing anemometer name to
row
    OUTPUT(instant,3:5)=Repetition86000(Rep_grab,:);
    % Creating Categorical Variables:
    if Mov_avg_86(Rep_grab) <= 3
        Repetition86000_cat{i,1} = 'Low';
    elseif Mov_avg_86(Rep_grab) > 3 && Mov_avg_86(Rep_grab,1) <= 8
        Repetition86000_cat{i,1} = 'Med';
    else
        Repetition86000_cat{i,1} = 'High';
    end

    OUTPUT(instant,9)=Repetition86000_cat(i);
    instant=instant+1; % Move to next row
    OUTPUT(instant,1)={i}; % Naming repetition
    OUTPUT(instant,2)={'92000'}; % Writing anemometer name to
row
    OUTPUT(instant,3:8)=Repetition92000(Rep_grab,:);

```

```

    if Mov_avg_92(Rep_grab) <= 3
        Repetition92000_cat{i,1} = 'Low';
    elseif Mov_avg_92(Rep_grab) > 3 && Mov_avg_92(Rep_grab) <= 8
        Repetition92000_cat{i,1} = 'Med';
    else
        Repetition92000_cat{i,1} = 'High';
    end
    OUTPUT(instant,9)=Repetition92000_cat(i);
    instant=instant+1;           % Move to next row
end
%% Printing
Mov_avg_92=Mov_avg_92';
var92=var92';
titles=[{'86000 Velocities (m/s)', 'Error 86000', '86000 Direction
(deg)', '92000 Velocity (m/s)', 'Error 92000', '92000 Direction (deg)'}];
p(:,1)=Mov_avg_86;
p(:,2)=sqrt(var86)+ManufacTol86;
p(:,3)=Mov_avg_dir86;
p(:,4)=Mov_avg_92;
p(:,5)=sqrt(var92)+ManufacTol92';
p(:,6)=Mov_avg_dir92;
Plotting='Plot';
xlswrite(Printfilename,titles,Plotting, 'A1')
xlswrite(Printfilename,p,Plotting, 'A2')
xlswrite(Printfilename,OUTPUT, 'Repetitions');

%% Calculating difference in Repetitions & Printing to Excel
difference=abs(Mov_avg_92-Mov_avg_86);
stdevDiff=sqrt(var92(1,:)+var86(:,1)); % These are added to find most
error

%% Histogram setting and extract
dhist = histogram(difference); %Store the histogram results into
variables
dhist.BinWidth=0.1;
dhist.Normalization='cdf';
cumul=cell(numel(difference),4); % Initialize printing cell array
%
difference=num2cell(difference); % Extract and Converting to cells
stdevDiff=num2cell(stdevDiff);
edges=num2cell(dhist.BinEdges)';
countprob=num2cell(dhist.Values*100)';
cumul(1,:)={'Difference in Velocity (m/s)', 'Filter Uncertainty (stdev
of filtered points)', 'Bin Edge', 'Probability'};
cumul(2:numel(difference)+1,1)=difference;
cumul(2:numel(stdevDiff)+1,2)=stdevDiff;
cumul(2:numel(edges)+1,3)=edges;
cumul(2:numel(countprob)+1,4)=countprob; % Fill cells

xlswrite(Printfilename,cumul, 'Difference'); % Printing to Excel

%% Cross-Correlation Analysis:
% Replacing NaNs with zeros:
[row,col]=find(isnan(Mov_avg_92));
Mov_avg_92(row)=0;
clear row;

```

```

[row,col]=find(isnan(Mov_avg_86));
Mov_avg_86(row)=0;
clear row;
% Running cross correlation
[corr,lags]=xcorr(Mov_avg_86,Mov_avg_92,[50],'coeff');
lags=lags';
% Autocorr
auto86=xcorr(Mov_avg_86,50,'coeff');
auto92=xcorr(Mov_avg_92,50,'coeff');
%% Printing Cross Correlation Sheet:
Printcross='Cross Correlation';
corrheads={['Lags','Cross-correlation coefficient','86000
Autocorr','92000 Autocorr'}];
crosscorr=[lags,corr,auto86,auto92];
cross=1;
if cross==1
    xlswrite(Printfilename,corrheads,Printcross,'A1')
    xlswrite(Printfilename,crosscorr,Printcross,'A2')
else
end
end
end

```

Appendix 3. MATLAB Code for Dynamic Testing

Caller Script:

```

%% dynamicPrompt.m
% Author: Austin Weiss
% Date: 2/19/19
% Description:
%     Loops through raw data files to create repetition data for
stats
%     analysis.
%     Passes filename, PrintSuffix, distance, and filter to
%     Dynamicweatherstation.m function
%
clear; clc;
colonfix=1; % Fix colon for millisecond timestamp (1=YES)
direction_ask=0; % Directional filtering (0=NO)
velocity_ask=0; % Velocity filtering (0=NO)
repetition_ask={1,3,5,7};
RTKfile={{'20-3-r1','20-6-r1','20-9-r1'},...
{'40-3-r1','40-6-r1','40-9-r1'},...
{'60-3-r1','60-6-r1','60-9-r1'};...
{'20-3-r2','20-6-r2','Dummy'},...
{'40-3-r2','40-6-r2','40-9-r2'},...
{'60-3-r2','60-6-r2','60-9-r2'};...
{'20-3-r3','20-6-r3','Dummy'},...
{'40-3-r3','40-6-r3','40-9-r3'},...
{'60-3-r3','60-6-r3','60-9-r3'}}; %% ADD THE GNSS FILES

```



```

filename={{'Dynamic_1_20ft_3mph','Dynamic_1_20ft_6mph','Dynamic_1_20ft_
9mph'}}...

{'Dynamic_1_40ft_3mph','Dynamic_1_40ft_6mph','Dynamic_1_40ft_9mph'}}...

{'Dynamic_1_60ft_3mph','Dynamic_1_60ft_6mph','Dynamic_1_60ft_9mph'}};...
{'Dynamic_2_20ft_3mph','Dynamic_2_20ft_6mph','Dummy'}},...

{'Dynamic_2_40ft_3mph','Dynamic_2_40ft_6mph','Dynamic_2_40ft_9mph'}},...

{'Dynamic_2_60ft_3mph','Dynamic_2_60ft_6mph','Dynamic_2_60ft_9mph'}};...
{'Dynamic_3_20ft_3mph','Dynamic_3_20ft_6mph','Dummy'}},...

{'Dynamic_3_40ft_3mph','Dynamic_3_40ft_6mph','Dynamic_3_40ft_9mph'}},...

{'Dynamic_3_60ft_3mph','Dynamic_3_60ft_6mph','Dynamic_3_60ft_9mph'}}};

%filename={{'Stationary_12ft_1cont'}}};

    % filename is 3 cells (1 for each distance) with 3 trials inside
each
PrintSuffix={'_NoFilter','_3pt','_5pt','_7pt'};
distance={20,40,60};
%distance={12};
count=1; % For printing row placement
%%
%           20ft       40ft       60ft
% Trial 1:  [3mph,6,9]   [3,A,9]
% Trial 2:
% Trial 3:

% A location is filename{1,2}{1,2}
for t=1:3
    for x=1:numel(distance)
        for f=1:3
            file_nosuffix=filename{t,x}{1,f};
            readFilename=sprintf('%s',filename{t,x}{1,f},'.csv');
            readgpsFile=sprintf('%s',RTKfile{t,x}{1,f},'.TXT');
            for j=1:numel(repetition_ask)

Printfilename=sprintf('%s',filename{t,x}{1,f},PrintSuffix{j},'.xlsx')
                %try %Should skip holes in the filename matrices
above
                    [cumulHAV,cumulSOG,cumulHAVREL] =
Dynamicweatherstation(readFilename,readgpsFile,repetition_ask{j},Printf
ilename,distance{x},colonfix,direction_ask,velocity_ask,file_nosuffix);
                    %[cumulHAV,cumulSOG,cumulHAVREL] =
Dynamicweatherstation('Dynamic_3_20ft_3mph.csv','20-3-
r3.TXT',1,'Dynamic_3_20ft_3mph_NoFilter.xlsx',20,1,0,0,'Dynamic_3_20ft_
3mph');

                    velthres(1,1)={Printfilename};
                    velthres(2,1)={'Difference in Wind Velocity (m/s)'};
velthres(3,1)={'Percent of Measurements <='};

velthres(2,2:numel(cumulHAV(:,1)))=cumulHAV(2:numel(cumulHAV(:,1)),3);

```

```

velthres(3,2:numel(cumulHAV(:,1)))=cumulHAV(2:numel(cumulHAV(:,1)),4);
    % Adding above result to a running table. Next
iteration will
    % paste below it.

velthresprint(count,1:numel(velthres(1,:))=velthres(1,:);

velthresprint(count+1,1:numel(velthres(1,:))=velthres(2,:);

velthresprint(count+2,1:numel(velthres(1,:))=velthres(3,:);
    velthresprint(count+3,1:numel(velthres(1,:))={ [] };
    clear velthres;
    count=count+4; % Counter for printing
%catch
    % 'There was an error'
    % Printfilename
%end
    end
    end
end
end
end
xlswrite('DifferenceAnalysis.xlsx',velthresprint,1)

%Printfilename='Stationary_40ft_2_NoFilter.xlsx'; % Define filename to
print processed data

```

Dynamic Processing Script

```

%% Austin Weiss
% Dynamicweatherstation.m
% Anemometer & RTK Processing:
% Called Functions:
%     RTKprocess.m - Calculates Haversine and Relative Velocity at
%                   Weather Station location from GNSS data
%     haversine.m - Called by RTKprocess to calculate Haversine
%                   distance and azimuth forward bearings
%     DynamicStats.m - Creates Repetitions and creates tables for
stats
%                   analysis
%     Comp2coord.m - Converts x,y components into 360 deg
coordinates
%
%     vdist.m -      Calculates distance between coordinates
%
%     vreckon.m -   Calculates new coordinates using Vincenty's
%                   algorithm
% KEY VARIABLES:
% HaversineVelocityAnemometer = Calculated Haversine Velocity at
Anemometer position
% courseAnem = Calculated course for Haversine Velocity at Anemometer
position

```

```

% V_anemTransSOG = Velocity at Anemometer position using "Speed over
Ground"
% coursegnd = Course
% HaversineVelocity = Haversine Velocity at GPS position
% HaversineCourse = Haversine course for GPS position
function [cumulHAV, cumulSOG, cumulHAVREL] =
Dynamicweatherstation(filename,RTKfile,repetition_ask,Printfilename,distance,
colonfix,direction_ask,velocity_ask,file_nosuffix)
bothmethplot=0; % CHANGE WHEN want to see vehicle vel calcs
%% OUTPUT NAME:
Filesuff='.xlsx';
%% ANEMOMETER DATA:
[num,txt,xlxreaddata]=xlsread(filename); % Read in data from file

% Run RTK calculations
sensor1=86000; % Sensor addresses as string (as displayed in
sheet)
sensor2=96000;
k=1; % Start writing counter for sensor1
m=1; % Start writing counter for sensor2
%% Scan for Errors and Delete
%colonfix=1; % COLON DELIMITS TIME STAMP!!!

% For xlsread:
data=xlxreaddata;
n=numel(data(:,1));
j=1;
count=1;
while n>j
    if isnan(data{j,8}) % if NaN
        data(j,:)=[]; % Deleting errors
        count=count+1; % Counting error that are removed.
        n=numel(data(:,1));
    elseif isequal(data{j,8},'00:00:00:000')
        data(j,:)=[]; % Deleting errors
        count=count+1; % Counting error that are removed.
        n=numel(data(:,1));
    elseif colonfix==1 && isnumeric(data{j,8})
        data(j,:)=[]; % Deleting errors
        count=count+1; % Counting error that are removed.
        n=numel(data(:,1));
    else
        j=j+1;
    end
end
%% Sorting data by sensor:
if colonfix==1 % If colon needs replaced...
for i=2:numel(data(:,1)) % For all rows in the data sheet
% Fixing millisecond : to . COMMENT OUT AS NEEDED
X(i-1,:)=strsplit(data{i,8},':');
fix=strjoin(X(i-1,:),{':',':', '.'});
data{i,8}=fix;
end
else
end
%% For all timestamps, create numeric values to compare with GNSS
for n=1:numel(X(:,1))

```

```

    Timeane=strjoin(X(n,:),{' ',' ','.'});
    TimeWind{n}=Timeane;
end
TimeWind=TimeWind';
TimeWind=str2double(TimeWind); % Converting to number to compare
%% IMPORTING/Processing GNSS DATA
datastrings = textread(RTKfile, '%s');
% Find incomplete strings to remove:
valid = strfind(datastrings, '$');
i=1; % First position initialized to grab
for n=1:numel(datastrings)
    if valid{n}==1

        GNSScheck=strlength(datastrings(n));
        filtered(i) = datastrings(n);
        i=i+1; % Move to next row
    else
    end
end
filtered=filtered';
%% Parsing GNSS strings and organizing data:
checkGGA=strfind(filtered, '$GNGGA');
checkRMC=strfind(filtered, '$GNRMC');

GGA(1,:)={'Format', 'UTC Time', 'Latitude', 'N/S
Indicator', 'Longitude', 'E/W Indicator', 'GPS Quality Indicator', 'Num
Satellites Used', 'Horizontal dilution of precision HDOP', 'Altitude
(m)', '"Meters"', 'Geoidal Separation (m)', '"Meters"', 'Time Since Last
RTK', 'DGPS Station ID/Checksum'};
RMC(1,:)={'Format', 'UTC Time', 'Status', 'Latitude', 'N/S
Indicator', 'Longitude', 'E/W Indicator', 'Speed over ground
(knots)', 'Course over ground (deg)', 'UTC Date', 'Magnetic Variation
(deg)', 'East/West Indicator', 'Mode Indicator', '*Checksum'};
i=2;
j=2;
% Extract data:
w=1; % Used for finding errors and accounting for them

for n=1:numel(filtered)
    GNSScheck=strlength(filtered(n));
    if checkGGA{n}==1 % Grab GGA (Don't need)
        try
            GGA(i,:)=strsplit(filtered{n,1},',','CollapseDelimiters',... %
Delimits by ',' and space for when DGPS is blank
            false, 'DelimiterType', 'RegularExpression');
            i=i+1;
        catch
        end
    else %checkRMC{n}==1 % Grab RMC
        try
            RMC(j,:)=strsplit(filtered{n,1},',');
            j=j+1;
        catch
            a(w)=j+1;
            w=w+1;
        end
    end
end

```



```

direction1=cell2mat(dataout1(:,3));
%% Removing NaN from data
time1(isnan(time1)) = [];
velocity1(isnan(velocity1)) = [];
direction1(isnan(direction1)) = [];

time2=cell2mat(dataout2(:,8));
velocity2=cell2mat(dataout2(:,2));
direction2=cell2mat(dataout2(:,3));
temperature=cell2mat(dataout2(:,4));    temperature(1)=[];
humidity=cell2mat(dataout2(:,5));        humidity(1)=[];
pressure=cell2mat(dataout2(:,6));        pressure(1)=[];
%% Removing NaN from data:
time2(isnan(time2)) = [];
velocity2(isnan(velocity2)) = [];
direction2(isnan(direction2)) = [];
temperature(isnan(temperature)) = [];
humidity(isnan(humidity)) = [];
pressure(isnan(pressure)) = [];
y=1;
%% Search for starting time, and grab everything after (Eliminating RTK
time before recording)
while UTCTime(1) < time2(1) % COMPARING WITH ANEMOMETER. Extract data
in timeframe
    UTCTime(1)=[];
    UTCTimedbl(1)=[];
    RMC(2,:)=[];
    y=y+1;
end
%% NOW Eliminate weather data that occurred before RTK recording
% Maybe not necessary, allow NaNs
%% Search for end and delete everything after:
% UTCTimedbl=str2double(RMC(:,2)); % Convert GNSS timestamp to
comparable number
time2(isnan(time2)) = []; %Clear out NaNs
while UTCTime(numel(UTCTime)) > time2(numel(time2))
    UTCTime(numel(UTCTime))=[];
    UTCTimedbl(numel(UTCTimedbl))=[];
    RMC(numel(UTCTimedbl),:)=[];
end
%% Extracting GPS INFO:
for j=2:numel(RMC(:,1))
    Latdegmin(j-1)=(insertAfter(RMC(j,4),2,' ')); % Pulling Latitude in
format for Haversine
    Latdegmin(j-1)=strcat(Latdegmin(j-1),RMC(j,5)); % Add N/S
    Longdegmin(j-1)=(insertAfter(RMC(j,6),3,' ')); % Pulling Longitude
    Longdegmin(j-1)=strcat(Longdegmin(j-1),RMC(j,7)); % Add E/W
    velocitygnd(j-1)=str2double(RMC{j,8}); % Speed over ground
(Velocity output from GNSS) %knots
    coursegnd(j-1)=str2double(RMC{j,9}); % Course over ground (deg)
end
%% Calling RTKprocess:
[SuccessiveVelocity,courseAnem,V_anemTransSOG,anemCOG,V_anemRelHav,anem
RelHav,newLong,newLat,Long4Vince,Lat4Vince] =
RTKprocess(Latdegmin,Longdegmin,velocitygnd,coursegnd,distance,repetiti
on_ask);

```

```

%% Interpolation of 86000 Stationary (Break into components,
interpolate, reconstruct)
    for i=1:numel(direction1)

        if direction1(i)<=90
            x86(i)=velocity1(i)*sin(direction1(i)*pi/180)*(-1);
            y86(i)=velocity1(i)*cos(direction1(i)*pi/180)*(-1);
        elseif direction1(i)<=180 && direction1(i)>90
            x86(i)=velocity1(i)*cos((direction1(i)-90)*pi/180)*(-1);
            y86(i)=velocity1(i)*sin((direction1(i)-90)*pi/180)*(1);
        elseif direction1(i)<=270 && direction1(i)>180
            x86(i)=velocity1(i)*sin((direction1(i)-180)*pi/180)*(1);
            y86(i)=velocity1(i)*cos((direction1(i)-180)*pi/180)*(1);
        else
            x86(i)=velocity1(i)*cos((direction1(i)-270)*pi/180)*(1);
            y86(i)=velocity1(i)*sin((direction1(i)-270)*pi/180)*(-1);
        end
    end

    interp86000_Xvel=interp1(time1,x86,time2,'linear');    %
Interpolating 86000 X Velcocity data onto 92000 time series
    interp86000_Yvel=interp1(time1,y86,time2,'linear');    % Y data
interp
    interp86000vel=sqrt(interp86000_Xvel.^2+interp86000_Yvel.^2); %
Magnitude
    interp86000_Xvel(1)=[]; interp86000_Yvel(1)=[];
interp86000vel(1)=[]; % REMOVING FIRST VALUES TO MATCH 92000 processed
    %% Recalculating direction: NOT USED!!! (For reference: should
equal Moving average of 1 point)
interp86000dir=Comp2coord(interp86000_Xvel,interp86000_Yvel);

%% SUCCESSIVE COORDINATE METHOD of Vehicle Velocity:
% Matching UTC time with Calculated Anemometer velocities
HavTimedbl=UTCTimedbl; HavTime=UTCtime;    % Copying time for
Haversine method
HavTimedbl(numel(HavTimedbl))=[]; % Clearing Last time stamp to match
Haversine methods 1 and 3 with SOG method 2
HavTime(numel(HavTime))=[];    %
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

%% Successive INTERPOLATION:
% Creating Midpoint Times to represent time for "Average Velocities"
between GNSS
% coordinates for HAVERSINE METHODS 1 AND 3
for n=2:(numel(HavTimedbl))
    try
        Havtimemidpt(n-1)=(HavTime(n)+HavTime(n-1))/2;
    catch
    end
end
Havtimemidpt=Havtimemidpt';
%% Interpolate Method 1:
SuccessiveVelocity=SuccessiveVelocity';
interpHaversineVel=interp1(Havtimemidpt(2:numel(Havtimemidpt)),Successi
veVelocity,time2,'linear'); % INTERPOLATING CALCULATED
VELOCITY/DIRECTION to anemometer time
SuccessiveCourse=courseAnem;

```

```

interpHaversineCourse=interp1(Havtimemidpt(2:numel(Havtimemidpt)), SuccessiveCourse, time2, 'linear'); % Interpolating course to anemometer sampling

%% Interpolate Method 2: "SPEED OVER GROUND"
interpSOG=interp1(HavTime, V_anemTransSOG, time2, 'linear');
interpSOGCourse=interp1(HavTime, anemCOG, time2, 'linear');

%% Interpolate Method 3:
% Combined: Haversine @ GNSS transformed w/relative motion to weather station interpolation:
interpanemRelHav=interp1(Havtimemidpt, V_anemRelHav, time2, 'linear');
interpRelHavCourse=interp1(Havtimemidpt, anemRelHav, time2, 'linear');

SuccessiveCourse=SuccessiveCourse';
%% SUBTRACTING VEHICLE VELOCITY FROM WIND DATA and Recalculating Wind direction relative to vehicle heading

for i=1:numel(direction2)

    if direction2(i)<=90
        x(i)=velocity2(i)*sin(direction2(i)*pi/180)*(-1);
        y(i)=velocity2(i)*cos(direction2(i)*pi/180)*(-1);
        % Y>0 so subtract vehicle vel
        y2(i)=y(i)+interpHaversineVel(i); % HAVERSINE
        y2s(i)=y(i)+interpSOG(i); % SOG
        y2hs(i)=y(i)+interpanemRelHav(i); % Method 3: SOG on Hav
        % Recalculating Wind Velocity
        VcorrectHAV(i)=sqrt(y2(i)^2+(abs(x(i))^2));
        VcorrectSOG(i)=sqrt(y2s(i)^2+(abs(x(i))^2));
        VcorrectHAVSOG(i)=sqrt(y2hs(i)^2+(abs(x(i))^2));
        % Recalculating Wind Direction (Still relative to "False North")

    elseif direction2(i)<=180 && direction2(i)>90
        x(i)=velocity2(i)*cos((direction2(i)-90)*pi/180)*(-1);
        y(i)=velocity2(i)*sin((direction2(i)-90)*pi/180)*(1);

        % Y<0 so add vehicle vel to (-Y) to effectively subtract
        y2(i)=y(i)-interpHaversineVel(i); % HAVERSINE
        y2s(i)=y(i)-interpSOG(i); % SOG
        y2hs(i)=y(i)-interpanemRelHav(i); % Method 3: SOG on Hav

        % Recalculating Wind Velocity
        VcorrectHAV(i)=sqrt(y2(i)^2+(abs(x(i))^2));
        VcorrectSOG(i)=sqrt(y2s(i)^2+(abs(x(i))^2));
        VcorrectHAVSOG(i)=sqrt(y2hs(i)^2+(abs(x(i))^2));

    elseif direction2(i)<=270 && direction2(i)>180
        x(i)=velocity2(i)*sin((direction2(i)-180)*pi/180)*(1);
        y(i)=velocity2(i)*cos((direction2(i)-180)*pi/180)*(1);

        % Y<0 so add vehicle vel to (-Y) to effectively subtract
        y2(i)=y(i)-interpHaversineVel(i); % HAVERSINE
        y2s(i)=y(i)-interpSOG(i); % SOG
        y2hs(i)=y(i)-interpanemRelHav(i); % Method 3: SOG on Hav

```



```

    % Recalculating Wind Velocity
    VcorrectHAV(i)=sqrt(y2(i)^2+(abs(x(i))^2));
    VcorrectSOG(i)=sqrt(y2s(i)^2+(abs(x(i))^2));
    VcorrectHAVSOG(i)=sqrt(y2hs(i)^2+(abs(x(i))^2));

else
    x(i)=velocity2(i)*cos((direction2(i)-270)*pi/180)*(1);
    y(i)=velocity2(i)*sin((direction2(i)-270)*pi/180)*(-1);
    % Y>0
    y2(i)=y(i)+interpHaversineVel(i);% HAVERSINE:
    y2s(i)=y(i)+interpSOG(i);% SOG
    y2hs(i)=y(i)+interpanemRelHav(i); % Method 3: SOG on Hav
    % Recalculating Wind Velocity
    VcorrectHAV(i)=sqrt(y2(i)^2+(abs(x(i))^2));
    VcorrectSOG(i)=sqrt(y2s(i)^2+(abs(x(i))^2));
    VcorrectHAVSOG(i)=sqrt(y2hs(i)^2+(abs(x(i))^2));

end

end
VcorrectHAV=VcorrectHAV';
VcorrectSOG=VcorrectSOG';
VcorrectHAVSOG=VcorrectHAVSOG';
VcorrectHAV(1)=[]; VcorrectSOG(1)=[]; VcorrectHAVSOG(1)=[]; % REMOVE
NANS
interpHaversineVel(1)=[]; interpSOG(1)=[]; interpanemRelHav(1)=[];
%% Recalculating Wind Direction (Still relative to "False North")
Dir_rel_HAV=Comp2coord(x,y2);
Dir_rel_SOG=Comp2coord(x,y2s);
Dir_rel_HAVSOG=Comp2coord(x,y2hs);
Dir_rel_HAV(1)=[]; Dir_rel_SOG(1)=[]; Dir_rel_HAVSOG(1)=[]; % REMOVE
NANS
interpHaversineCourse(1)=[]; interpSOGCourse(1)=[];
interpRelHavCourse(1)=[];
%% Rotating Dynamic Wind Data Toward Heading of Stationary Data (TRUE
NORTH):
% Need to add vehicle heading to wind direction

for i=1:numel(interpHaversineCourse)
    directionTrueNorth_Hav(i) =
    mod(interpHaversineCourse(i)+Dir_rel_HAV(i),360);%*360;
    directionTrueNorth_SOG(i) =
    mod(interpSOGCourse(i)+Dir_rel_SOG(i),360);%*360;
    directionTrueNorth_HAVSOG(i) =
    mod(interpRelHavCourse(i)+Dir_rel_HAVSOG(i),360);%*360;

end
directionTrueNorth_Hav=directionTrueNorth_Hav';
directionTrueNorth_SOG=directionTrueNorth_SOG';
directionTrueNorth_HAVSOG=directionTrueNorth_HAVSOG';
% Plotting New direction compared to old
% figure(4)
% %scatter(time2,direction2,'.'); hold on;
% scatter(time2,directionTrueNorth_Hav,','r'); hold on;
% scatter(time2,directionTrueNorth_SOG,','b'); hold on;
% scatter(time2,directionTrueNorth_HAVSOG,','m'); hold on;
% legend('Raw Wind Direction','Corrected Wind Direction
(Haversine)','Corrected Wind Direction (SOG)')

```

```

% %
% Comparing Rotated direction to stationary

%% WHENEVER DIFFERENCE IS NEGATIVE: it means the vehicle was moving
faster than the wind
%% Preparing for STATS Analysis:
%% NEED TO AVERAGE COMPONENTS THEN RECALCULATE DIRECTION
% directionTrueNorth_Hav and direction TrueNorth_SOG are corrected
% directions.

% Method 1: HAVERSINE METHOD:
for i=1:numel(directionTrueNorth_Hav)

    if directionTrueNorth_Hav(i)<=90
        xHAV(i)=VcorrectHAV(i)*sin(directionTrueNorth_Hav(i)*pi/180)*(-
1);
        yHAV(i)=VcorrectHAV(i)*cos(directionTrueNorth_Hav(i)*pi/180)*(-
1);
    elseif directionTrueNorth_Hav(i)<=180 &&
directionTrueNorth_Hav(i)>90
        xHAV(i)=VcorrectHAV(i)*cos((directionTrueNorth_Hav(i)-
90)*pi/180)*(-1);
        yHAV(i)=VcorrectHAV(i)*sin((directionTrueNorth_Hav(i)-
90)*pi/180)*(1);
    elseif directionTrueNorth_Hav(i)<=270 &&
directionTrueNorth_Hav(i)>180
        xHAV(i)=VcorrectHAV(i)*sin((directionTrueNorth_Hav(i)-
180)*pi/180)*(1);
        yHAV(i)=VcorrectHAV(i)*cos((directionTrueNorth_Hav(i)-
180)*pi/180)*(1);
    else
        xHAV(i)=VcorrectHAV(i)*cos((directionTrueNorth_Hav(i)-
270)*pi/180)*(1);
        yHAV(i)=VcorrectHAV(i)*sin((directionTrueNorth_Hav(i)-
270)*pi/180)*(-1);
    end
end
% Method 2: "SPEED OVER GROUND" RELATIVE MOTION METHOD:
for i=1:numel(directionTrueNorth_SOG)

    if directionTrueNorth_SOG(i)<=90
        xSOG(i)=VcorrectSOG(i)*sin(directionTrueNorth_SOG(i)*pi/180)*(-
1);
        ySOG(i)=VcorrectSOG(i)*cos(directionTrueNorth_SOG(i)*pi/180)*(-
1);
    elseif directionTrueNorth_SOG(i)<=180 &&
directionTrueNorth_SOG(i)>90
        xSOG(i)=VcorrectSOG(i)*cos((directionTrueNorth_SOG(i)-
90)*pi/180)*(-1);
        ySOG(i)=VcorrectSOG(i)*sin((directionTrueNorth_SOG(i)-
90)*pi/180)*(1);
    elseif directionTrueNorth_SOG(i)<=270 &&
directionTrueNorth_SOG(i)>180
        xSOG(i)=VcorrectSOG(i)*sin((directionTrueNorth_SOG(i)-
180)*pi/180)*(1);
        ySOG(i)=VcorrectSOG(i)*cos((directionTrueNorth_SOG(i)-
180)*pi/180)*(1);

```

```

        else
            xSOG(i)=VcorrectSOG(i)*cos((directionTrueNorth_SOG(i)-
270)*pi/180)*(1);
            ySOG(i)=VcorrectSOG(i)*sin((directionTrueNorth_SOG(i)-
270)*pi/180)*(-1);
        end
    end

% Method 3: HAVERSINE Velocity at GNSS Transformed Relative to
Anemometer:
for i=1:numel(directionTrueNorth_HAVSOG)

    if directionTrueNorth_HAVSOG(i)<=90

xHAVREL(i)=VcorrectHAVSOG(i)*sin(directionTrueNorth_HAVSOG(i)*pi/180)*(-
-1);

yHAVREL(i)=VcorrectHAVSOG(i)*cos(directionTrueNorth_HAVSOG(i)*pi/180)*(-
-1);
        elseif directionTrueNorth_HAVSOG(i)<=180 &&
directionTrueNorth_HAVSOG(i)>90
            xHAVREL(i)=VcorrectHAVSOG(i)*cos((directionTrueNorth_HAVSOG(i)-
90)*pi/180)*(-1);
            yHAVREL(i)=VcorrectHAVSOG(i)*sin((directionTrueNorth_HAVSOG(i)-
90)*pi/180)*(1);
        elseif directionTrueNorth_HAVSOG(i)<=270 &&
directionTrueNorth_HAVSOG(i)>180
            xHAVREL(i)=VcorrectHAVSOG(i)*sin((directionTrueNorth_HAVSOG(i)-
180)*pi/180)*(1);
            yHAVREL(i)=VcorrectHAVSOG(i)*cos((directionTrueNorth_HAVSOG(i)-
180)*pi/180)*(1);
        else
            xHAVREL(i)=VcorrectHAVSOG(i)*cos((directionTrueNorth_HAVSOG(i)-
270)*pi/180)*(1);
            yHAVREL(i)=VcorrectHAVSOG(i)*sin((directionTrueNorth_HAVSOG(i)-
270)*pi/180)*(-1);
        end
    end

%% Moving average of components 92000 (HAVERSINE AND RELATIVE MOTION
RESULTS:
% Method 1: Haversine vehicle method:
magxHavMov_avg_92 =
movmean(abs(xHAV),repetition_ask,'Endpoints','discard');    % 92000
velocity 3 pt. average
magyHavMov_avg_92 =
movmean(abs(yHAV),repetition_ask,'Endpoints','discard');    % 92000
velocity 3 pt. average
magHavMov_avg_92=sqrt(magxHavMov_avg_92.^2+magyHavMov_avg_92.^2);    %
Magnitude
% Variance for uncertainty
var92HAV=movvar(VcorrectHAV,repetition_ask,'Endpoints','discard');
% mag92HAVvarx=movvar(abs(xHAV),repetition_ask,'Endpoints','discard');
% mag92HAVvary=movvar(abs(yHAV),repetition_ask,'Endpoints','discard');
% var92HAV=sqrt(mag92HAVvarx.^2+mag92HAVvary.^2);
% For direction (averaging negatives for vector averages)

```

```

xHavMov_avg_92 = movmean(xHAV, repetition_ask, 'Endpoints', 'discard');
% 92000 velocity 3 pt. average
yHavMov_avg_92 = movmean(yHAV, repetition_ask, 'Endpoints', 'discard');
% 92000 velocity 3 pt. average
%
% Method 2: "SpeedOverGround" Vehicle method:
magxSOGMov_avg_92 =
movmean(abs(xSOG), repetition_ask, 'Endpoints', 'discard');
magySOGMov_avg_92 =
movmean(abs(ySOG), repetition_ask, 'Endpoints', 'discard'); % 92000
velocity 3 pt. average
magSOGMov_avg_92=sqrt(magxSOGMov_avg_92.^2+magySOGMov_avg_92.^2); %
Magnitude
% Variance for uncertainty
var92SOG=movvar(VcorrectSOG, repetition_ask, 'Endpoints', 'discard');
%mag92SOGvarx=movvar(abs(xSOG), repetition_ask, 'Endpoints', 'discard');
%mag92SOGvary=movvar(abs(ySOG), repetition_ask, 'Endpoints', 'discard');
%var92SOG=sqrt(mag92SOGvarx.^2+mag92SOGvary.^2);
% For direction
xSOGMov_avg_92 = movmean(xSOG, repetition_ask, 'Endpoints', 'discard');
% 92000 velocity 3 pt. average
ySOGMov_avg_92 = movmean(ySOG, repetition_ask, 'Endpoints', 'discard');
% 92000 velocity 3 pt. average
%
% Method 3: Haversine vehicle method Transformed by relative motion:
magxHavRelMov_avg_92 =
movmean(abs(xHAVREL), repetition_ask, 'Endpoints', 'discard'); % 92000
velocity 3 pt. average
magyHavRelMov_avg_92 =
movmean(abs(yHAVREL), repetition_ask, 'Endpoints', 'discard'); % 92000
velocity 3 pt. average
magHavRelMov_avg_92=sqrt(magxHavRelMov_avg_92.^2+magyHavRelMov_avg_92.^
2); % Magnitude
% Variance for uncertainty
%mag92HAVRelvarx=movvar(abs(xHAVREL), repetition_ask, 'Endpoints', 'discar
d');
%mag92HAVRelvary=movvar(abs(yHAVREL), repetition_ask, 'Endpoints', 'discar
d');
var92HAVRel=movvar(VcorrectHAVSOG, repetition_ask, 'Endpoints', 'discard')
;
%var92HAVRel=sqrt(mag92HAVvarx.^2+mag92HAVvary.^2);
% For direction (averaging negatives for vector averages)
xHavRelMov_avg_92 = movmean(xHAV, repetition_ask, 'Endpoints', 'discard');
% 92000 velocity 3 pt. average
yHavRelMov_avg_92 = movmean(yHAV, repetition_ask, 'Endpoints', 'discard');
% 92000 velocity 3 pt. average
%
%% RECONSTRUCTING AVERAGED WIND VECTORS: Calculating the DIRECTION %%

dirHAV_Movavg=Comp2coord(xHavMov_avg_92, yHavMov_avg_92); % Haversine
dirSOG_Movavg=Comp2coord(xSOGMov_avg_92, ySOGMov_avg_92); % SOG
dirHAVRel_Movavg=Comp2coord(xHavRelMov_avg_92, yHavRelMov_avg_92);%
Method 3

%% Moving Average 86000

```

```

xMov_avg_86 =
movmean(abs(interp86000_Xvel),repetition_ask,'Endpoints','discard'); %
Interpolated 86000 velocity moving average
yMov_avg_86 =
movmean(abs(interp86000_Yvel),repetition_ask,'Endpoints','discard'); %
Interpolated 86000 velocity moving average
Mov_avg_86=sqrt(xMov_avg_86.^2+yMov_avg_86.^2); % Magnitude
% Variance for uncertainty
mag86varx=movvar(abs(interp86000_Xvel),repetition_ask,'Endpoints','disc
ard');
mag86vary=movvar(abs(interp86000_Yvel),repetition_ask,'Endpoints','disc
ard');
var86=sqrt(mag86varx.^2+mag86vary.^2);
% For direction!
Mov_avg_86X =
movmean(interp86000_Xvel,repetition_ask,'Endpoints','discard'); %
Interpolated 86000 velocity 3 pt. average
Mov_avg_86Y =
movmean(interp86000_Yvel,repetition_ask,'Endpoints','discard'); %
Interpolated 86000 velocity 3 pt. average
%
% Reconstruct 86000 Direction after filtering!:
dir86_Movavg=Comp2coord(Mov_avg_86X,Mov_avg_86Y);

Mov_avg_temp =
movmean(temperature,repetition_ask,'Endpoints','discard');
Mov_avg_humid =
movmean(humidity,repetition_ask,'Endpoints','discard');
Mov_avg_press =
movmean(pressure,repetition_ask,'Endpoints','discard');
diff=0;
if diff==1
MovAvgDiff = abs(HavMov_avg_92 - SOGMov_avg_92);
h=histogram(MovAvgDiff,'Normalization','probability');
else
end

%% Calling DynamicStats for Randomized Repetitions:
[OUTPUTHAV, OUTPUTSOG, OUTPUTHAVREL] = DynamicStats(distance,
Mov_avg_86, dir86_Movavg, magHavMov_avg_92, dirHAV_Movavg,
magSOGMov_avg_92, dirSOG_Movavg, magHavRelMov_avg_92, dirHAVRel_Movavg,
Mov_avg_temp, Mov_avg_humid, Mov_avg_press);

%% Difference Analysis: (Percent values <= velocity)
%% METHOD 1: Calculating difference in Repetitions & Printing to Excel
magHavMov_avg_92=magHavMov_avg_92';
var92HAV=var92HAV';
differenceHAV=abs(magHavMov_avg_92-Mov_avg_86);
stdevDiffHAV=sqrt(var92HAV(:,1)+var86(:,1)); % These are added to find
most error
% Histogram setting and extract

dhistHAV = histogram(differenceHAV); %Store the histogram results into
variables
dhistHAV.BinWidth=0.1;
dhistHAV.Normalization='cdf';
cumulHAV=cell(numel(differenceHAV),4); % Initialize printing cell array

```

```

differenceHAV=num2cell(differenceHAV); % Extract and Converting to
cells
stdevDiffHAV=num2cell(stdevDiffHAV);
edgesHAV=num2cell(dhistHAV.BinEdges)';
countprobHAV=num2cell(dhistHAV.Values*100)';
cumulHAV(1,:)={'Difference in Velocity (m/s)', 'Filter Uncertainty
(stdev of filtered points)', 'Bin Edge', 'Probability'};
cumulHAV(2:numel(differenceHAV)+1,1)=differenceHAV;
cumulHAV(2:numel(stdevDiffHAV)+1,2)=stdevDiffHAV;
cumulHAV(2:numel(edgesHAV)+1,3)=edgesHAV;
cumulHAV(2:numel(countprobHAV)+1,4)=countprobHAV; % Fill cells

%% Method 2:
magSOGMov_avg_92=magSOGMov_avg_92';
var92SOG=var92SOG';
differenceSOG=abs(magSOGMov_avg_92-Mov_avg_86);
stdevDiffSOG=sqrt(var92SOG(:,1)+var86(:,1)); % These are added to find
most error
% Histogram setting and extract
dhistSOG = histogram(differenceSOG); %Store the histogram results into
variables
dhistSOG.BinWidth=0.1;
dhistSOG.Normalization='cdf';
cumulSOG=cell(numel(differenceSOG),4); % Initialize printing cell array

differenceSOG=num2cell(differenceSOG); % Extract and Converting to
cells
stdevDiffSOG=num2cell(stdevDiffSOG);
edgesSOG=num2cell(dhistSOG.BinEdges)';
countprobSOG=num2cell(dhistSOG.Values*100)';
cumulSOG(1,:)={'Difference in Velocity (m/s)', 'Filter Uncertainty
(stdev of filtered points)', 'Bin Edge', 'Probability'};
cumulSOG(2:numel(differenceSOG)+1,1)=differenceSOG;
cumulSOG(2:numel(stdevDiffSOG)+1,2)=stdevDiffSOG;
cumulSOG(2:numel(edgesSOG)+1,3)=edgesSOG;
cumulSOG(2:numel(countprobSOG)+1,4)=countprobSOG; % Fill cells

%% Method 3:
magHavRelMov_avg_92=magHavRelMov_avg_92';
var92HAVRel=var92HAVRel';
differenceHAVREL=abs(magHavRelMov_avg_92-Mov_avg_86);
stdevDiffHAVREL=sqrt(var92HAVRel(:,1)+var86(:,1)); % These are added to
find most error
% Histogram setting and extract
dhistHAVREL = histogram(differenceHAVREL); %Store the histogram results
into variables
dhistHAVREL.BinWidth=0.1;
dhistHAVREL.Normalization='cdf';
cumulHAVREL=cell(numel(differenceHAVREL),4); % Initialize printing cell
array

differenceHAVREL=num2cell(differenceHAVREL); % Extract and Converting
to cells
stdevDiffHAVREL=num2cell(stdevDiffHAVREL);
edgesHAVREL=num2cell(dhistHAVREL.BinEdges)';
countprobHAVREL=num2cell(dhistHAVREL.Values*100)';

```

```

cumulHAVREL(1,:)={'Difference in Velocity (m/s)', 'Filter Uncertainty
(stdev of filtered points)', 'Bin Edge', 'Probability'};
cumulHAVREL(2:numel(differenceHAVREL)+1,1)=differenceHAVREL;
cumulHAVREL(2:numel(stdevDiffHAVREL)+1,2)=stdevDiffHAVREL;
cumulHAVREL(2:numel(edgesHAVREL)+1,3)=edgesHAVREL;
cumulHAVREL(2:numel(countprobHAVREL)+1,4)=countprobHAVREL; % Fill cells

%% Cross-Correlation Analysis:
% Replacing NaNs with zeros:
[row,col]=find(isnan(Mov_avg_86));
Mov_avg_86(row)=0;
[row,col]=find(isnan(magHavMov_avg_92));
magHavMov_avg_92(row)=0;
clear row;
[row,col]=find(isnan(magSOGMov_avg_92));
magSOGMov_avg_92(row)=0;
clear row;
[row,col]=find(isnan(magHavRelMov_avg_92));
magHavRelMov_avg_92(row)=0;
clear row; clear col;
% Running cross correlation
[Havcorr,lags]=xcorr(Mov_avg_86,magHavMov_avg_92,[50], 'coeff');
lags=lags';
Sogcorr=xcorr(Mov_avg_86,magSOGMov_avg_92,[50], 'coeff');
Havrelcorr=xcorr(Mov_avg_86,magHavRelMov_avg_92,[50], 'coeff');

autostationary=xcorr(Mov_avg_86,50, 'coeff');
autoHav=xcorr(magHavMov_avg_92,50, 'coeff');
autoSog=xcorr(magSOGMov_avg_92,50, 'coeff');
autoHavrel=xcorr(magHavRelMov_avg_92,50, 'coeff');

%% Print Excel sheets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
print=1;
if print==1
    %% Plot Sheet:
    titles=[{'86000 Velocities (m/s)', 'Std dev 86000', '86000 Direction
(deg)', '92000 Velocity M1 (m/s)', 'Std dev 92000 M1', '92000 Direction M1
(deg)', '92 Velocity M2 (m/s)', 'Std dev 92 M2', '92 Direction M2', '92
Velocity M3(m/s)', 'Std dev 92 M3', '92 Direction M3'}]];

p(:,1)=Mov_avg_86;
p(:,2)=sqrt(var86);
p(:,3)=dir86_Movavg;
p(:,4)=magHavMov_avg_92;
p(:,5)=sqrt(var92HAV);
p(:,6)=dirHAV_Movavg;
p(:,7)=magSOGMov_avg_92;
p(:,8)=sqrt(var92SOG);
p(:,9)=dirSOG_Movavg;
p(:,10)=magHavRelMov_avg_92;
p(:,11)=sqrt(var92HAVRel);
p(:,12)=dirHAVRel_Movavg;
scatter(1:numel(dirHAV_Movavg), dirHAV_Movavg, '.')
hold on
scatter(1:numel(dir86_Movavg), dir86_Movavg, '.')

```

```

Plotting='Plot';
xlswrite(Printfilename,titles,Plotting,'A1')
xlswrite(Printfilename,p,Plotting,'A2')
%xlswrite(Printfilename,OUTPUT,'Repetitions');
%% Printing Cross Correlation Sheet:
Printcross='Cross Correlation';
corrheads=[{'Lags','M1 Successive Coordinate','M2 Relative
Velocity','M3 Relative Haversine','','86000 Autocorr','Dynamic M1 Hav
Autocorr','M2 Rel SOG Autocorr','M3 HavRel Autocorr'}];
crosscorr=[lags,Havcorr,Sogcorr,Havrelcorr,NaN(numel(lags),1),autostati
onary,autoHav,autoSog,autoHavrel];
cross=1;
if cross==1
xlswrite(Printfilename,corrheads,Printcross,'A1')
xlswrite(Printfilename,crosscorr,Printcross,'A2')

%% Printing DIFFERENCES:
% Sheet names:
DifferenceHAVsheet=strcat('Diff_',file_nosuffix,'HAV');
DifferenceSOGsheet=strcat('Diff_',file_nosuffix,'SOG');
DifferenceHAVRELSheet=strcat('Diff_',file_nosuffix,'HAV_REL');

xlswrite(Printfilename,cumulHAV,DifferenceHAVsheet); % Printing to
Excel
xlswrite(Printfilename,cumulSOG,DifferenceSOGsheet); % Printing to
Excel
xlswrite(Printfilename,cumulHAVREL,DifferenceHAVRELSheet); % Printing
to Excel
else
end
%% Printing Vehicle Velocity and Course Calculations
%Vehicle course calcs
% interpHaversineVel(1)=[]; % Clear NaN in first position
% interpHaversineCourse(1)=[]; % Clear NaN in first position
% interpSOG(1)=[]; % Clear NaN in first position
% interpSOGCourse(1)=[]; % Clear NaN in first position
% interpanemRelHav(1)=[]; % Clear NaN in first position
% interpRelHavCourse(1)=[]; % Clear NaN in first position
time2(1)=[];
vehiclelabel=[{'Time','M1 Successive Recalculated Coordinate','M2
Relative Velocity SOG','M3 Relative Haversine'}];
coursecalcs=[time2,interpHaversineCourse,interpSOGCourse,interpRelHavCo
urse];
velocityprints=[time2,interpHaversineVel,interpSOG,interpanemRelHav];
if cross==1
xlswrite(Printfilename,vehiclelabel,'Vehicle Velocity Calcs','A1'); %
Print heading
xlswrite(Printfilename,velocityprints,'Vehicle Velocity Calcs','A2');
% Print data
xlswrite(Printfilename,vehiclelabel,'Vehicle Course Calcs','A1'); %
Print heading
xlswrite(Printfilename,coursecalcs,'Vehicle Course Calcs','A2');
% Print data
else
end

%% Printing Randomized Repetitions from DynamicStats.m

```



```

elHav,newLong,newLat,Long4Vince,Lat4Vince] =
RTKprocess (Latdegmin,Longdegmin,velocitygnd,coursegnd,separation,repeti
tion_ask)
old=0;
t=0.05; % time interval (s)
d=31.25*0.0254; %Offset in meters to anemometer
f=0.5*0.0254; % Offset in meters to anemometer
r_earth = 6371.037*1000; %meter radius of Earth
%% Calculating Haversine Distance & Course at GPS Location
for n=2:numel (Latdegmin)
    coord1=strcat (Latdegmin{n-1},',',Longdegmin{n-1});
    coord2=strcat (Latdegmin{n},',',Longdegmin{n});
    [distance (n-1),HaversineCourse (n-1)]=haversine (coord1,coord2); % km
distance
end
HaversineCourse2=HaversineCourse; % Creating copy for later use in calc
bearing GNSS->Anemometer

%% Haversine Velocity at GPS LOCATION:
HaversineVelocity=distance (:)./t; %km/s
HaversineVelocity=HaversineVelocity*1000; %m/s
%velocity=velocity.*1943.84; % knots
HaversineVelocity=HaversineVelocity';
%% Haversine at Anemometer location!
% Calculating Bearing towards anemometer
% ROTATING GNSS HAVERSINE COURSE to face Anemometer
for n=1:numel (HaversineCourse2)+1
    % Create Bearing FROM GPS TO ANEMOMETER:
    % HaverGps2AnemBearing (n)=HaversineCourse2 (n);
    %
    % - - previously
    % HaverGps2AnemBearing (n) =
    HaverGps2AnemBearing (n)+(90+(atan (f/d)*180/pi)); % Subtract difference
in
    % HaverGps2AnemBearing (n)=mod (HaverGps2AnemBearing (n),360);
    %% Lat:
    temp = regexp (Latdegmin{n},',','split'); % STARTING ON FIRST
COORDINATE!
    % LAST WILL BE IGNORED AS IT WAS WON'T HAVE HEADING TO CALCULATE
COORDINATE FROM
    temp{2} (numel (temp{2}))=[]; % Remove Cardinal direction letter
    Latdegree (n) = str2double (temp{1}) + str2double (temp{2})/60;
    clear temp;
    %% Long:
    temp = regexp (Longdegmin{n},',','split'); % SEE LAT FOR EXPLAIN n!
    temp{2} (numel (temp{2}))=[]; % Remove Cardinal direction letter
    Longdegree (n) = str2double (temp{1}) + str2double (temp{2})/60;
    clear temp;
    % Convert Latitude and longitude in radians for transform to anem
    % location

    Lat4Vince (n)=Latdegree (n);
    Long4Vince (n)=Longdegree (n);

%% VINCENTY
vince=1;
if vince==1
    lng=sqrt (d^2+f^2); % OFFSET DISTANCE

```

```

    offsetAngle = asin(f/lng)*180/pi;
    for i=2:numel(Lat4Vince)
        [dist(i-1),vinceCourse(i-1)] = vdist(Lat4Vince(i-1),Long4Vince(i-1),Lat4Vince(i),Long4Vince(i));
        %[dN1(i-1),cN1(i-1)] = vdist(Lat4Vince(i-1),Long4Vince(i-1),90,135); % Point 1 Distance to the North Pole
        %[dN2(i-1),cN2(i-1)] =
vdist(Lat4Vince(i),Long4Vince(i),90,135); % Point 2 Distance to the North Pole
    end

%% Converting to 360 degree format: initial bearing
%vinceCourse=vinceCourse*180/pi;
vinceCourse=coursegnd;
vinceCourse(1)=[];
vinceCourse=vinceCourse*(-1)+360;
vinceCourse=mod(vinceCourse+180,360);
Lat4Vince(1)=[]; Long4Vince(1)=[]; % Clearing bc there's no data for these
%% Calculating angle facing transform direction
for n=2:numel(vinceCourse)
    %%%% %ang(n)=180-(offsetAngle+(real(acos(((dN1(n)^2)-(dN2(n)^2)+(dist(n)^2))/(2*dN1(n)*dist(n)))))*180/pi)); % Angle with North Pole
    % Angle at North pole Not needed.. ang(n-1)=offsetAngle+90-((real(acos(((dN1(n)^2)+(dN2(n)^2)-(dist(n)^2))/(2*dN1(n)*dN2(n)))))*180/pi) + 0.5*(real(acos(((dN1(n)^2)+(dN2(n)^2)-(dist(n)^2))/(2*dN1(n)*dN2(n)))))*180/pi);
    ang(n-1)=90+offsetAngle;
    course(n-1)=mod(((vinceCourse(n)+vinceCourse(n-1))/2)+ang(n-1),360);
    [newLat(n-1),newLong(n-1),finalbear(n-1)]=vreckon(Lat4Vince(n-1),Long4Vince(n-1),lng,course(n-1));%/(1/cos((Lat4Vince(n)-Lat4Vince(n-1)*pi/180)));
end
% %% CONVERTING BACK TO -180,180 DEGREE FORMAT DONT USE HERE
% if course(n)>180
% course(n)=course(n)-180;
% course(n)=course(n)*(-1);
% else
% course(n)=mod(360-course(n),180);
% end
%% Testing distance
for k=1:numel(newLat)
    [dist1(k),vinceCourse1(k)] =
vdist(Lat4Vince(k),Long4Vince(k),newLat(k),newLong(k));
end
% n=1000
% % scatter(Long4Vince(100:n),Lat4Vince(100:n),'.')
% % hold on
% % scatter(newLong(100:n),newLat(100:n),'.')

%% CALCULATING TRANSFORMED DISTANCE (Note inconsistency):
for k=2:numel(newLat)
    [distanceVINCE(k-1),vinceCourse1(k-1)] = vdist(newLat(k-1),newLong(k-1),newLat(k),newLong(k));

```

```

end
vinceCourse1=vinceCourse1*180/pi;
vinceCourse1=vinceCourse1*(-1)+360;
vinceCourse1=mod(vinceCourse1+180,360);
courseAnem=vinceCourse1;
% scatter(1:numel(vinceCourse),vinceCourse, '.')
% hold on
% scatter(1:numel(vinceCourse1),vinceCourse1, '.')

% hold on
% scatter(1:numel(courseAnem),courseAnem, '.')
% hold on
% scatter(1:numel(coursegnd),coursegnd, '.')

%dist1(1)=[];
%% Velocity calcs, smoothing, assignment to output
% dN1(1)=[];
% dist(1)=[];
SucessiveVelocity=distanceVINCE/t;
%HaversineVelocityAnemometer=(HaversineVelocity.*(lng+dN1)./dN1)/t;
if repetition_ask==0
    smoothVelocity=movmean(SucessiveVelocity,4);
else
    smoothVelocity=movmean(SucessiveVelocity,(4*repetition_ask));
end
SucessiveVelocity=smoothVelocity;
%HaversineVelocityAnemometer=(distanceVINCE-dist1)/t;
% scatter(1:numel(test),test, '.')
% hold on
%
scatter(1:numel(HaversineVelocityAnemometer),HaversineVelocityAnemometer, '.')
% hold on
% scatter(1:numel(HaversineVelocity),HaversineVelocity, '.')
% hold on
% scatter(1:numel(dist1),dist1, '.')

%% PRINTING COORDINATE DATA
% Lat4Vince(numel(Lat4Vince))=[]; Long4Vince(numel(Lat4Vince))=[]; %
Clearing bc there's no data for these
%
% titlee=[{'Original Long'},{'Original Lat'},{'New Long'},{'New Lat'}];
% p(:,1)=Long4Vince(1:numel(Long4Vince));
% p(:,2)=Lat4Vince(1:numel(Long4Vince));
% p(:,3)=newLong;
% p(:,4)=newLat;
% xlswrite('Coordinates_1_20ft.xlsx',titlee)
% xlswrite('Coordinates_1_20ft.xlsx',p,1,'A2')

end
% for n=2:numel(newLat)
%     [distanceVINCE(n-1),HaversineCourseVINCE(n-1)]=haversine([newLat(n-1) newLong(n-1)],[newLat(n) newLong(n)]); % km
distance
% end
% distanceVINCE=distanceVINCE*1000;
% HaversineVelocityAnemometer=distanceVINCE/t;

```

```

% courseAnem=(HaversineCourseVINCE*-1)+360;
%
% scatter(1:numel(courseAnem),courseAnem, '.')
% hold on
% scatter(1:numel(HaversineCourse)-
1,HaversineCourse(2:numel(HaversineCourse)), '.')
% hold on
% scatter(1:numel(coursegnd)-2,coursegnd(3:numel(coursegnd)), '.')

% figure(2)
% scatter(Long4Vince(1:100),Lat4Vince(1:100), '.')
% hold on
% scatter(newLat(1:100),newLong(1:100), '.')
% figure(2)
% scatter(1:numel(HaversineCourse),HaversineCourse, '.')
% hold on
% scatter(1:numel(courseAnem),courseAnem, '.')
% hold on
% fi=mod(finalbear+90,360);
% scatter(1:numel(fi),fi, '.')
%
%
scatter(1:numel(HaversineVelocityAnemometer),HaversineVelocityAnemomete
r, '.')
% hold on
% scatter(1:numel(HaversineVelocity),HaversineVelocity, '.')

%% OLD METHOD OF calculating new GPS coordinate: Very Noisy
old=1
if old==1
    HaverGps2AnemBearing=vinceCourse1*pi/180;

[HaversineVelocityAnemometer,courseAnem1]=old(Latdegree,Longdegree,d,f,
r_earth,t);
else
end
%% Calculating Velocity at Anemometer based on "Speed over ground"
% *****
%velocitygnd2=velocitygnd; % FOR INTERPOLATION S.O.G ESTIMATE
velocitygnd=velocitygnd*0.514444; %m/s
count=1;
for n=2:numel(velocitygnd)
    Vgps(n-1) = (velocitygnd(n)+velocitygnd(n-1))/2; % AVERAGE SPEED OF
TWO POINTS AS CENTER POINT
    DistSOG(n-1) = Vgps(n-1)*t; % DISTANCE TRAVELED (METERS) based on
AVERAGE SPEED

    %% HANDLING 359-> 1 deg change!
    anglediff(n-1) = coursegnd(n)-coursegnd(n-1); % Angle difference to
calculate turn radius
    if coursegnd(n)<90 && coursegnd(n-1)>270
        c2=360-coursegnd(n-1);
        anglediff(n-1)=c2+coursegnd(n);

        %anglediff(n-1)=mod(coursegnd(n)+coursegnd(n-1),360);
    else
    end
end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if anglediff(n-1)<0
    count=count+1;
    anglediff(n-1)=abs(anglediff(n-1));
    %turnrad(n-1)=DistSOG(n-1)/(2*tan(anglediff(n-1)*pi/180/2)); %
Straight line approx XXX
    turnrad(n-1)=DistSOG(n-1)*360/(2*pi*anglediff(n-1)); % GNSS
Turn Radius using Arc length
    %turnrad(n-1)=Vgps(n-1)/(anglediff(n-1)/t*pi/180);
    anemTurn(n-1)= (turnrad(n-1)+
((d/2)/((separation*0.3048)+(d/2)))); % TURN RADIUS AT ANEMOMETER
    %anemTurn(n-1)=DistSOG(n-
1)*((separation*0.3048+(d/2))/separation*0.3048)*360/(2*pi*anglediff(n-
1));
    %anemTurn(n-1)= (turnrad(n-1)+ ((d/2)/((turnrad(n-1))+d/2)));
% TURN RADIUS AT ANEMOMETER
    %anemTurn(n-1)= (d+turnrad(n-1));
    %anemTurn(n-1)= sqrt((turnrad(n-1)+ d)^2 + dforward^2); % TURN
RADIUS AT ANEMOMETER
    %V_anemTransSOG(n-1) = Vgps(n-1)*((turnrad(n-1)+d)/turnrad(n-
1));
    V_anemTransSOG(n-1) = Vgps(n-1)*anemTurn(n-1)/(turnrad(n-1));
    %% Adjusting for forward offset
    theta=atan(f/d);
    C=sqrt((f^2)+(d^2));
    x=d*cos(theta*pi/180);
    q=C-x;
    q=q/((separation*0.3048)+q);
    %q=sqrt(((V_anemTransSOG(n-1)*t)^2)+(dforward^2));
    anemTurn(n-1)=anemTurn(n-1)+q;
    anemCOG(n-1)=coursegnd(n-1)+ (asin(f/anemTurn(n-1))*180/pi); %
adding course difference of weather station to course of GNSS
    V_anemTransSOG(n-1)= Vgps(n-1)*anemTurn(n-1)/(turnrad(n-1));

else
    anglediff(n-1)=abs(anglediff(n-1));
    %turnrad(n-1)=DistSOG(n-1)/(2*tan(anglediff(n-1)*pi/180/2)); %
Straight line approx XXX
    turnrad(n-1)=DistSOG(n-1)*360/(2*pi*anglediff(n-1)); % GNSS
Turn Radius using Arc length
    %turnrad(n-1)=Vgps(n-1)/(anglediff(n-1)/t*pi/180);
    %anemTurn(n-1)= sqrt((turnrad(n-1)-d)^2 + dforward^2); % TURN
RADIUS AT ANEMOMETER
    anemTurn(n-1)= (turnrad(n-1)-
((d/2)/((separation*0.3048)+(d/2)))); % TURN RADIUS AT ANEMOMETER
    %anemTurn(n-1)=(turnrad(n-1)+d)*pi*anglediff(n-1)/t/180;
    %anemTurn(n-1)=DistSOG(n-
1)*((separation*0.3048)/(separation*0.3048)+(d/2))*360/(2*pi*angledif
f(n-1));
    %anemTurn(n-1)= (turnrad(n-1)- ((d/2)/((turnrad(n-1))+d/2)));
% TURN RADIUS AT ANEMOMETER
    %anemTurn(n-1)=turnrad(n-1)-d;

    %V_anemTransSOG(n-1) = Vgps(n-1)*anemTurn(n-1)/(turnrad(n-1));
    V_anemTransSOG(n-1) = Vgps(n-1)*(anemTurn(n-1)/(turnrad(n-1)));
    %V_anemTransSOG(n-1) = Vgps(n-1)/(anemTurn(n-1)/(turnrad(n-
1)));

```

```

    %% Adjusting for forward offset
    theta=atan(f/d);
    C=sqrt((f^2)+(d^2));
    x=d*cos(theta*pi/180);
    q=(C-x);
    q=q/((separation*0.3048)+q);

    %q=sqrt(((V_anemTransSOG(n-1)*t)^2)+(dforward^2));
    anemTurn(n-1)=anemTurn(n-1)-q;
    anemCOG(n-1)=coursegnd(n-1)-(asin(f/anemTurn(n-1))*180/pi);
    V_anemTransSOG(n-1)=Vgps(n-1)*(anemTurn(n-1)/(turnrad(n-1)));

end
end
anemCOG=real(anemCOG);
%% Combining Methods: Using Haversine at GNSS location and using
relative velocity to transform
HaversineVelocity;
HaversineCourse;
count=1;
for n=2:numel(HaversineCourse)

    %VgpsHav(n-1) = (HaversineVelocity(n)+HaversineVelocity(n-1))/2; %
AVERAGE SPEED OF TWO POINTS AS CENTER POINT
    %DistHav(n-1) = VgpsHav(n-1)*t; % DISTANCE TRAVELED (METERS)
based on AVERAGE SPEED
    DistHav(n-1)=distance(n-1)*1000;
    VgpsHav(n-1)=distance(n-1)*1000/t;
    %% HANDLING 359-> 1 deg change!
    anglediffHav(n-1) = HaversineCourse(n)-HaversineCourse(n-1); %
Angle difference to calculate turn radius
    % if HaversineCourse(n)>270 && HaversineCourse(n-1)<90
    % anglediffHav(n-1)=360-HaversineCourse(n)+HaversineCourse(n-
1);
    % else
    % %HaversineCourse(n-1)<90 && HaversineCourse(n)
    % end
    %anglediffHav(n-1) = HaversineCourse(n)-HaversineCourse(n-1); %
Angle difference to calculate turn radius
    if anglediffHav(n-1)<0
        count=count+1;
        anglediffHav(n-1)=abs(anglediffHav(n-1));
        %turnrad(n-1)=DistSOG(n-1)/(2*tan(anglediff(n-1)*pi/180/2)); %
Straight line approx XXX
        turnradHav(n-1)=DistHav(n-1)*360/(2*pi*anglediffHav(n-1)); %
GNSS Turn Radius using Arc length

        %anemTurnHav(n-1)= sqrt((turnradHav(n-1)+ d)^2 + dforward^2);
% TURN RADIUS AT ANEMOMETER
        %anemTurnHav(n-1)= (turnradHav(n-1)+ (d/(20*0.3048))); % TURN
RADIUS AT ANEMOMETER
        anemTurnHav(n-1)= (turnradHav(n-1)+
((d/2)/((separation*0.3048)+(d/2)))); % TURN RADIUS AT ANEMOMETER
        %anemTurnHav(n-1)=turnradHav(n-1)+d;

        V_anemRelHav(n-1) = VgpsHav(n-1)*anemTurnHav(n-
1)/(turnradHav(n-1));

```

```

theta=atan(f/d);
C=sqrt((f^2)+(d^2));
x=d*cos(theta*pi/180);
q=C-x;
q=q/((separation*0.3048)+q);
%q=sqrt(((V_anemRelHav(n-1)*t)^2)+(dforward^2));
anemTurnHav(n-1)=anemTurnHav(n-1)+q;
anemRelHav(n-1)=HaversineCourse(n-1)+(asin(f/anemTurnHav(n-
1))*180/pi); % adding course difference of weather station to course of
GNSS
V_anemRelHav(n-1)= VgpsHav(n-1)*anemTurnHav(n-1)/(turnradHav(n-
1));

else
anglediffHav(n-1)=abs(anglediffHav(n-1));
%turnrad(n-1)=DistSOG(n-1)/(2*tan(anglediff(n-1)*pi/180/2)); %
Straight line approx XXX
turnradHav(n-1)=DistHav(n-1)*360/(2*pi*anglediffHav(n-1)); %
GNSS Turn Radius using Arc length
%anemTurnHav(n-1)= sqrt((turnradHav(n-1)-d)^2 + dforward^2); %
TURN RADIUS AT ANEMOMETER
%% d/SEPARATION DISTANCE (20m)
%anemTurnHav(n-1)= (turnradHav(n-1)-(d/(20*0.3048))); % TURN
RADIUS AT ANEMOMETER
%anemTurnHav(n-1)= (turnradHav(n-1)+((d/2)/(turnrad(n-
1)+(d/2)))); % TURN RADIUS AT ANEMOMETER
%anemTurnHav(n-1)=turnradHav(n-1)-d;
anemTurnHav(n-1)= (turnradHav(n-1)-
((d/2)/((separation*0.3048)+(d/2))));

%V_anemRelHav(n-1) = VgpsHav(n-1)/(anemTurnHav(n-
1)/(turnradHav(n-1)));
V_anemRelHav(n-1) = VgpsHav(n-1)*(anemTurnHav(n-
1)/(turnradHav(n-1)));

theta=atan(f/d);
C=sqrt((f^2)+(d^2));
x=d*cos(theta*pi/180);
q=C-x;
q=q/((separation*0.3048)+q);
%q=sqrt(((V_anemRelHav(n-1)*t)^2)+(dforward^2));
anemTurnHav(n-1)=anemTurnHav(n-1)-q;
anemRelHav(n-1)=HaversineCourse(n)-(asin(f/anemTurnHav(n-
1))*180/pi); % Course calc
V_anemRelHav(n-1)= VgpsHav(n-1)*anemTurnHav(n-1)/(turnradHav(n-
1));

end
end
anemRelHav=real(anemRelHav);
%% Plots: FIGURE 6 IS BEARING
% Plot Haversine vs "Speed over ground"
aplot=0;
if aplot==1
sa=1:numel(turnrad);

```



```

scatter(sa,turnrad, '.')
    %% DIRECTION:
sample=1:numel(coursegnd);
sample2=1:numel(HaversineCourse);
scatter(sample,coursegnd, '.')
hold on
scatter(sample2,HaversineCourse, '.')
hold on
scatter(sample2,anemCOG, '.')    % SOG
hold on
sample5=1:numel(courseAnem);
scatter(sample5,courseAnem, '.') % HAV at transformed point

%% VELOCITY

sample2=1:numel(SuccessiveVelocity);
sample3=1:numel(V_anemRelHav);
%% ***
figure(2)
sample=1:numel(V_anemTransSOG);
scatter(sample,V_anemTransSOG, '.')
hold on
sample4=1:numel(velocitygnd);
scatter(sample4,velocitygnd, '.')
ylim([0 4])

sample3=1:numel(SuccessiveVelocity);
scatter(sample3,SuccessiveVelocity, '.')
hold on
%% Original GNSS HAV vs HAVREL
sample=1:numel(HaversineVelocity);
scatter(sample,HaversineVelocity, '.')
hold on
samplev=1:numel(SuccessiveVelocity);
scatter(samplev,SuccessiveVelocity, '.');
hold on
sample2=1:numel(V_anemRelHav);
scatter(sample2,V_anemRelHav, '.')
ylim([1 2])

scatter

figure(1)
scatter(HaversineVelocity(1:numel(HaversineVelocity)),velocitygnd(2:numel(velocitygnd)), '.')
xlabel('Haversine Calculated Velocity (knots)')
ylabel('GNSS Output Velocity (m/s)')
correlate=corr(HaversineVelocity(:),velocitygnd(:)); % There is good
correlation, but inaccuracy

% Distribution of error for GNSS Velocities:
figure(2)
velocitygnd2=velocitygnd;
velocitygnd2(1)=[];
diff=HaversineVelocity-velocitygnd2;
diff=abs(diff);
histogram(diff, 'Normalization', 'probability')

```

```

xlabel('m/s')
ylabel('Probability')
title('Difference In GNSS Output "Speed over ground" and Haversine
Calculation')

% Comparing Velocity Calculations over time:
figure(3)
samples=1:numel(HaversineVelocity);
sample2=1:numel(velocitygnd);
scatter(samples,HaversineVelocity,'.')
hold on
scatter(sample2,velocitygnd,','r')
legend('Haversine calculated velocity','NMEA "Speed Over Ground"')

% Comparing Anemometer velocity calculated using "Speed over ground"
with
% GNSS S.O.G
figure(4)
samples=1:numel(HaversineVelocity);
sample2=1:numel(V_anemTransSOG);
sample3=1:numel(velocitygnd);
scatter(sample2,V_anemTransSOG,','') % SPEED OVER GROUND TRANSFORMED TO
ANEMOMETER
hold on
%scatter(samples,HaversineVelocity,','') % HAVERSINE Not transformed
%hold on
scatter(sample3,velocitygnd,','') % Speed over Ground Not Transformed
%hold on
%interpsample(numel(interpsample))=[];
%scatter(interpsample,V_aneminterp,','')
legend('Transformed Anemometer from "S.O.G"', 'NMEA "Speed Over
Ground"')

% Distance comparison:
figure(5)
distanceHav=distance*1000;
scatter(samples,distanceHav,','')
hold on
scatter(sample2,DistSOG,','')
legend('Haversine Distance','Speed Over Ground Distance')

% Bearing (course) comparison:
figure(6)
samples=1:numel(HaversineVelocity);
coursegndcomp=coursegnd;
coursegndcomp(numel(coursegnd))=[];

scatter(samples,HaversineCourse,','')
hold on
samples(1)=[];
scatter(samples,courseAnem,','')
hold on
scatter(samples,anemCOG(2:numel(anemCOG)),','')
%%

scatter(samples,coursegndcomp,','')

```

```

% Weather station Haversine vel vs S.O.G
sample=1:numel(SucessiveVelocity)
sample2=1:numel(V_anemTransSOG)
figure(7)
scatter(sample,SucessiveVelocity, '.')
hold on
scatter(sample2,V_anemTransSOG, '.')
xlabel('Sample Number')
ylabel('Velocity (m/s)')
legend('Haversine Velocity Anemometer Position','Speed Over Ground"
Anemometer Position')
hold off
else
end

end

function
[HaversineVelocityAnemometer,COGANem]=old(Latdegree,Longdegree,d,f,r_ea
rth,t)
Latdegree=Latdegree*pi/180;
Longdegree=Longdegree*pi/180;
angulardistance=sqrt((d^2)+(f^2))/r_earth;

for n=2:numel(Latdegree)-1
HaverGps2AnemBearing(n-1)=mod(((HaversineCourse(n-
1)+HaversineCourse(n))/2)+90+offsetAngle,360)*pi/180;
%HaverGps2AnemBearing(n-1)=mod(HaversineCourse(n-1)+90,360)*pi/180;
Lattrans(n-
1)=asin((sin(Latdegree(n))*cos(angulardistance))+cos(Latdegree(n))*sin
(angulardistance)*cos(HaverGps2AnemBearing(n-1)));
%Longtrans(n)=Longdegree(n)+
atan2(sin(HaverGps2AnemBearing(n))*sin(angulardistance)*cos(Latdegree(n)
),cos(angulardistance)-sin(Latdegree(n))*sin(Lattrans(n)));
Longtrans(n-1)=mod((Longdegree(n)-asin(sin(HaverGps2AnemBearing(n-
1))*sin(angulardistance)/cos(Latdegree(n))+pi),2*pi)-pi;
% Convert back to degrees
Lattrans(n-1)=Lattrans(n-1)*180/pi;
Longtrans(n-1)=Longtrans(n-1)*180/pi;
end

%% Vincenty distance between original and spherical transform
for k=1:numel(Lattrans)
[dist2(k),havveCourse1(k)] =
vdist(Lat4Vince(k),Long4Vince(k),Lattrans(k),Longtrans(k));
end
diff=abs(dist1-dist2);
comptit=[{'Spherical'},{'WGS-84'}];
comp=[dist2',dist1'];
xlswrite('SphereVsWGS84.xlsx',comptit,1,'A1')
xlswrite('SphereVsWGS84.xlsx',comp,1,'A2')
%% Calculating Haversine Distance/Velocity/Heading(bearing) for
Transformed data:
% Recreating String format if needed: UNUSED!!!!!!!!!!!!!!!!!!!!!!
% Separating decimal for conversion from deg to deg,minutes
for n=1:numel(Lattrans) % LATITUDE
if Lattrans(n)>0

```

```

        latint(n)=floor(Lattrans(n));
        fractlat(n)=Lattrans(n)-latint(n);
        cardinalLAT{n}='N';
    else
        latint(n)=floor(Lattrans(n));
        fractlat(n)=latint(n)-ceil(Lattrans(n));
        cardinalLAT{n}='S';
    end
end
fractlat=fractlat*60; % Convert decimal part back to minutes

for n=1:numel(Lattrans) % LONGITUDE
    if Longtrans(n)>0
        longint(n)=floor(Longtrans(n));
        fractlong(n)=Longtrans(n)-longint(n);
        cardinalLONG{n}='W';
    else
        longint(n)=floor(Longtrans(n));
        fractlong(n)=longint(n)-ceil(Longtrans(n));
        cardinalLONG{n}='E';
    end
end
fractlong=fractlong*60; % Convert decimal part back to minutes
% Convert to string
for n=1:numel(longint)
    latint1{n}=num2str(latint(n)); longint1{n}=num2str(longint(n));
    fractlat1{n}=num2str(fractlat(n)); fractlong1{n}=num2str(fractlong(n));
end
% Concatenate Coordinates
for n=1:numel(fractlat)
    TransformedLat{n}=strcat(latint1{n}, '
', fractlat1{n}, ',', cardinalLAT{n});
    TransformedLat{n}=insertAfter(TransformedLat(n), 2, ' ');
    TransformedLong{n}=strcat(longint1{n}, '
', '0', fractlong1{n}, ',', cardinalLONG{n});
    TransformedLong{n}=insertAfter(TransformedLong(n), 2, ' ');
end
%dummy='dumb';

%% Haversine for Transformed points: INPUT DECIMAL FORM!!!
%*****
***
%% CHECK IF STRING INPUT HAS DIFFERENT RESULTS
for n=2:numel(Lattrans)
    [distanceAnem(n-1), courseAnem(n-1)]=haversine([Lattrans(n-1)
Longtrans(n-1)], [Lattrans(n) Longtrans(n)]);

    % coord1=strcat(TransformedLat{n-1}, ',', TransformedLong{n-1});
    % coord2=strcat(TransformedLat{n}, ',', TransformedLong{n});
    % [distanceAnem2(n-1), courseAnem2(n-
1)]=haversine(coord1{1}, coord2{1});
end
%% Haversine Velocity at ANEMOMETER LOCATION:
courseAnem=(courseAnem*-1)+360;
HaversineVelocityAnemometer=distanceAnem./t; %km/s
HaversineVelocityAnemometer=HaversineVelocityAnemometer*1000; %m/s
% %velocity=velocity.*1943.84; % knots

```

```

% HaversineVelocityAnemometer=HaversineVelocityAnemometer';
scatter(Longtrans,Lattrans, '.')
hold on
scatter(Long4Vince,Lat4Vince, '.')
figure(2)
scatter(HaversineVelocityAnemometer,SucessiveVelocity, '.')
end

```

Comp2Card

```

% Comp2card
% Austin Weiss
% Converts X and Y components into 360 degree heading

function dir = Comp2coord(x,y)

for i=1: numel(y)
    if x(i) >=0 & y(i)>=0
        %dir(i)=(180/pi*atan(abs(x(i))/abs(y(i)))));
        dir(i)=180+(180/pi*atan(abs(x(i))/abs(y(i)))));
    elseif x(i)>0 & y(i)<0
        %dir(i)=90+(180/pi*atan(abs(y(i))/abs(x(i)))));
        dir(i)=270+(180/pi*atan(abs(y(i))/abs(x(i)))));
    elseif x(i)<0 & y(i)<0
        %dir(i)=270-(180/pi*atan(abs(y(i))/abs(x(i)))));
        dir(i)=180/pi*atan(abs(x(i))/abs(y(i)));
    else % x<0, y>0
        %dir(i)=270+(180/pi*atan(abs(y(i))/abs(x(i)))));
        dir(i)=90+(180/pi*atan(abs(y(i))/abs(x(i)))));
    end
end
end

```

Haversine Function:

```

function [km, bearing,nmi, mi] = haversine(loc1, loc2)
% HAVERSINE      Compute distance between locations using Haversine
formula
% KM = HAVERSINE(LOC1, LOC2) returns the distance KM in km between
% locations LOC1 and LOC2 using the Haversine formula. LOC1 and LOC2
are
% latitude and longitude coordinates that can be expressed as either
% strings representing degrees, minutes, and seconds (suffixed with
% N/S/E/W), or numeric arrays representing decimal degrees (where
% negative indicates West/South).
%
% [KM, NMI, MI] = HAVERSINE(LOC1, LOC2) returns the computed distance
in
% kilometers (KM), nautical miles (NMI), and miles (MI).
%
% Examples

```

```

%     haversine('53 08 50N, 001 50 58W', '52 12 16N, 000 08 26E')
returns
%     170.2547
%     haversine([53.1472 -1.8494], '52 12.16N, 000 08.26E') returns
%     170.2508
%     haversine([53.1472 -1.8494], [52.2044 0.1406]) returns 170.2563
%
% Inputs
%     LOC must be either a string specifying the location in degrees,
%     minutes and seconds, or a 2-valued numeric array specifying the
%     location in decimal degrees.  If providing a string, the
latitude
%     and longitude must be separated by a comma.
%
%     The first element indicates the latitude while the second is
the
%     longitude.
%
% Notes
%     The Haversine formula is used to calculate the great-circle
%     distance between two points, which is the shortest distance
over
%     the earth's surface.
%
%     This program was created using equations found on the website
%     http://www.movable-type.co.uk/scripts/latlong.html
% Created by Josiah Renfree
% May 27, 2010
%% MODIFIED by Austin Weiss
% 12/21/2018
% Now includes initial and final bearing (set to final)
% Now takes input as string Latitude,Longitude:
%"degrees minutes.decimalminutes,degrees minutes.decimal minutes"
%% Check user inputs
% If two inputs are given, display error
if ~isequal(nargin, 2)
    error('User must supply two location inputs')

% If two inputs are given, handle data
else

    locs = {loc1 loc2};      % Combine inputs to make checking easier

% Cycle through to check both inputs
for i = 1:length(locs)

    % Check inputs and convert to decimal if needed
    if ischar(locs{i})

        % Parse lat and long info from current input
        temp = regexp(locs{i}, ',', 'split');
        lat = temp{1}; lon = temp{2};
        clear temp
        locs{i} = [];      % Remove string to make room for
array

        % Obtain degrees, minutes, seconds, and hemisphere

```

```

temp = regexp(lat, '(\d+)\D+(\d+)\D+(\d+) (\w?)', 'tokens');
temp = temp{1};
num=numel(num2str(temp{3})); % number of digits in decimal,
to remake decimal to add back
% Calculate latitude in decimal degrees
locs{i}(1) = str2double(temp{1}) +
str2double(temp{2})/60+str2double(temp{3})/(10^num)/60;

% Make sure hemisphere was given
if isempty(temp{4})
    error('No hemisphere given')
% If latitude is south, make decimal negative
elseif strcmpi(temp{4}, 'S')
    locs{i}(1) = -locs{i}(1);
end

clear temp
% Obtain degrees, minutes, seconds, and hemisphere
temp = regexp(lon, '(\d+)\D+(\d+)\D+(\d+) (\w?)', 'tokens');
temp = temp{1};
num=numel(num2str(temp{3})); % number of digits in decimal,
to remake decimal to add back
% Calculate longitude in decimal degrees
locs{i}(2) = str2double(temp{1}) +
str2double(temp{2})/60+str2double(temp{3})/(10^num)/60;

% Make sure hemisphere was given
if isempty(temp{4})
    error('No hemisphere given')

% If longitude is west, make decimal negative
elseif strcmpi(temp{4}, 'W')
    locs{i}(2) = -locs{i}(2);
end

clear temp lat lon
end
end
end
% Check that both cells are a 2-valued array
if any(cellfun(@(x) ~isequal(length(x),2), locs))
    error('Incorrect number of input coordinates')
end
% Convert all decimal degrees to radians
locs = cellfun(@(x) x .* pi./180, locs, 'UniformOutput', 0);
%% Begin Distance calculation
R = 6371; % Earth's radius in km
delta_lat = locs{2}(1) - locs{1}(1); % difference in latitude
delta_lon = locs{2}(2) - locs{1}(2); % difference in longitude
a = sin(delta_lat/2)^2 + cos(locs{1}(1)) * cos(locs{2}(1)) * ...
    sin(delta_lon/2)^2;
c = 2 * atan2(sqrt(a), sqrt(1-a));
km = R * c; % distance in km
%% Convert result to nautical miles and miles
nmi = km * 0.539956803; % nautical miles
mi = km * 0.621371192; % miles
%% INITIAL Bearing Calculation:

```

```

b1=sin(delta_lon)*cos(locs{2}(1));
b2=(cos(locs{1}(1))*sin(locs{2}(1)))-
(sin(locs{1}(1))*cos(locs{2}(1))*cos(delta_lon));
bearing1=atan2(b2,b1);
bearing1=bearing1*180/pi; % convert to degrees
% Converting to 360 degree format
bearing1=bearing1+180;
    if bearing1>=360
        bearing1=bearing1-360;
    else
    end
% bearing1=mod(bearing1-90,360);
%% FINAL Bearing calculation
final=1;
if final==1
bearing2=atan2(b1,b2);
bearing2=bearing2*180/pi; % Convert to degrees
bearing2=mod(bearing2,360);

%for n=1:numel(bearing2)
    %if bearing2>=360
    %    bearing2=bearing2-360;
    %else
    %end
%end
bearing=bearing2;
else
    bearing=bearing1;
end
%bearing=(bearing1+bearing2)/2;
end

```

Vincenty Distance Function:

```

function [s,a21] = vdist(lat1,lon1,lat2,lon2)
% VDIST - compute distance between points on the WGS-84 ellipsoidal
Earth
%         to within a few millimeters of accuracy using Vincenty's
algorithm
%
% s = vdist(lat1,lon1,lat2,lon2)
%
% s = distance in meters
% lat1 = GEODETIC latitude of first point (degrees)
% lon1 = longitude of first point (degrees)
% lat2, lon2 = second point (degrees)
%
% Original algorithm source:
% T. Vincenty, "Direct and Inverse Solutions of Geodesics on the
Ellipsoid
% with Application of Nested Equations", Survey Review, vol. 23, no.
176,
% April 1975, pp 88-93

```



```

%
% Notes: (1) Error correcting code, convergence failure traps,
antipodal corrections,
%         polar error corrections, WGS84 ellipsoid parameters,
testing, and comments
%         written by Michael Kleder, 2004.
%         (2) Vincenty describes his original algorithm as precise to
within
%         0.01 millimeters, subject to the ellipsoidal model.
%         (3) Essentially antipodal points are treated as exactly
antipodal,
%         potentially reducing accuracy by a small amount.
%         (4) Failures for points exactly at the poles are eliminated by
%         moving the points by 0.6 millimeters.
%         (5) Vincenty's azimuth formulas are not implemented in this
%         version, but are available as comments in the code.
%         (6) The Vincenty procedure was transcribed verbatim by Peter
Cederholm,
%         August 12, 2003. It was modified and translated to English
by Michael Kleder.
%         Mr. Cederholm's website is http://www.plan.aau.dk/~pce/
%         (7) Code to test the disagreement between this algorithm and
the
%         Mapping Toolbox spherical earth distance function is
provided
%         as comments in the code. The maximum differences are:
%         Max absolute difference: 38 kilometers
%         Max fractional difference: 0.56 percent

% Input check:
if abs(lat1)>90 | abs(lat2)>90
    error('Input latitudes must be between -90 and 90 degrees,
inclusive.')

```

```

%%
if L > pi
    L = 2*pi - L;
end
lambda = L;
lambdaold = 0;
itercount = 0;
while ~itercount | abs(lambda-lambdaold) > 1e-12 % force at least one
execution
    itercount = itercount+1;
    if itercount > 50
        warning('Points are essentially antipodal. Precision may be
reduced slightly.');
```

$$\lambda = \pi;$$

```

        break
    end
    lambdaold = lambda;
    sinsigma = sqrt((cos(U2)*sin(lambda))^2+(cos(U1)*...
        sin(U2)-sin(U1)*cos(U2)*cos(lambda))^2);
    cossigma = sin(U1)*sin(U2)+cos(U1)*cos(U2)*cos(lambda);
    sigma = atan2(sinsigma,cossigma);
    alpha = asin(cos(U1)*cos(U2)*sin(lambda)/sin(sigma));
    cos2sigmam = cos(sigma)-2*sin(U1)*sin(U2)/cos(alpha)^2;
    C = f/16*cos(alpha)^2*(4+f*(4-3*cos(alpha)^2));
    lambda = L+(1-C)*f*sin(alpha)*(sigma+C*sin(sigma)*...
        (cos2sigmam+C*cos(sigma)*(-1+2*cos2sigmam^2)));
    % correct for convergence failure in the case of essentially
antipodal points
    if lambda > pi
        warning('Points are essentially antipodal. Precision may be
reduced slightly.');
```

$$\lambda = \pi;$$

```

        break
    end
end
end
u2 = cos(alpha)^2*(a^2-b^2)/b^2;
A = 1+u2/16384*(4096+u2*(-768+u2*(320-175*u2)));
B = u2/1024*(256+u2*(-128+u2*(74-47*u2)));
deltastigma = B*sin(sigma)*(cos2sigmam+B/4*(cos(sigma)*(-
1+2*cos2sigmam^2)...
    -B/6*cos2sigmam*(-3+4*sin(sigma)^2)*(-3+4*cos2sigmam^2)));
s = b*A*(sigma-deltastigma);

% %
=====
% % Vicenty's azimuth calculation code is left unused:
% % (results in radians)
% % From point #1 to point #2

a12 = atan2(cos(U2)*sin(lambda),cos(U1)*sin(U2)-
sin(U1)*cos(U2)*cos(lambda));
if a12 < 0
    a12 = a12+2*pi;
end
% % from point #2 to point #1

```

```

a21 = atan2(cos(U1)*sin(lambda),-
sin(U1)*cos(U2)+cos(U1)*sin(U2)*cos(lambda));
if a21 < 0
    a21 = a21+pi;
end
if (L>0) & (L<pi)
    a21 = a21 + pi;
end

% %
=====
% % Code to test the Mapping Toolbox spherical earth distance against
% % Vincenty's algorithm using random test points:
% format short g
% errmax=0;
% abserrmax=0;
% for i = 1:10000
%     llat = rand * 184-92;
%     tlat = rand * 184-92;
%     llon = rand * 364 - 2;
%     tlon = rand * 364 - 2;
%     llat = max(-90,min(90,llat)); % round to include occasional exact
poles
%     tlat = max(-90,min(90,tlat));
%     llon = max(0,min(360,llon));
%     tlon = max(0,min(360,tlon));
%     % randomly test exact equator
%     if rand < .01
%         llat = 0;
%         llon = 0;
%     else
%         if rand < .01
%             llat = 0;
%         end
%         if rand < .01
%             tlat = 0;
%         end
%     end
%     dm = 1000*deg2km(distance(llat,llon,tlat,tlon));
%     dv = vdist(llat,llon,tlat,tlon);
%     abserr = abs(dm-dv);
%     if abserr < 1e-2 % disagreement less than a centimeter
%         err = 0;
%     else
%         err = abs(dv-dm)/dv;
%     end
%     errmax = max(err,errmax);
%     abserrmax = max(abserr,abserrmax);
%     %     if i==1 | rand > .99
%     disp([i dv dm err errmax abserrmax])
%     %     end
%     if err > .01
%         break
%     end
% end
% end

```

Vincenty Coordinate Calculation Function:

```
function [lat2,lon2,a21] = vreckon(lat1,lon1,s,a12)
% RECKON - Using the WGS-84 Earth ellipsoid, travel a given distance
along
%         a given azimuth starting at a given initial point, and
return
%         the endpoint within a few millimeters of accuracy, using
%         Vincenty's algorithm.
%
% USAGE:
% [lat2,lon2] = vreckon(lat1, lon1, s, a12)
%
% VARIABLES:
% lat1 = initial latitude (degrees)
% lon1 = initial longitude (degrees)
% s     = distance (meters)
% a12  = initial azimuth (degrees)
% lat2, lon2 = second point (degrees)
% a21  = reverse azimuth (degrees), at final point facing back toward
the
%       initial point
%
% Original algorithm source:
% T. Vincenty, "Direct and Inverse Solutions of Geodesics on the
Ellipsoid
% with Application of Nested Equations", Survey Review, vol. 23, no.
176,
% April 1975, pp 88-93.
% Available at: http://www.ngs.noaa.gov/PUBS\_LIB/inverse.pdf
%
% Notes:
% (1) The Vincenty reckoning algorithm was transcribed verbatim into
%     JavaScript by Chris Veness. It was modified and translated to
Matlab
%     by Michael Kleder. Mr. Veness's website is:
%     http://www.movable-type.co.uk/scripts/latlong-vincenty-
direct.html
% (2) Error correcting code, polar error corrections, WGS84 ellipsoid
%     parameters, testing, and comments by Michael Kleder.
% (3) By convention, when starting at a pole, the longitude of the
initial
%     point (otherwise meaningless) determines the longitude line along
%     which to traverse, and hence the longitude of the final point.
% (4) The convention noted in (3) above creates a discrepancy with
VDIST
%     when the the initial or final point is at a pole. In the VDIST
%     function, when traversing from a pole, the azimuth is 0 when
%     heading away from the south pole and 180 when heading away from
the
%     north pole. In contrast, this VRECKON function uses the azimuth
as
%     noted in (3) above when traversing away from a pole.
% (5) In testing, where the traversal subtends no more than 178
degrees,
%     this function correctly inverts the VDIST function to within 0.2
```

```

%      millimeters of distance, 5e-10 degrees of forward azimuth,
%      and 5e-10 degrees of reverse azimuth. Precision reduces as test
%      points approach antipodal because the precision of VDIST is
reduced
%      for nearly antipodal points. (A warning is given by VDIST.)
% (6) Tested but no warranty. Use at your own risk.
% (7) Ver 1.0, Michael Kleder, November 2007

% Input check:
if abs(lat1)>90
    error('Input latitude must be between -90 and 90 degrees,
inclusive.')
```

```

end
a = 6378137; % semimajor axis
%b = 6356752.314140347;
    b = 6356752.31424518; % semiminor axis ORIGINAL
f = 1/298.257223563; % flattening coefficient WGS-84 ORIGINAL
%f=1/298.257222100882711243;
lat1 = lat1 * .1745329251994329577e-1; % intial latitude in radians
lon1 = lon1 * .1745329251994329577e-1; % intial longitude in radians
% correct for errors at exact poles by adjusting 0.6 millimeters:
kidx = abs(pi/2-abs(lat1)) < 1e-10;
if any(kidx);
    lat1(kidx) = sign(lat1(kidx))*(pi/2-(1e-10));
end
alpha1 = a12 * .1745329251994329577e-1; % inital azimuth in radians
sinAlpha1 = sin(alpha1);
cosAlpha1 = cos(alpha1);
tanU1 = (1-f) * tan(lat1);
cosU1 = 1 / sqrt(1 + tanU1*tanU1);
sinU1 = tanU1*cosU1;
sigma1 = atan2(tanU1, cosAlpha1);
sinAlpha = cosU1 * sinAlpha1;
cosSqAlpha = 1 - sinAlpha*sinAlpha;
uSq = cosSqAlpha * (a*a - b*b) / (b*b);
A = 1 + uSq/16384*(4096+uSq*(-768+uSq*(320-175*uSq)));
%k1=((sqrt(1+uSq)-1)/(sqrt(1+uSq)+1));
%A = (1+(.25*k1^2));
%B = k1*(1-(3/8)*k1^2);
B = uSq/1024 * (256+uSq*(-128+uSq*(74-47*uSq)));
sigma = s / (b*A);
sigmaP = 2*pi;
while (abs(sigma-sigmaP) > 1e-12)
    cos2SigmaM = cos(2*sigma1 + sigma);
    sinSigma = sin(sigma);
    cosSigma = cos(sigma);
    deltaSigma = B*sinSigma*(cos2SigmaM+B/4*(cosSigma*(-1+...
        2*cos2SigmaM*cos2SigmaM)-...
        B/6*cos2SigmaM*(-3+4*sinSigma*sinSigma)*(-3+...
        4*cos2SigmaM*cos2SigmaM));
    sigmaP = sigma;
    sigma = s / (b*A) + deltaSigma;
end
tmp = sinU1*sinSigma - cosU1*cosSigma*cosAlpha1;
lat2 = atan2(sinU1*cosSigma + cosU1*sinSigma*cosAlpha1,...
    (1-f)*sqrt(sinAlpha*sinAlpha + tmp*tmp));
lambda = atan2(sinSigma*sinAlpha1, cosU1*cosSigma - ...
```

```

    sinU1*sinSigma*cosAlpha1);
C = f/16*cosSqAlpha*(4+f*(4-3*cosSqAlpha));
L = lambda - (1-C) * f * sinAlpha * (sigma + C*sinSigma*(cos2SigmaM+...
    C*cosSigma*(-1+2*cos2SigmaM*cos2SigmaM)));
lon2 = lon1 + L;
% output degrees
lat2 = lat2 * 57.295779513082322865;
lon2 = lon2 * 57.295779513082322865;
lon2 = mod(lon2,360); % follow [0,360] convention
if nargout > 2
    a21 = atan2(sinAlpha, -tmp);
    a21 = 180 + a21 * 57.295779513082322865; % note direction
reversal
    a21=mod(a21,360);
end
return

```

Appendix 4. R Script

```
getwd()
#setwd("C:/Users/Weiss/OneDrive - University of Kentucky/1 Thesis Work/Correlation
Between Stationary Ultrasonic Anemometers/Feb 13th Test")
setwd("C:/Users/Weiss/Desktop/Dynamic result")
require(xlsx)
require(gstat)
require(sp)
require(lattice)
require(gstat)
data<-read.xlsx("Dynamic_1_20ft_3mph_NoFilter.xlsx", "Plot")
a<- summary(data)
head(data)

# 1st Order Probability (mean)
mean(data$X86000.Velocities..m.s.)
# 2nd Order Probability (variance)
sort(var(data$X86000.Velocities..m.s.))
library(moments)
skewness(data$X86000.Velocities..m.s., na.rm = FALSE)
kurtosis(data$X86000.Velocities..m.s., na.rm = FALSE)
# 86000 QQ
qqnorm(data$X86000.Velocities..m.s., main = "Normal Q-Q Plot - Yield")
qqline(data$X86000.Velocities..m.s., col='red')
# 92000 QQ
qqnorm(data$X92000.Velocity.M1..m.s., main = "Normal Q-Q Plot - Yield")
qqline(data$X92000.Velocity.M1..m.s., col='red')
pearcorr <- cor(data) # Calculates Pearson correlation coefficients
#Spearman not appropriate
```

```

#spearcorr <- cor(data, method = "spearman") # Calculates Spearman correlation
coefficients

#

## 86000 Autocorr

lag=50 # Change for lag distance

acf1 <- acf(data$X86000.Velocities..m.s., lag=lag) # Calc autocorr function for yield at
lag distance 16

acf1

acf1.d<-data.frame(acf1$lag, acf1$acf) # Reorganize (lag distance, autocorr function)

acf1.d

plot(acf1.d, type="p",main="86000 Velocities", xlab="Lag (h)", ylab="Correlation
Coefficient", ylim=c(0,1), xlim=c(0,lag))

#write.csv(acf1.d,"acf1.d.csv")

#

## 92000 Autocorr

lag=50 # Change for lag distance

#acf1 <- acf(data$X92000.Velocity..m.s., lag=lag) # Calc autocorr function for yield at
lag distance 16

acf1 <- acf(data$X92000.Velocity.M1..m.s., lag=lag) # Calc autocorr function for yield
at lag distance 16

acf1

acf1.d<-data.frame(acf1$lag, acf1$acf) # Reorganize (lag distance, autocorr function)

acf1.d

plot(acf1.d, type="p",main="92000 Velocities", xlab="Lag (h)", ylab="Correlation
Coefficient", ylim=c(0,1), xlim=c(0,lag))

## FAST PROCESS CROSS CORR: CHANGE FILE READ AND SAVE AND RUN
TO END

setwd("C:/Users/Weiss/Desktop/Feb 13th Test")

#setwd("C:/Users/Weiss/Desktop/Dynamic result")

#data<-read.xlsx("Dynamic_3_40ft_3mph_NoFilter.xlsx", "Plot")

```



```

data<-read.xlsx("Stationary_20ft_1_NoFilter.xlsx", "Plot")
#### Cross-Corrleation
lag=50
ccf1<-ccf(data$X86000.Velocities..m.s.,data$X92000.Velocity.M1..m.s.,
lag=lag,na.action = na.pass)
#ccf1<-ccf(data$X86000.Velocities..m.s.,data$X92.Velocity.M2..m.s., lag=lag,na.action
= na.pass)
#ccf1<-ccf(data$X86000.Velocities..m.s.,data$X92.Velocity.M3.m.s., lag=lag,na.action
= na.pass)

plot(ccf1, type="b",main="86000 vs 92000 Velocities", xlab="Lag (h)", ylab="Cross
Correlation Coefficient", ylim=c(0,.7), xlim=c(-lag,lag))

ccf1line<-data.frame((-lag:lag), ccf1$acf)
lines(smooth.spline(ccf1line, spar=0.1), col="blue")
ccf1.print <- data.frame(ccf1$lag,ccf1$acf) # Reorganize data in columns
ccf1.print

# Printing Cross Corr
setwd("C:/Users/Weiss/Desktop/Dynamic result/cross corr")
write.xlsx(ccf1.print,"CrossCorr_3_20ft_3mph_NoFilter_M2.xlsx")

```

Appendix 5. Data Logging Code (VB.NET)

Form1.vb

```
Public Class Form1
    Private WithEvents CommPort As New RS232
    Private WithEvents CommPort2 As New RS232
    Private Sample As Integer = 1
    Private TriggerThread As System.Threading.Thread
    Private ElapsedTime As New Stopwatch
    Private LogFileName As String

    Dim myPort As Array
    Dim DataCollect As Array
    'Private WithEvents Printer As New Printcsv
    Public Event ListBuild()
    Public inputs As String
    Public PrintList As New List(Of String)
    Public PrintList2 As New List(Of String)
    Public WriteTrigger = 1

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Form1_Load is the main program!!
        CheckForIllegalCrossThreadCalls = False
        BaudRate.Items.Add(38400)
        For Each PORT In CommPort.GetComPortNames
            ModelPortName.Items.Add(PORT)
            ResponseOnePort.Items.Add(PORT)
        Next
        ModelPortName.Text = ModelPortName.Items.Item(0) 'Sets Port List to first
option so text appears on startup
        ResponseOnePort.Text = ResponseOnePort.Items.Item(0) 'Sets Port List to
first option so text appears on startup
        BaudRate.Text = BaudRate.Items.Item(0)
        CloseButton.Enabled = False 'Disables Close Button since there is no Open
'Dim newTread As New System.Threading.Thread(AddressOf MessageReceived)
'Dim thread As New Thread(AddressOf MyBackgroundPrinter) 'Starting
background printer thread
        Thread.Start()

    End Sub

    'Start CONNECT BUTTON'
    Private Sub OpenButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles OpenButton.Click
        CommPort.OpenPort(ModelPortName.Text, 38400, 8, "N", 1) 'Opens the Port
CommPort2.OpenPort(ResponseOnePort.Text, 38400, 8, "N", 1) 'Opens other
port
        OpenButton.Enabled = False
        CloseButton.Enabled = True
    End Sub

    Public Sub MessageReceived() Handles CommPort.NewMessage
```

```

DataPrint.Text = CommPort.GetMessage 'Grabs Message and Time
Dim Message As String = CommPort.GetMessage

'DataPrint.Text = Message
Dim Items As String() = Split(Message, " ")
SensorAddress.Text = "86000"
'SensorAddress.Text = Message
WindSpeed.Text = Items(1)
WindDirection.Text = Items(2)
StatusCodeChecksum.Text = Items(3)
'OLD METHOD: Write time into text box
'Time1.Text = CommPort.Time
Time1.Text = Items(4) ' Grabs timestamp from GPS

'Writes Data When Log Button is Pressed
If (LogData.Text = "Stop Logging") Then
    ' Grab text box data and create string:
    'Dim inputs As String = "86000" & "," & WindSpeed.Text & "," &
WindDirection.Text & "," & "," & "," & "," & "," & StatusCodeChecksum.Text & "," &
Time1.Text
    inputs = "86000" & "," & Items(1) & "," & Items(2) & "," & "," & "," &
"," & Items(3) & "," & Items(4)
    ' Call file writer with string to write as input
    'BackgroundWorker1.RunWorkerAsync(inputs)

    'Dim Printcheck = Printer.List4write(inputs)

    List4write(inputs)
    'File.AppendAllText(Path.Text, "86000" & "," & WindSpeed.Text & "," &
WindDirection.Text & "," & "," & "," & "," & "," & StatusCodeChecksum.Text & "," &
Time1.Text & vbCrLf)
Else
    LogData.Text = "Log Data"
    LogData.Enabled = True
    Filename.Enabled = True

End If

End Sub

Public Sub MessageReceived2() Handles CommPort2.NewMessage
DataPrint2.Text = CommPort2.GetMessage 'Grabs Message and Time
Dim MessageOne As String = CommPort2.GetMessage
Dim ItemsOne As String() = Split(MessageOne, " ")
SensorAddress2.Text = "92000"
WindSpeed2.Text = ItemsOne(1)
WindDirection2.Text = ItemsOne(2)
Temperature.Text = ItemsOne(3)
RelativeHumidity.Text = ItemsOne(4)
BarometricPressure.Text = ItemsOne(5)
StatusCodeChecksum2.Text = ItemsOne(6)
'Write time into text box from RS232 class old method of calling computer
timestamping
'Time2.Text = CommPort2.Time
Time2.Text = ItemsOne(7) ' Grabs timestamp from GPS

'Writes Data When Log Button is Pressed

```

```

    If (LogData.Text = "Stop Logging") Then
        ' Grab text box data and create string:
        'Dim inputs As String = "92000" & "," & WindSpeed2.Text & "," &
WindDirection2.Text & "," & Temperature.Text & "," & RelativeHumidity.Text & "," &
BarometricPressure.Text & "," & StatusCodeChecksum2.Text & "," & Time2.Text
        inputs = "92000" & "," & ItemsOne(1) & "," & ItemsOne(2) & "," &
ItemsOne(3) & "," & ItemsOne(4) & "," & ItemsOne(5) & "," & ItemsOne(6) & "," &
ItemsOne(7)

        List4write(inputs)
        'BackgroundWorker2.RunWorkerAsync(inputs) 'Raises DoWork Event

        'File.AppendAllText(Path.Text, "92000" & "," & WindSpeed2.Text & "," &
WindDirection2.Text & "," & Temperature.Text & "," & RelativeHumidity.Text & "," &
BarometricPressure.Text & "," & StatusCodeChecksum2.Text & "," & Time2.Text &
vbCrLf)
    Else
        LogData.Text = "Log Data"
        LogData.Enabled = True
        Filename.Enabled = True
    End If
End Sub

'Private Sub Bittester() Handles CommPort2.Bitgot
'Dim file2 As System.IO.StreamWriter
'File2.WriteAllText(C:\Users\amwe235\Documents\test.txt)
'file2.AppendAllText(SerialPort1.ReadExisting + Time1)
'file2 =
My.Computer.FileSystem.OpenTextFileWriter("C:\Users\amwe235\Documents\test.txt",
True)
'file2.WriteLine(SerialPort1.ReadExisting + CommPort2.Time1)
'End Sub

'Start Disconnect Button'
Private Sub CloseButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CloseButton.Click
    SerialPort1.Close()
    SerialPort2.Close()
    OpenButton.Enabled = True
    CloseButton.Enabled = False
End Sub

'Send Button Start'
Private Sub SendButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SendButton.Click
    SerialPort1.Write(SendInput.Text)
End Sub

Private Sub LogData_Click(sender As Object, e As EventArgs) Handles
LogData.Click

    If LogData.Text = "Log Data" Then
        LogData.Text = "Stop Logging"
        Filename.Enabled = False
        Timer1.Enabled = True

    Else
        LogData.Text = "Log Data"
    End If
End Sub

```

```

        Filename.Enabled = True
        Timer1.Enabled = False
    End If

End Sub

Public Sub Browse_Click(sender As Object, e As EventArgs) Handles Browse.Click

    Dim result As DialogResult = FolderBrowserDialog1.ShowDialog()
    'FolderBrowserDialog1.ShowDialog()
    If (result = DialogResult.OK) Then
        Dim FolderChoice As String = FolderBrowserDialog1.SelectedPath
        Dim Destination As String = String.Concat(FolderChoice, "\")
        Dim Filepath = String.Concat(Destination, Filename.Text)
        Dim FullFilepath = String.Concat(Filepath, ".csv")
        Path.Text = FullFilepath
        File.WriteAllText(FullFilepath, "Sensor Address,Wind Speed (m/s),Wind
Direction (deg),Temperature (deg C),Relative Humidity (%),Barometric Pressure
(hPa),StatusCode*Checksum,Time" & vbCrLf) ' Create File and Write Titles

    ElseIf (result = DialogResult.Cancel) Then
        Return
    End If
End Sub

Public Function List4write(inputs) 'As Task(Of Integer)
    Dim done As Integer
    If WriteTrigger = 1 And PrintList.Count < 20 Then
        PrintList.Add(inputs)
        'Return PrintList
    ElseIf WriteTrigger = 2 And PrintList2.Count < 20 Then
        PrintList2.Add(inputs)
        'Return PrintList2
    ElseIf WriteTrigger = 1 And PrintList.Count >= 20 Then
        If BackgroundPrinter.IsBusy = False Then
            PrintList.Add(inputs)
            BackgroundPrinter.RunWorkerAsync()
            done = 1

        Else
            End If

    ElseIf WriteTrigger = 2 And PrintList2.Count >= 20 Then
        If BackgroundPrinter.IsBusy = False Then
            PrintList2.Add(inputs)
            BackgroundPrinter.RunWorkerAsync()
            done = 1

        Else
            End If
    End If

    Return done
    'Dim dummy = 1

End Function

```

```

Private Sub BackgroundPrinter_DoWork(sender As Object, e As DoWorkEventArgs)
Handles BackgroundPrinter.DoWork
    'While PrintList.Count <> 0
    If WriteTrigger = 1 Then
        WriteTrigger = 2
        File.AppendAllLines(Path.Text, PrintList)

        PrintList.Clear()
    ElseIf WriteTrigger = 2 Then 'And PrintList2.Count > 10
        WriteTrigger = 1
        File.AppendAllLines(Path.Text, PrintList2)

        PrintList2.Clear()
    End If
    'End While
    'Return 1
End Sub

Private Sub BackgroundPrinter_RunWorkerCompleted(sender As Object, e As
RunWorkerCompletedEventArgs) Handles BackgroundPrinter.RunWorkerCompleted
    'Timer1.Enabled = True
End Sub
End Class

```

RS232.vb

```

*****
'*      TITLE: RS232.vb (c)2010                                     *
'*      AUTHOR: Michael P. Sama / Austin Weiss                    *
'*      COMPANY: Biosystems & Agricultural Engineering, Univeristy 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. *
'* Latest Version: 2/15/2018 Modified for timestamping data stream as property *
'*                accessible from outside the class *
*****

```

Public Class RS232

```

Public Event NewMessage()
Private Buffer As String = ""
Private Message As String
Private WithEvents SerialPort1 As New System.IO.Ports.SerialPort
Private LastOutgoingMessage As String = ""
Public PauseSerialInput As Boolean = False
'Public Property Time As String
Public Property Time As String ' Define Time as a property to call
Public TimeTrigger As Boolean = False ' Trigger for timing

```

```

    Public Sub Write(ByVal BytesToWrite() As Byte, ByVal StartIndex As Integer,
ByVal Length As Integer)
        SerialPort1.Write(BytesToWrite, StartIndex, Length)
    End Sub

    Public Sub SendMessage(ByVal OutgoingMessage As String)
        LastOutgoingMessage = OutgoingMessage
        Try
            SerialPort1.Write(OutgoingMessage)
        Catch ex As Exception
            Dim Dummy As Boolean = False
        End Try

    End Sub
    Public Sub ResendMessage()
        Try
            SerialPort1.Write(LastOutgoingMessage)
        Catch ex As Exception
            'error
        End Try

    End Sub

    Public Function GetMessage()
        Return Message
        'Return Time unnecessary because it's a property of the class
    End Function

    Public Function ClosePort()
        If SerialPort1.IsOpen Then
            Try
                SerialPort1.Close()
                Return 1
            Catch ex As Exception
                Return 0
            End Try
        Else
            Return 1
        End If
    End Function

    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
            Return 0
        Else
            Try
                SerialPort1.PortName = PortName
                SerialPort1.BaudRate = BaudRate
                SerialPort1.DataBits = DataBits
                Select Case 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
    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
    SerialPort1.Open()
    SerialPort1.DiscardInBuffer()

    AddHandler SerialPort1.DataReceived, AddressOf
Me.SerialBytesRecieved

    Catch ex As Exception
        Return 0
    End Try
    Return 1
End If

End Function

Public Function IsOpen() As Boolean
    Return SerialPort1.IsOpen()
End Function

Private Sub SerialBytesRecieved(ByVal Sender As Object, ByVal e As
System.IO.Ports.SerialDataReceivedEventArgs)
    'If timestamp is here, it will timestamp everytime bits are recieved
    'Assign the current time to the property "Time"
hours:minutes:seconds:thousandth of second
    'Time trigger check and timing
    If TimeTrigger = True Then
        'Time = DateTime.Now.ToString("HH:mm:ss.fff")
        'TimeTrigger = False
    Else

    End If

    If Not PauseSerialInput Then
        Try
            'ADD TO BUFFER
            AddToBuffer(SerialPort1.ReadExisting)

        Catch ex As Exception

```



```

        Dim dummy As Boolean = False
    End Try
Else
    SerialPort1.DiscardInBuffer()
    Buffer = ""
End If
End Sub

Private Sub AddToBuffer(ByVal characters As String)
    Buffer += characters
    StringSearch()

End Sub

Private Sub StringSearch()

    RemoveHandler SerialPort1.DataReceived, AddressOf Me.SerialBytesRecieved

    Dim First As Integer = -1
    Dim Last As Integer = -1

    Try
        'Defining demlimiter of data sentence separation
        First = Buffer.IndexOf(vbCrLf) ' CHANGED vbCr to
vbCrLf for new GNSS included string

        Last = Buffer.LastIndexOf(vbCrLf)

    Catch ex As Exception
        Dim dummy As Boolean = False
    End Try

    Try
        If (First <> -1 And Last <> -1) And (Last > First) Then
            Message = Buffer.Substring(First, (Last - First))
            Buffer = Buffer.Remove(0, Last - 1)
            If PauseSerialInput Then

                Else

                    ' A Full message is found, get ready to time the next one!
                    'TimeTrigger = True
                    RaiseEvent NewMessage()

                End If

            End If

        Catch ex As Exception
            Dim dummy As Boolean = False
        End Try

        AddHandler SerialPort1.DataReceived, AddressOf Me.SerialBytesRecieved

    End Sub

```

```

Public Function GetComPortNames()
    Dim PortNames As New List(Of String)
    For i As Integer = 0 To (My.Computer.Ports.SerialPortNames.Count - 1)
        PortNames.Add(My.Computer.Ports.SerialPortNames(i))
    Next
    BubbleSort(Of String)(PortNames)
    Return PortNames
End Function

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

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 6. Microcontroller Design and Code:

PCB Design Schematics:

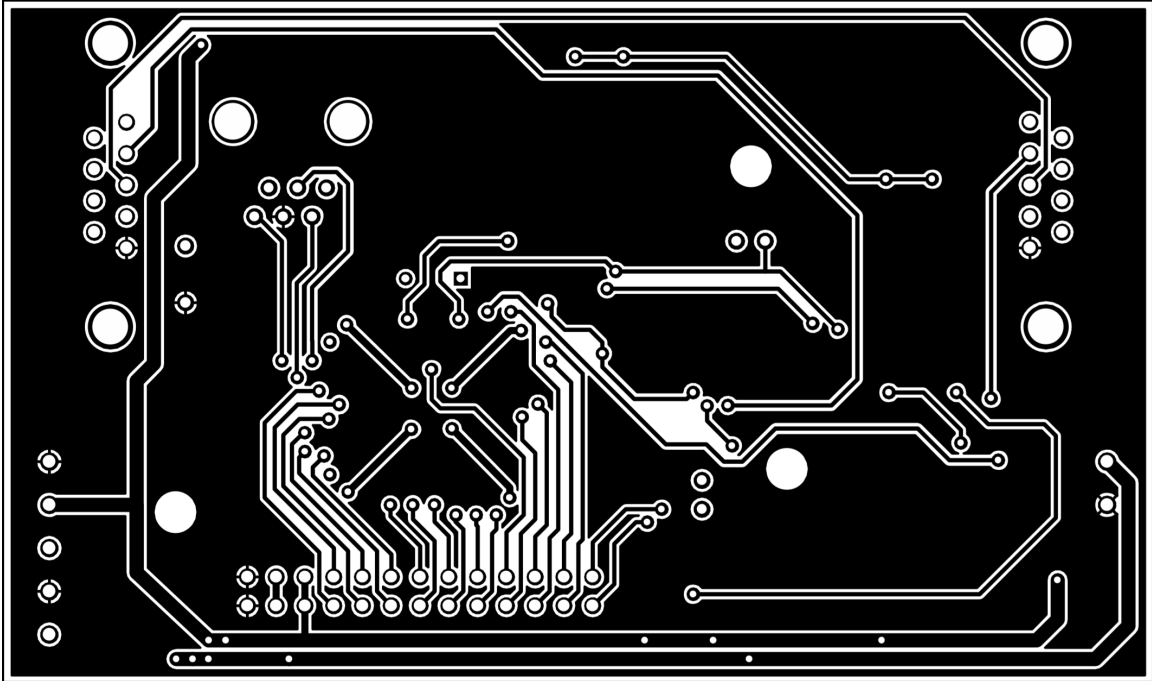


Figure A.6.1 Bottom copper layer of microcontroller

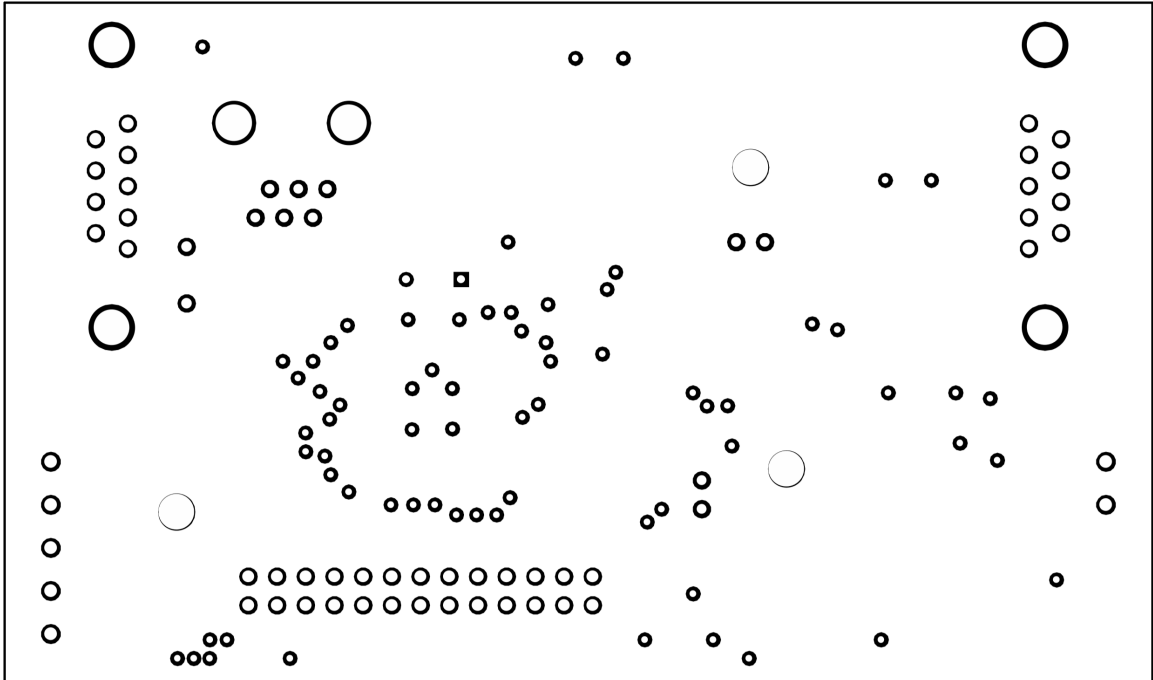


Figure A.4.2 Bottom solder mask

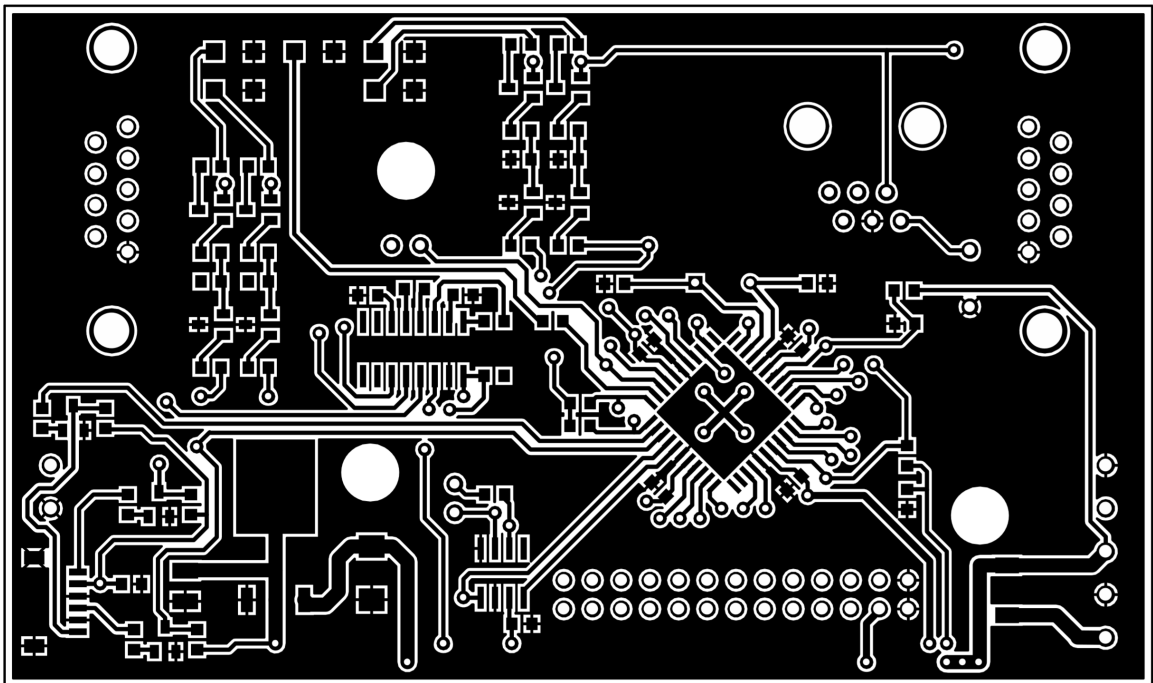


Figure A.6.3 Top copper layer

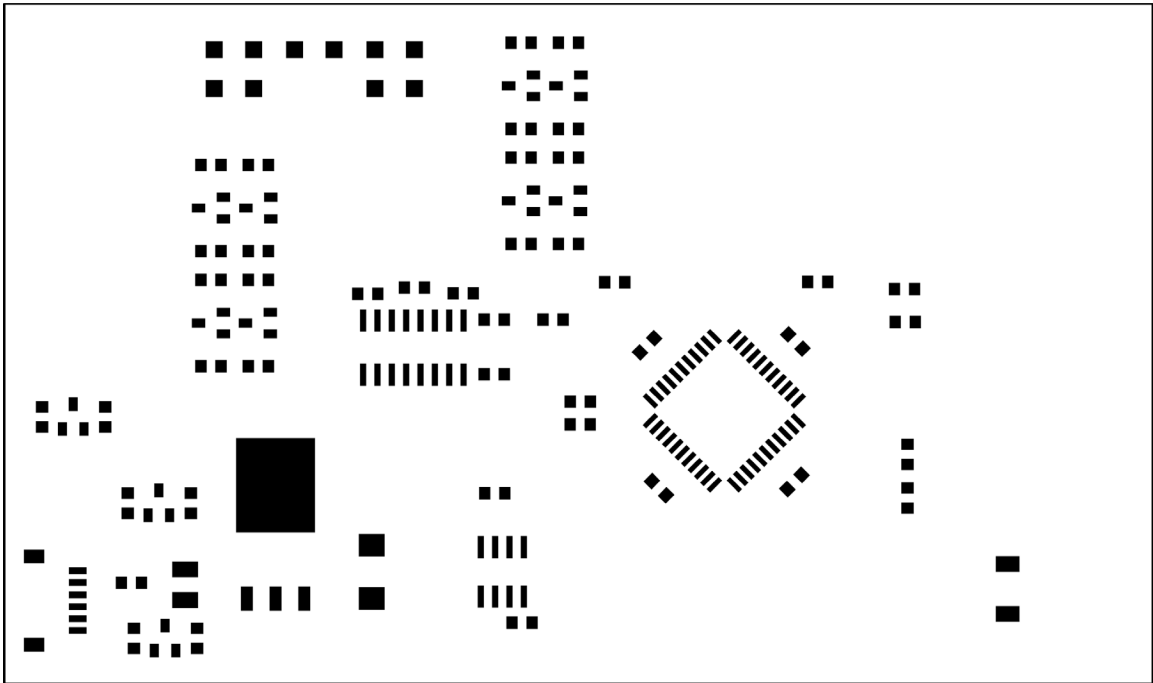


Figure A.6.4 Top paste mask

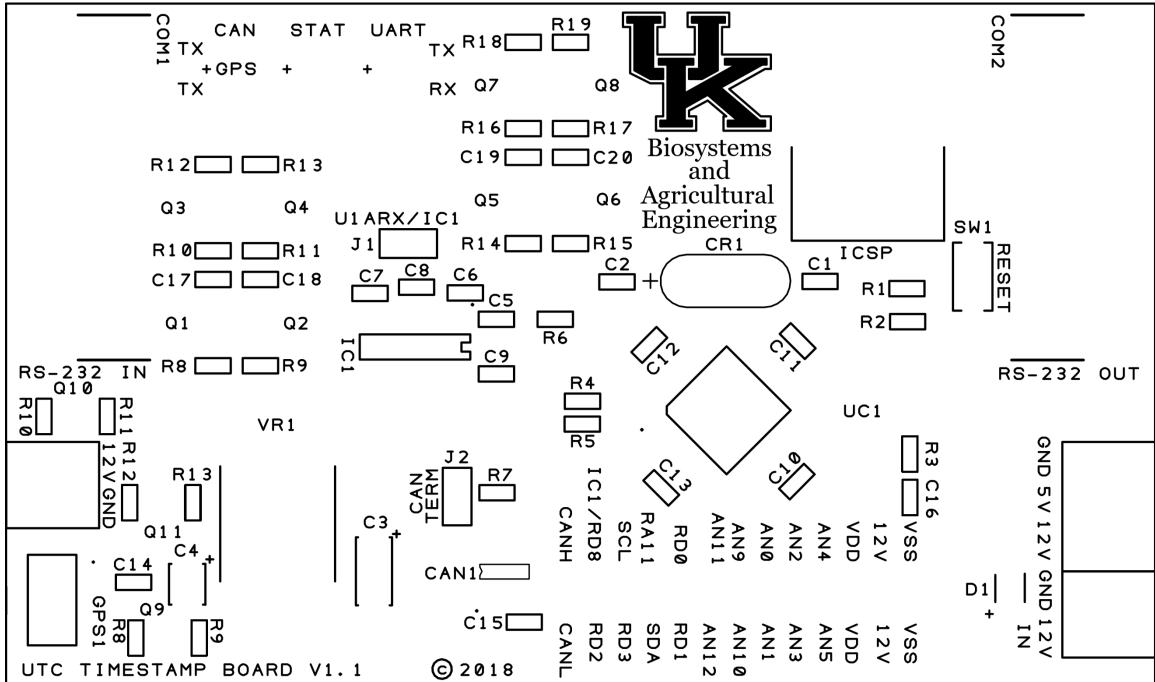


Figure A.6.5 Top silkscreen

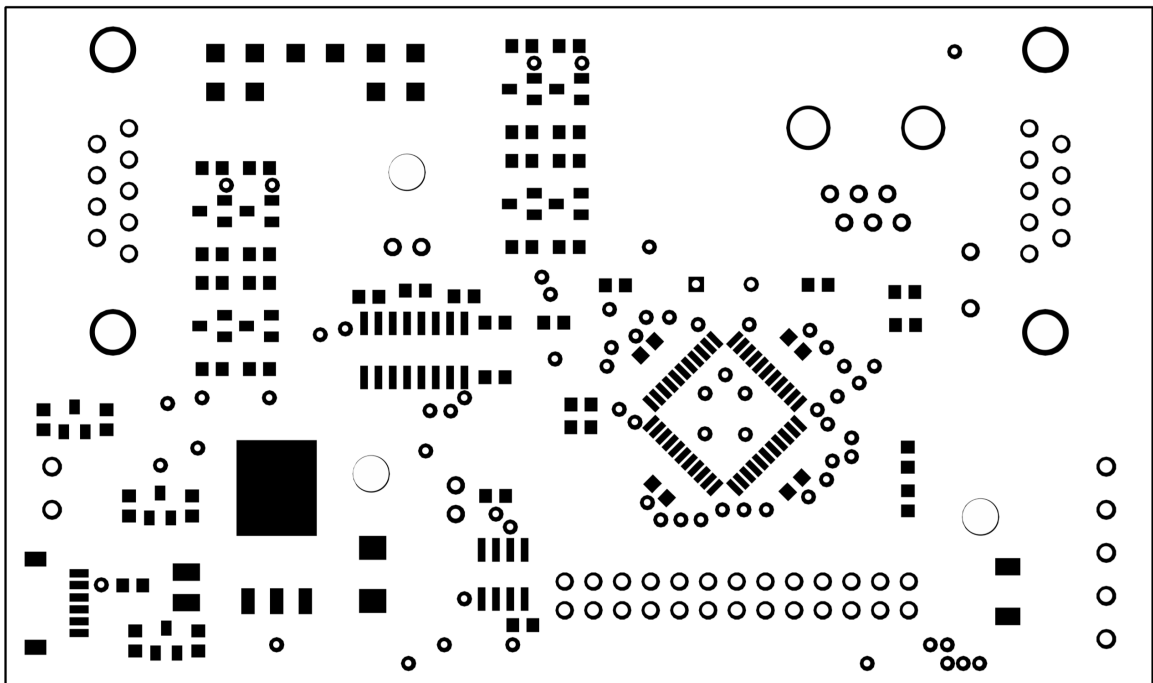


Figure A.6.6 Top solder mask

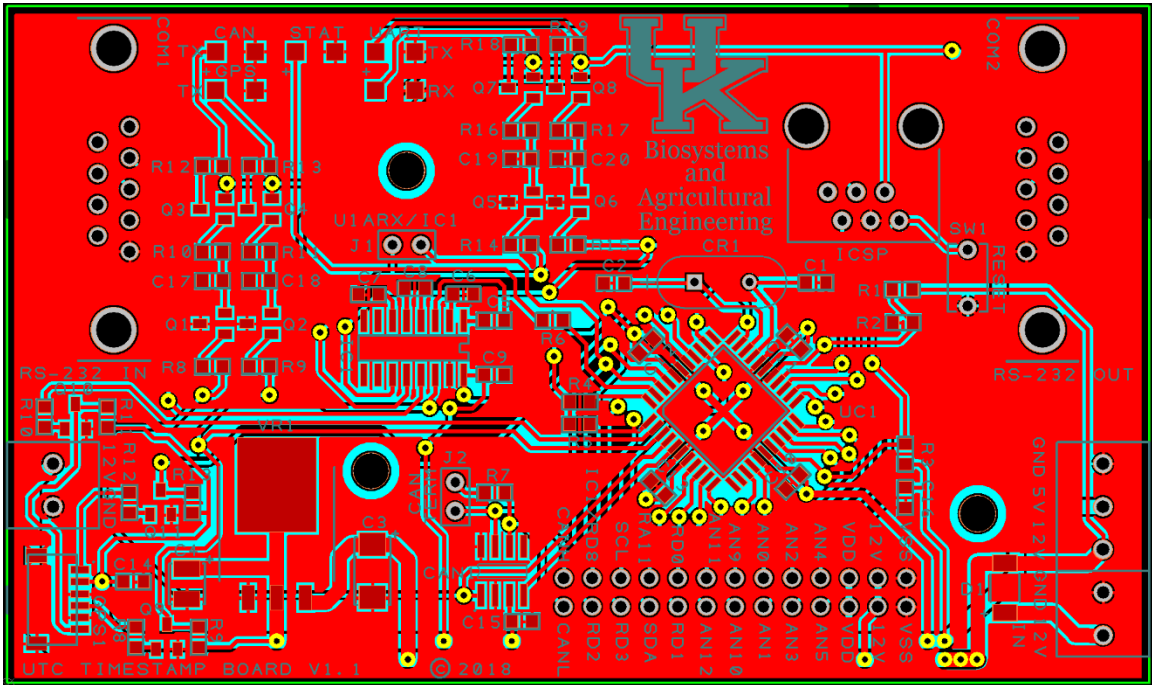


Figure A.6.7 PCB color drawing

Table A.6 PCB Bill of Materials:

Compon-ents	Description	Manufac-turer Part Number	Manufac-turer	Supplier Part Number	Suppli-er	Unit Cost	Unit Quan-tity	Unit Exten-ded
UC1	Digital Signal Processor	DSPIC30F4013-30I/PT	Microchip Technology	DSPIC30F401330IPT-ND	Digi-Key Corporation	\$5.73	1	\$5.73
COM1	Male D-SUB 9 Connector	5747840-3	TE Connectivity	A32091-ND	Digi-Key Corporation	\$2.55	1	\$2.55
COM2	Female D-SUB 9 Connector	1734354-1	TE Connectivity	A35107-ND	Digi-Key Corporation	\$1.57	1	\$1.57
GPS1	Right Angle JST connector	SM06B-SRSS-TB(LF)(SN)	JST Sales America Inc.	455-1806-1-ND	Digi-Key Corporation	\$0.83	1	\$0.83
ICSP1	6P6C RJ-11 Jack	5520470-3	TE Connectivity	A31417-ND	Digi-Key Corporation	\$2.35	1	\$2.35
CR1	15 Mhz Crystal Oscillator	ECS-150-S-4X	ECS Inc.	X1070-ND	Digi-Key Corporation	\$0.69	1	\$0.69
VR1	5.0V Linear Voltage Regulator	LM1084IS-5.0/NOPB	Texas Instruments	LM1084IS-5.0/NOPB-ND	Digi-Key Corporation	\$2.85	1	\$2.85
IC1	RS-232 Level Shifter	MAX232DR	Texas Instruments	296-14619-1-ND	Digi-Key Corporation	\$1.37	1	\$1.37
CAN1	CAN Transceiver	MCP2551T-I/SN	Microchip Technology	MCP2551-I/SN-ND	Digi-Key Corporation	\$1.08	1	\$1.08
D1	2A 40V Shottky Diode	CD1206-B240	Bournes Inc.	CD1206-B240CT-ND	Digi-Key Corporation	\$0.51	1	\$0.51
J3	2 Position Jumper	382811-8	TE Connectivity	A26228-ND	Digi-Key Corporation	\$0.13	1	\$0.13
T1,T3	2 Position Terminal Block	284392-2	TE Connectivity	A98166-ND	Digi-Key Corporation	\$1.12	2	\$2.24
POWER1	3 Position Terminal Block	284392-3	TE Connectivity	A98167-ND	Digi-Key Corporation	\$1.68	1	\$1.68
C3	10 uF Capacitor (Tantalum 2313)	F931V106MCC	Nichicon	478-8332-1-ND	Digi-Key Corporation	\$1.01	1	\$1.01

C4	10 uF Capacitor (Tantalum 1210)	F931A106M BA	Nichicon	478-8194-1-ND	Digi-Key Corporation	\$0.50	1	\$0.50
LED5	Red LED (1206 2V)	LTST-C150KRKT	Lite-On Inc	160-1405-1-ND	Digi-Key Corporation	\$0.39	1	\$0.39
LED1, LED2, LED3, LED 4	Blue LED (1206 3.3V)	LTST-C150TBKT	Lite-On Inc	160-1643-1-ND	Digi-Key Corporation	\$0.39	4	\$1.56
R1, R3, R8-R11, R14-R17	10 K Resistor (Thick Film 0603)	CRCW060310K0FKEA	Vishay Dale	541-10.0KHCT-ND	Digi-Key Corporation	\$0.10	10	\$1.00
R6, R12, R13, R18, R19	470 Resistor (Thick Film 0603)	CRCW0603470RJNEA HP	Vishay Dale	541-470SACT-ND	Digi-Key Corporation	\$0.17	5	\$0.85
R7	120 Resistor (Thick Film 0603)	CRCW0603120RFKEA	Vishay Dale	541-120HCT-ND	Digi-Key Corporation	\$0.10	1	\$0.10
R4, R5	4.7 K Resistor (Thick Film 0603)	CRCW06034K70FKEA	Vishay Dale	541-4.70KHCT-ND	Digi-Key Corporation	\$0.08	2	\$0.16
R2	1 K Resistor (Thick Film 0603)	CRCW06031K00FKEC	Vishay Dale	541-2986-1-ND	Digi-Key Corporation	\$0.10	2	\$0.20
C5-C9, C17-C20	1 uF Capacitor (Ceramic 0603)	C0603C105Z3VAC7867	KEMET	399-14943-6-ND	Digi-Key Corporation	\$0.26	9	\$2.34
C1, C2, C10-C16	0.1 uF Capacitor (Ceramic 0603)	GRM188R72A104KA35D	Murata Electronics North America	490-3285-1-ND	Digi-Key Corporation	\$0.23	9	\$2.07
SW1	SPST NO Tactile Switch	EVQ-PJJ04T	Lite-On Inc	P12240SCT-ND	Digi-Key Corporation	\$0.45	1	\$0.45
Q1 - Q8	PNP Transistor	MMBT3906LT3G	ON Semiconductor	MMBT3906LT3GOSC T-ND	Digi-Key Corporation	\$0.17	4	\$0.68
H1	26 Pin Header	PRPC013DAAN-RC	Sullins Connector Solutions	S2011EC-13-ND	Digi-Key Corporation	\$0.69	1	\$0.69
Q9 - Q11	MOSFET N-CH 30V 6A SOT23	SI2338DS-T1-GE3	Vishay Siliconix	SI2338DS-T1-GE3CT-ND	Digi-Key Corporation	\$0.52	3	\$1.56
GPS	Garmin GPS 18 LVC	010-00321-31	GARMIN	010-00321-31	Garmin	\$84.99	1	\$84.99

UTC_Timestamp Code:

```
////////////////////////////////////
// Title: UTC_Timestamp //
// Author: Michael P. Sama //
// Date: 3/27/18 //
// Description: This program configures the dsPIC30F4013 to measure the local times
when //
// a Pulse Per Second (PPS) signal is received on IC2 and the leading edge of //
// a serial data stream into UART1 on IC1. A GPGGA string from a GPS receiver //
// is also recieved by UART2 for determing the local time in UTC. Serial data //
// recieved on UART1 is appended with a UTC timestamp (HH:MM:SS:mms) upon
//
// reception of a carriage return (0x0D).//
// Notes: UART1 and and UART2 are configured to 38,400-8-N-1. Data rates on UART2
in //
// excess of 1 Hz or streams that include messages other than GPGAA may result in
//
// ISR overflows. //
////////////////////////////////////

/**Definitions**/
#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**/
#define IC1_OV IC1CONbits.ICOV //Flag indicates IC1 buffer was not read
before next event
#define IC2_OV IC2CONbits.ICOV //Flag indicates IC2 buffer was not read
before next event
#define UART1_OV U1STAbits.OERR //Flag indicates UART1 RX buffer overflow
#define UART2_OV U2STAbits.OERR //Flag indicates UART2 RX buffer overflow
```

```

#define LED PORTFbits.RF6//Status LED

****External References****
#include <p30fxxxx.h> //Base library for the dsPIC30F
#include <libpic30.h> //General c30 Functions (delays, etc.)
#include <uart.h> //Universal Asynchronous Receiver/Transmitter
#include <string.h> //String Manipulation
#include <stdio.h> //Standard Input/Output
#include <InCap.h> //Input Capture
#include "RS232.h" //Custom RS-232 Message Processing Library

****Microcontroller Configuration****
_FOSC(HS) //Set the oscillator to external high speed crystal
_FWDT(WDT_OFF) //Turn off the watch dog timer

****Global Variables****
char TXdata1[128]; //Data transmit string PPS event (disabled by default)
char TXdata2[128]; //Data transmit string UTC timestamp
char RXdata[128]; //Data receive string for UART2

unsigned int PPS = 0; //Stores the Timer2 value at a PPS event
unsigned int SER = 0; //Stores the Timer2 value at a serial data recieved event

unsigned int UTC_H = 0; //UTC Hours unsigned int UTC_M = 0; //UTC
Minutes unsigned int UTC_S = 0; //UTC Seconds unsigned int UTC_MS = 0; //UTC
Milliseconds

unsigned int PPS_Flag = 0; //Indicates a new PPS event has occured
unsigned int SER_Flag = 0; //Indicates a new serial data has been recieved unsigned int
IC1_Flag = 0; //Indicates the first seral data character leading edge unsigned int PPS_IL
= 0; //Interlock for PPS signal

```

```

unsigned int GPS_IL = 0;    //Interlock for GPS data
unsigned int SER_IL = 0;    //Interlock for serial data

****Function Prototypes****

void attribute (( interrupt )) _U1RXInterrupt(void); //UART1 receive interrupt handler
void attribute (( interrupt )) _U2RXInterrupt(void); //UART2 receive interrupt handler
void attribute (( interrupt )) _IC1Interrupt(void); //IC1 interrupt handler
void attribute (( interrupt )) _IC2Interrupt(void); //IC2 interrupt handler

****UART1 Receive Interrupt Handler****

void attribute ((interrupt, no_auto_psv)) _U1RXInterrupt(void)
{
unsigned char Character = ReadUART1(); //Read character from UART1 RX buffer
if (Character == 0x0D) //Carriage Return
{
SER_Flag = 1; //Rase the serial data flag since the string has completed

IC1CONbits.ICM = 2;    //Turn on IC1 Module after receiving carrage return
}
else if (Character == 0x0A) //New Line
{
//do nothing
}
else //Any other serial data character
{
putcUART1(Character);    //Echo the character out UART1 TX
}
IFS0bits.U1RXIF = 0; //Clear UART1 RX interrupt flag
}

```

```

****UART2 Receive Interrupt Handler****

void attribute ((interrupt, no_auto_psv)) _U2RXInterrupt(void)
{
IFS0bits.U2RXIF = 0; //Clear UART2 RX interrupt flag
char Character = ReadUART2(); //Read character from UART2 RX buffer
//putcUART1(Character); //Enable for debugging, passes GPS data through UART1
CharacterBuffer(Character); //Process character to compile data string
}

****IC1 Receive Interrupt Handler****

void attribute ((interrupt, no_auto_psv)) _IC1Interrupt(void)
{
IFS0bits.IC1IF = 0; //Clear IC1 interrupt flag
IC1CONbits.ICM = 0; //Turn off IC1 module until re-armed by serial character SER =
IC1BUF; //Store Timer2 value of the first serial character SER_IL += 1;
//Increment serial data interlock value
IC1_Flag = 1; //Raise IC1 flag in the main function
}

//IC2 Receive Interrupt Handler
void attribute ((interrupt, no_auto_psv)) _IC2Interrupt(void)
{
IFS0bits.IC2IF = 0; //Clear IC2 interrupt flag UTC_S += 1; //Increment elapsed UTC
second
PPS = IC2BUF; //Store Timer2 value of the PPS signal PPS_IL = 1; //Raise PPS
interlock value
//PPS_Flag = 1; //Enable for debugging, causes PPS Timer2 value to transmitted
}

****Main Function****

int main (void)

```

```

{
****I/O Type and Direction****

ADPCFG = 0b000000000111000; //Sets ANx pins to analog (0) or digital (1) TRISF =
0b10011111; //RF5 and RF6 set to output (LED, UART1)

****Open UART1 38400-8-N-1**** (uses Microchip C30 library) 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, 5);

U1MODEbits.ALTIO = 1; //Set UART1 to the default pins

****Open UART2 38400-8-N-1**** (uses Microchip C30 library) 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, 5);

****Configure Timer 2****

T2CONbits.TSIDL = 0; //Timer2 operation in Idle mode T2CONbits.TGATE = 0;
//Timer2 gated time accumulation disabled T2CONbits.TCKPS = 2;
//Timer2 input clock prescale bits set to 1:64 T2CONbits.T32 = 0; //Timer2 and
Timer3 form separate 16-bit timers T2CONbits.TCS = 0; //Timer 2 uses internal clock
source (FOSC/4) T2CONbits.TON = 1; //Start Timer2

****Configure IC 1****

IC1CONbits.ICM = 0; //Turn off IC1 Module while configuring

IC1CONbits.ICSIDL = 0; //Input capture module will continue to operate in CPU Idle
Mode IC1CONbits.ICTMR = 1; //TMR 2 contents are captured on capture event

```

```

IC1CONbits.ICI = 0; //interrupt on every capture event IC1CONbits.ICM = 2;
    //capture every falling edge

****Configure IC 2****

IC2CONbits.ICM = 0;      //Turn off IC1 Module while configuring
IC2CONbits.ICSIDL = 0; //Input capture module will continue to operate in CPU Idle
Mode IC2CONbits.ICTMR = 1;    //TMR 2 contents are captured on capture event
IC2CONbits.ICI = 0; //interrupt on every capture event IC2CONbits.ICM = 2;
    //capture every falling edge

****I/O Initialization and Startup Output String**** putsUART1((unsigned int
*)"\r\n****Ultrasonic Anemomter V 1.1****\r\n"); LED = 1;

****Interrupt Priorities**** IPC2bits.U1RXIP = 7; //Highest priority IPC6bits.U2RXIP
= 4;    //Lowest priority IPC0bits.IC1IP = 6;
IPC1bits.IC2IP = 5;

//Enable Interrupts
IEC0bits.U1RXIE = 1;    //Enable UART 1 RX interrupt IEC1bits.U2RXIE = 1;
    //Enable UART 2 RX interrupt IEC0bits.IC1IE = 1; //Enable IC1 interrupt
IEC0bits.IC2IE = 1; //Enable IC2 interrupt

****Main loop to handle data processing
while(1)    //Loop indefinitely
{
if (IC1_OV) //IC1 buffer was not read before next IC1 event
{
IC1CONbits.ICM = 0; //Reset module IC1CONbits.ICM = 2; //Capture every falling
edge
putsUART1((unsigned int *)"IC1 OVERRUN\r\n");//Report overrun
}
if (IC2_OV) //IC2 buffer was not read before next IC2 event
{

```

```

IC2CONbits.ICM = 0; //Reset module IC2CONbits.ICM = 2; //Capture every rising edge
putsUART1((unsigned int *)"IC2 OVERRUN\r\n");//Report overrun
}
if (UART1_OV) //UART1 RX buffer overrun
{
UART1_OV = 0; //Clear overrun flag
putsUART1((unsigned int *)"UART1 RX OVERRUN\r\n");//Report overrun
}
if (UART2_OV) //UART2 RX buffer overrun
{
UART2_OV = 0; //Clear overrun flag
putsUART1((unsigned int *)"UART2 RX OVERRUN\r\n");//Report overrun
}
if (PPS_Flag) //PPS flag raised by IC2 interrupt
{
PPS_Flag = 0; //Clear PPS flag sprintf(TXdata1,"$PPS,%u\r\n",PPS); //Compile
output string putsUART1((unsigned int *) TXdata1); //Transmit PPS Timer2 value
}
if (SER_Flag) //Serial data flag raised by UART1 RX interrupt
{
SER_Flag = 0; //Clear serial data flag
if (UTC_S > 59) //UTC seconds overflow
{
UTC_S = 0; //Set UTC seconds to zero UTC_M += 1; //Increment UTC minutes
if (UTC_M > 59) //UTC minutes overflow
{
UTC_M = 0; //Set UTC minutes to zero UTC_H += 1; //Increment UTC hours
if (UTC_H > 23) //UTC hours overflow
{
UTC_H = 0; //Set UTC hours to zero

```



```

}
}
}

sprintf(TXdata2, " %02u:%02u:%02u:%03u\r\n",UTC_H,UTC_M,UTC_S,UTC_MS);
//Compile timestamp putsUART1((unsigned int *) TXdata2); //Transmit timestamp
}

if (IC1_Flag && PPS_IL && GPS_IL) //Serial character flag raised by IC1 and
interlocks passed
{
IC1_Flag = 0; //Clear serial character flag
if (SER >= PPS) //Timer2 values are in order
{
//Compute the elapsed milliseconds since most recent PPS
UTC_MS = (unsigned int) ((unsigned long) (SER - PPS) * 1000 / 58594);
}
else //Timer2 values wrap around 65535
{
//Unwrap and compute the elapsed milliseconds since most recent PPS UTC_MS =
(unsigned int) ((unsigned long) (65535-PPS+SER) * 1000 / 58594);
}
if ((SER_IL > 6)) //No new GPS data have been received since the last six serial
messages
{
UTC_H = 0; //Clear UTC hours UTC_M = 0; //Clear UTC minutes UTC_S = 0; //Clear
UTC seconds
UTC_MS = 0; //Clear UTC milliseconds
SER_IL = 7; //Limit the interlock from incrementing indefinitely
}
}
if (NewMessage()) //A new NEMA 0183 message has been detected
{

```

```

strcpy(RXdata, GetMessage()); //Retrieve the data from the message buffer
if ((RXdata[0] == '$') && //"$GPGGA" has been received (RXdata[1] == 'G') &&
(RXdata[2] == 'P') &&
(RXdata[3] == 'G') &&
(RXdata[4] == 'G') &&
(RXdata[5] == 'A'))

seconds
{
UTC_H = 10*(RXdata[7] - 48) + (RXdata[8] - 48); //Decode and store the UTC hours
UTC_M = 10*(RXdata[9] - 48) + (RXdata[10] - 48); //Decode and store the UTC
minutes UTC_S = 10*(RXdata[11] - 48) + (RXdata[12] - 48); //Decode and store the
UTC

GPS_IL = 1; //Set the GPS data interlock
}
SER_IL = 0; //Reset the serial interlock counter to zero
}
}
return 0; //Program does not reach this line due to infinite while loop
}

```

REFERENCES:

- Azaroff, L. S., & Neas, L. M. (1999). Acute Health Effects Associated with Nonoccupational Pesticide Exposure in Rural El Salvador. *Environmental Research*, 80(2), 158-164. doi:<https://doi.org/10.1006/enrs.1998.3878>
- B. Smith, D., E. Bode, L., & D. Gerard, P. (2000). PREDICTING GROUND BOOM SPRAY DRIFT. *Transactions of the ASAE*, 43(3), 547. doi:<https://doi.org/10.13031/2013.2734>
- Baetens, K., Ho, Q. T., Nuyttens, D., De Schampheleire, M., Melese Endalew, A., Hertog, M. L. A. T. M., . . . Verboven, P. (2009). A validated 2-D diffusion–advection model for prediction of drift from ground boom sprayers. *Atmospheric Environment*, 43(9), 1674-1682. doi:<https://doi.org/10.1016/j.atmosenv.2008.12.047>
- Butler Ellis, M. C., Alanis, R., Lane, A. G., Tuck, C. R., Nuyttens, D., & van de Zande, J. C. (2017). Wind tunnel measurements and model predictions for estimating spray drift reduction under field conditions. *Biosystems Engineering*, 154(Supplement C), 25-34. doi:<https://doi.org/10.1016/j.biosystemseng.2016.08.013>
- Calculate distance, bearing and more between Latitude/Longitude points. Retrieved from <https://www.movable-type.co.uk/scripts/latlong.html>
- Contini, D., Donato, A., & Belosi, F. (2006). Accuracy of Measurements of Turbulent Phenomena in the Surface Layer with an Ultrasonic Anemometer. *Journal of Atmospheric & Oceanic Technology*, 23(6), 785-801.
- D. Luck, J., A. Shearer, S., P. Sama, M., & K. Pitla, S. (2015). Control System Development and Response Analysis of an Electronically Actuated Variable-Orifice Nozzle for Agricultural Pesticide Applications. *Transactions of the ASABE*, 58(4), 997. doi:<https://doi.org/10.13031/trans.58.10945>
- D. Luck, J., K. Pitla, S., P. Sama, M., & A. Shearer, S. (2015). Flow, Spray Pattern, and Droplet Spectra Characteristics of an Electronically Actuated Variable-Orifice Nozzle. *Transactions of the ASABE*, 58(2), 261. doi:<https://doi.org/10.13031/trans.58.10798>
- de Snoo, G. R., & van der Poll, R. J. (1999). Effect of herbicide drift on adjacent boundary vegetation. *Agriculture, Ecosystems & Environment*, 73(1), 1-6. doi:[https://doi.org/10.1016/S0167-8809\(99\)00008-0](https://doi.org/10.1016/S0167-8809(99)00008-0)
- Duke, S. O. (2005). Taking stock of herbicide-resistant crops ten years after introduction. *Pest management science*, 61(3), 211-218.
- Everitt, J. D., & Keeling, J. W. (2009). Cotton Growth and Yield Response to Simulated 2,4-D and Dicamba Drift. *Weed Technology*, 23(4), 503-506.
- Fujimura, M., & Maeda, J. (2009). *Cross-correlation of fluctuating components of wind speed based on strong wind measurement*.
- Gaglione, S. (2015). How does a GNSS receiver estimate velocity? *InsideGNSS*(March/April 2015), 38-41.
- Google. (n.d). Google Maps image of 38.026924°, -84.509623°.
- Gove, B., Power, S. A., Buckley, G. P., & Ghazoul, J. (2007). Effects of herbicide spray drift and fertilizer overspread on selected species of woodland ground flora: comparison between short-term and long-term impact assessments and field surveys. *Journal of Applied Ecology*, 44(2), 374-384.
- Grover, R., Maybank, J., Caldwell, B. C., & Wolf, T. M. (1997). Airborne off-target losses and deposition characteristics from a self-propelled, high speed and high clearance ground sprayer. *Canadian Journal of Plant Science*, 77(3), 493-500. doi:10.4141/P96-169

- Heinemann, D., Langner, D., Stabe, U., & Waldl, H.-P. (1997). *MEASUREMENT AND CORRECTION OF ULTRASONIC ANEMOMETER ERRORS AND IMPACT ON TURBULENCE MEASUREMENTS*.
- Holt, M. S. (2000). Sources of chemical contaminants and routes into the freshwater environment. *Food and Chemical Toxicology*, 38, S21-S27.
doi:[https://doi.org/10.1016/S0278-6915\(99\)00136-2](https://doi.org/10.1016/S0278-6915(99)00136-2)
- Imai, T., Kusunoki, K., Hono, Y., Takemi, T., Araki, K., Fukuhara, T., . . . Shibata, T. (2009). Spatial correlation and temporal fluctuations of wind velocities observed on a plain field. *Wind Engineering*.
- Landwehr, S., O'Sullivan, N., & Ward, B. (2015). Direct Flux Measurements from Mobile Platforms at Sea: Motion and Airflow Distortion Corrections Revisited. *Journal of Atmospheric and Oceanic Technology*, 32(6), 1163-1178. doi:10.1175/jtech-d-14-00137.1
- Lebeau, F., Verstraete, A., Stainier, C., & Destain, M. F. (2011). RTDrift: A real time model for estimating spray drift from ground applications. *Computers and Electronics in Agriculture*, 77(2), 161-174. doi:<https://doi.org/10.1016/j.compag.2011.04.009>
- Mekonnen, Y., & Agonafir, T. (2002). Pesticide sprayers' knowledge, attitude and practice of pesticide use on agricultural farms of Ethiopia. *Occupational Medicine*, 52(6), 311-315. doi:10.1093/occmed/52.6.311
- Ooms, D., Ruter, R., Lebeau, F., & Destain, M. F. (2003). Impact of the horizontal movements of a sprayer boom on the longitudinal spray distribution in field conditions. *Crop Protection*, 22(6), 813-820. doi:[https://doi.org/10.1016/S0261-2194\(03\)00045-0](https://doi.org/10.1016/S0261-2194(03)00045-0)
- Reid, S. J., & Turner, R. (2001). Correlation of Real and Model Wind Speeds in Different Terrains. *Weather and Forecasting*, 16(5), 620-627. doi:10.1175/1520-0434(2001)016<0620:coramw>2.0.co;2
- Robusto, C. C. (1957). The Cosine-Haversine Formula. *The American Mathematical Monthly*, 64(1), 38-40. doi:10.2307/2309088
- Sama, M. P., Stombaugh, T. S., & Lumpp, J. E. (2013). A hardware method for time-stamping asynchronous serial data streams relative to GNSS time. *Computers and Electronics in Agriculture*, 97(Supplement C), 56-60.
doi:<https://doi.org/10.1016/j.compag.2013.07.003>
- Vincenty, T. (1975). Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey review*, 23(176), 88-93.
- Xie, K., & Wang, K. (2011). Measurement of Wind Speed and Direction with Ultrasonic Sensor Using FPGA. *Energy Procedia*, 12(Supplement C), 837-843.
doi:<https://doi.org/10.1016/j.egypro.2011.10.110>
- Young, R. M. Instructions: ResponseONE Model 92000 Weather Transmitter. Retrieved from [http://www.youngusa.com/Manuals/92000-90\(A\).pdf](http://www.youngusa.com/Manuals/92000-90(A).pdf)
- Young, R. M. Instructions: Ultrasonic Anemometer Model 86000. Retrieved from [http://www.youngusa.com/Manuals/86000-90\(E\).pdf](http://www.youngusa.com/Manuals/86000-90(E).pdf)

VITA

Austin Weiss

EDUCATION:

- B.S, in Biosystems Engineering, University of Kentucky (Fall 2012- May 2017)

PROFESSIONAL EXPERIENCE:

- Sr. Application Technology Specialist, Syngenta (July 2019 – Present)
- Graduate Research Assistant, University of Kentucky (August 2017 – August 2019)
- Fellow, Office of Technology Commercialization, University of Kentucky (July 2018 – July 2019)
- Undergraduate Research Assistant, University of Kentucky (January 2017 – August 2017)
- Intern at Big Ass Solutions, Lexington, KY (August 2016 – January 2017)

PUBLICATIONS:

- Sama, Michael P.; Weiss, Austin M.; and Benedict, Emma K., "Validating Spray Coverage Rate Using Liquid Mass on a Spray Card" (2018). Biosystems and Agricultural Engineering Faculty Publications. 219.

CERTIFICATIONS:

- Engineer-In-Training: Passed the Fundamentals of Engineering exam (FE) August 2017

PROFESSIONAL SKILLS:

Programming (MATLAB, Visual Basic, Arduino), Computer-aided component design (Autodesk Inventor), Instrumentation, Data acquisition, Data processing and statistical analysis, Image analysis

PROFESSIONAL ORGANIZATIONS:

- Member of American Society of Agricultural and Biological Engineers (ASABE)