DEVELOPMENT AND EVALUATION OF A 360° DIFFERENTIAL PRESSURE GUST SENSOR FOR EXTREME WEATHER ENVIRONMENTS

By

ANDREW COLE

Bachelor of Science in Aerospace Engineering
Oklahoma State University
Stillwater, Oklahoma
2020

Bachelor of Science in Mechanical Engineering
Oklahoma State University
Stillwater, Oklahoma
2020

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2022

DEVELOPMENT AND EVALUATION OF A 360° DIFFERENTIAL PRESSURE GUST SENSOR FOR EXTREME WEATHER ENVIRONMENTS

Thesis Approved:

Dr. Jamey D. Jacob

Thesis Advisor

Dr. Brain R. Elbing

Dr. Ryan C. Paul

ACKNOWLEDGMENTS

This project is the product of a lot of effort from a lot of people. I would like to acknowledge those that have made major contributions to the work presented in the paper. First I would like to thank my committee. Their guidance and feedback through this process has been invaluable. Working on developing new technology with access to industry experts has been a fantastic experience. I would like to acknowledge Sean Waugh with the National Severe Storm Laboratory. His input was crucial to furthering my understanding of the problems addressed in this paper and for helping me to understand what would actually be helpful to weather researchers. I would like to thank the team of staff engineers at the Unmanned System's Research Institute, particularly Levi Ross. They provided guidance and expertise that allowed me to achieve results that would not have been possible otherwise. I would like to specifically acknowledge Dr. Jamey Jacob, the head of my committee. I owe most of my identity as an engineer to the opportunities he has provided me. They have deeply shaped my abilities and outlook on the field in the most positive way possible. Lastly I would like to thank my wife Rylee. Firstly for putting up with me while working on this thesis but also for supporting me throughout this entire process. Without her love and support this project would not have been possible.

Acknowledgments reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: ANDREW COLE

Date of Degree: JULY, 2022

Title of Study: DEVELOPMENT AND EVALUATION OF A 360° DIFFERENTIAL PRESSURE GUST SENSOR FOR EXTREME WEATHER ENVIRONMENTS

Major Field: MECHANICAL AND AEROSPACE ENGINEERING

Abstract: Measuring rapidly fluctuating high wind speeds and direction in the extreme weather environments of severe storms proves to be a difficult task with current sensors. For example, mechanical anemometers do not provide sufficient response time to detect micro-gusts and ultrasonic anemometers are quickly obscured by rain and other particles. In addition, most current systems are susceptible to damage from hail and debris. This effort details the design, implementation, and testing of a 360° gust probe using differential pressure sensors. Similar in concept to a multi-hole probe, this cylindrical probe uses multiple inlets connected to differential pressure transducers whose differential readings can provide high-resolution wind magnitude and direction measurements. This design can be manufactured using different transducers to allow for different accuracy, range, and resolution to meet the requirements for a wide range of applications, in its current test configuration the sensor is able to measure wind speeds. The probe has no external moving parts and is equipped with a purge system to vent water and debris from the inlets. This along with its small size and rigid construction make it damage resistant, creating a system uniquely suited to capture data in severe conditions. This paper includes discussion of existing systems and their shortcomings, the details and results of design process for a new instrument for collecting wind data, and some initial testing done with the instrument. Sample data is provided with analysis and comparisons to data sets from existing validated systems.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

1.1 Motivation 1

1.2 Background 2

1.2.1 Measuring Gusts 6

1.3 Goals and Objectives 8

1.4 Outline 8

II. LITERATURE REVIEW 10

2.1 Current Wind Measurement Solutions 10

2.1.1 Cup Anemometer and Weather Vane 10

2.1.2 Propeller Anemometer 12

2.1.3 Hot Wire Anemometer 13

2.1.4 Ultrasonic Anemometers 14

2.1.5 Pitot Tubes 15

2.1.6 Five Hole Probes 17

2.1.7 Flush Air Data Sensors 18

2.2 Pressure Based Systems 20

2.2.1 MEMS Based Cylinderical Sensor 20

Chapter Page

2.2.2 Extreme Turbulence Probe 21

2.3 Current Systems for Atmospheric Observation 23

2.3.1 TOtable Tornado Observatory 23

2.3.2 Weather Towers 24

2.3.3 Weather Balloons 26

2.3.4 Mobile Mesonets 27

2.3.5 Manned Aircraft 28

2.3.6 Unmanned Aerial Systems 29

III. OVERVIEW OF DIFFERENTIAL PRESSURE SENSING 31

3.1 What is Differential Pressure Sensing 31

3.2 Current Applications 33

3.2.1 Pitot Tubes 33

3.2.2 Five Hole Probes and Flush Air Sensors 34

3.3 360° Differential Sensing 34

3.3.1 Concept 35

3.3.2 Direction Calculation Techniques 37

IV. INSTRUMENT DESIGN 43

4.1 Design Requirements 43

4.2 Probe Development 45

4.2.1 General 47

4.2.2 Purge Mechanism 49

Chapter Page

4.2.3 Manufacturing 53

4.3 Electronic Development 55

4.3.1 Sensor Selection 56

4.3.2 Printed Circuit Board Design 58

4.4 Manufacturing 63

4.5 Multiplexers and Design Changes 65

4.6 Calibration 66

V. INITIAL TESTING AND PROOF OF CONCEPT 68

5.1 Proof of Concept Testing 68

5.2 Initial Test Result 69

VI. TESTING AND RESULTS 72

6.1 Wind Tunnel Trials 72

6.1.1 Static Tests 72

6.1.2 Dynamic Tests 76

6.2 Mesonet Tower Test 78

6.3 Mobile Mesonet Test 81

VII. CONCLUSION 84

7.1 Comparison to Existing Instruments 84

7.2 Advantages and Shortcomings 85

7.2.1 Future Work 86

Chapter Page

7.2.2 Final Thoughts 87

REFERENCES **88**

APPENDICES **92**

LIST OF TABLES

Table		Page
1	Suggested Benchmarks for Meteorological Data Collection [10]	44
2	Calibration Values	67

LIST OF FIGURES

Figure		Page
1	The Atmospheric Boundary Layer	2
2	Measurements of wind speed in a hurricane.	3
3	Development of Systems for Atmospheric Observations [24]	4
4	More Systems for Atmospheric Observations [24]	6
5	Cup Anemometer and Weather Vane	10
6	Propeller Anemometer	12
7	Probe of a Hot Wire Anemometer	13
8	Young 81000 Ultrasonic Anemometer	14
9	Pitot Tube combined with an Angle of Attack Sensor	16
10	5 Hole Probe from NASA's Lewis Research Center	17
11	Types of Five Hole Probe Tip Geometries[8]	18
12	FADS system on the nose of NASA's F/A-18 Systems Research Aircraft[7]	19
13	Muilthope probe based on MEMS Sensors[15]	20
14	Error Across Directional Range of MEMS based Anemometer. Sensors[15]	21
15	The Extreme Turbulence Probe. Sensors[9]	21
16	An Extreme Turbulence Probe Field Setup. Sensors[9]	22
17	The TOtable Tornado Observatory[5]	23
18	Mesonet Tower in Goodwell Oklahoma	24
19	Standard Layout for Oklahoma Mesonet Towers[16]	25
20	USRI Student Researchers Filling a Weather Balloon	26
21	Skew-T Plot showing the results of a sounding[1]	27
22	Mobile Mesonet used by the National Severe Storm Laboratory[21]	28
23	A Multihole Probe Equiped on a NCAR C130Q Hercules[14]	29
24	DJI Matrice 600 outfitted with a Young 8100 and aerosol sensor[4]	30
25	Basic Differential Pressure Configuration for finding Airspeed	33
26	360° Differential Pressures Concept	35
27	Surface Pressure Coefficients for Flow around a Cylinder [17]	40
28	Cylindrical and Spherical Probe Designs.	45
29	Spherical Probe Internals	46
30	Cylindrical Multi-hole Probe Design	47
31	Cylindrical Multi-hole Probe's Internal Compartment.	49
32	Cutaway view of a pressure port showing a debris trap	50
33	Purge Mechanism Diagram	51
34	A Semicircular Aeration Hole Located below the Pressure Ports	54

Figure Page

35 **Left:** SLA 3-D Printing **Right:** UV Curing Post Process 55
36 HSCMRRN005ND7A3 Differential Pressure Transducer 57
37 Multi-hole Dev Board v3 with Manufacturing Errors. 59
38 Example LTC4316 Setup 61
39 LTC4316 Wire Diagram 62
40 Fully Populated PCB 65
41 Finished Board with TCA9548 66

42 CMHP Initial Testing Setup 69
43 Static Test Raw Pressures 70
44 Static Test Calculated Differential Pressures 70
45 Static Test Calculated Velocities by Transducer 71
46 Cylindrical Multi-hole Probe's Initial Test Measurements. 71

47 Static Test Raw Pressures 73
48 Static Test Calculated Velocities 74
49 Static Test Calculated Direction 75
50 Dynamic Test Raw Pressures 76
51 Dynamic Test Calculated Velocities 77
52 Dynamic Test Calculated Direction 78
53 Test Tower set up next to Marena Mesonet Site 79
54 CMHP and a Young 81000 on a Blue Sky Mast Tower 80
55 Smoothed CMHP Wind Speed Data vs Marena Mesonet 81
56 Smoothed CMHP Wind Direction Data vs Marena Mesonet 81
57 Vehicle Setup to Simulate Mobile Mesonet 82
58 CMHP Wind Direction Data vs Young 92000 Mobile Mesonet 83

59 Comparison of the CMHP with other Anemometers 85
60 PCB layout. 92
61 Anemometer CAD. 93

CHAPTER I

INTRODUCTION

1.1 Motivation

The weather is one of the most impactful forces on humanity. As such, predicting the weather has be the focal point of research for centuries. Accurate weather predictions affect everything from children's sports to the sustainability of growing certain crops in a region. Severe weather in particular can lead to lasting impacts on areas by contributing to property damage and loss of life. This problem is only getting worse. The National Oceanic and Atmospheric Administration (NOAA) has recorded a rise in billion-dollar disaster events since 1980 with 262 deaths and over an estimated 102 billion US dollars of damage in just 2020 alone.[20] Better understanding of the formation and interaction of different weather systems can help to improve predictions, leading to more accurate threat assessments and longer warning times. This is an area that can save lives, along with billions of dollars.

Meteorology is entirely dictated by interactions within earth's atmosphere. The most important of these interactions takes place within the atmospheric boundary layer (ABL). Shown in Figure 1, the ABL home to the weather events that directly impact us. Unfortunately, it is also one of the least understood areas of our atmosphere. Events here take place above the height of towers, but below the flight of most manned aircraft. This means there is

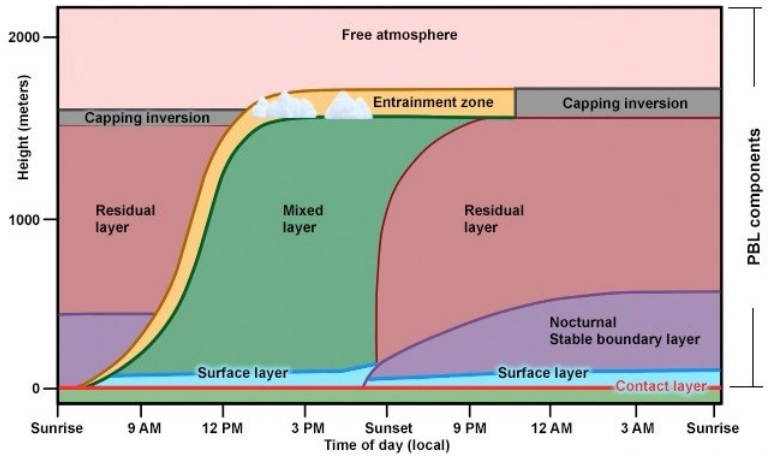

Figure 1: The Atmospheric Boundary Layer

comparatively very little data in this region, especially data on the wind. The development of a new sensor for measuring wind speed and direction that is designed to be integrated into existing weather observation platforms will help bridge this gap in data and could help drastically improve our understanding of weather as a whole.

1.2 Background

One of the largest problems facing modern meteorological research is one with data. While difficulties with data plague every scientific field, meteorology has a unique relationship with the collection and analysis of data. The main demand from meteorology is weather prediction. The prediction of the weather is widely accepted to be of extreme importance to humanity and because of this, it is possibly the area of the most data collection of any scientific field worldwide. This means that every day vast amounts of weather data is collected using various instrumentation and that data is then used to predict future weather events. This happens en mass across the globe and the impact of these predictions is incalculable.

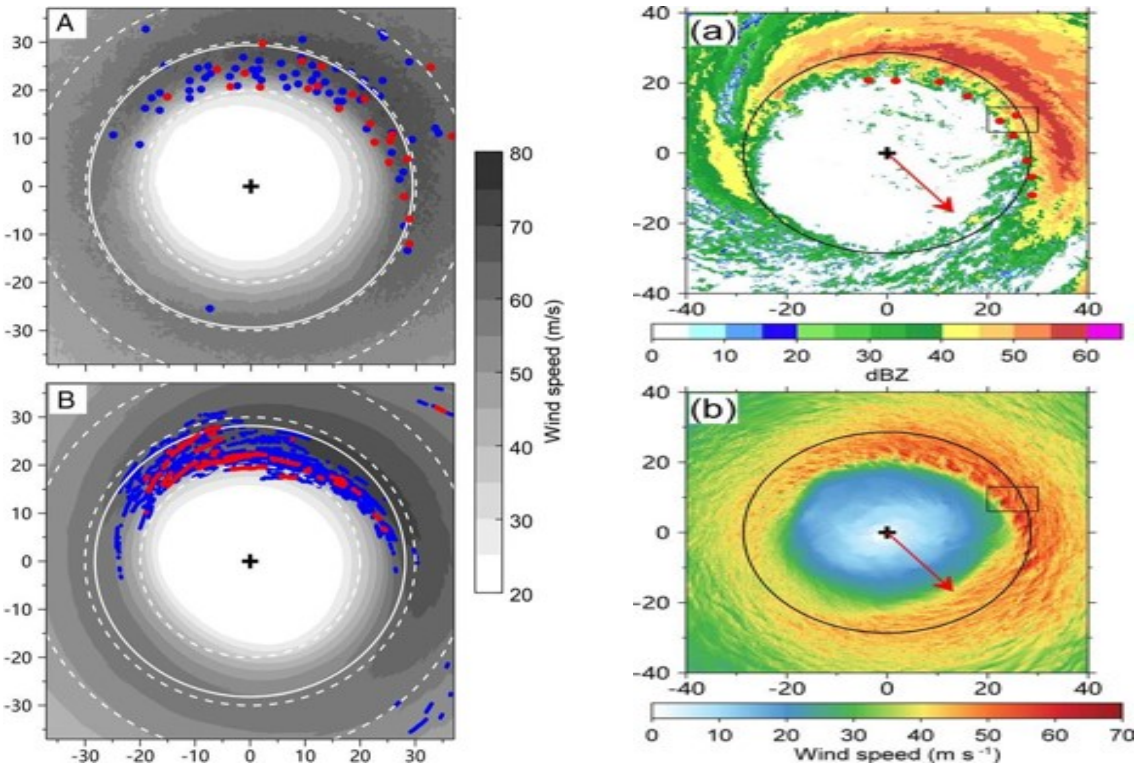

Figure 2: Measurements of wind speed in a hurricane.

While this is fantastic for researchers, it also comes with a fairly large drawback. To understand the hurdles faced by researchers one must understand the difference between the data needed for weather prediction and the data needed for weather research. For prediction, large amounts of meteorological data is needed over long periods of time across as big an area as possible to help identify trends and patterns of events taking place in the atmosphere. In contrast, meteorological research is essentially the study of the specific aerodynamics around the globe at the planetary scale. This aerodynamic analysis requires much higher quality data. In the broader approach used for prediction there isn't a specific need for the accuracy or frequency typically associated with the laboratory grade measurements required for weather research. This is underlying issue, is most data used for weather research isn't actually collected for the purpose of scientific research. Because there is such an immediate

necessity for weather prediction, only a very small amount of data is actually collected for the purpose of research. Normally when scientific data is collected it, it is targeted in such a way to be the precise data needed to test a specific hypothesis. Without this, researchers are stuck sifting through huge collections of low quality data retroactively. This makes the process slow, frustrating, and unlikely to yield results that meet the scientific rigor of other fields. This issue warrants action because on a higher level, weather prediction is a product of weather research. The only way to improve weather prediction is to improve the knowledge that informs weather prediction. This is done through better research.

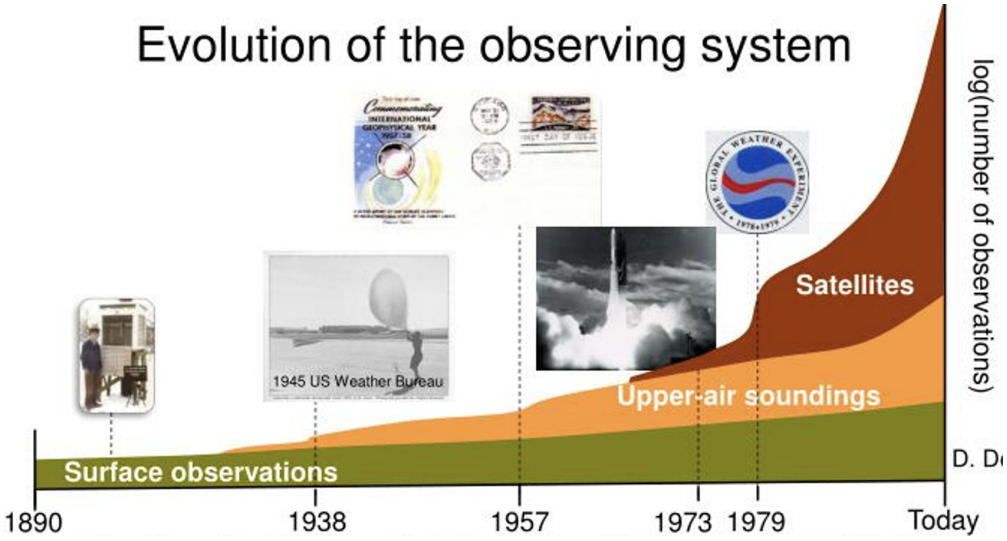

Figure 3: Development of Systems for Atmospheric Observations [24]

To improve prediction methods, meteorologists need the right data to be able to increase their understanding of weather formations. But, because weather events happen on such a massive scales with so many different influencing factors, it can be extremely difficult or even impossible to meaningfully recreate most events in a lab or simulate them in a computer. This brings us back to the problems mentioned before. While there is a vast amount of real world data available, very little of it is usable data of unique weather events. Most areas just

need general temperatures and rainfall predictions to plan around. Because of this, most sensors are designed to work at a very low resolution. Generally, instruments designed to monitor the weather at most take a measurement once per minute, with rates of even once every five minutes being considered "high resolution". This is because those rates are high enough to show the general trends in an area while also not producing high amounts of data that then has to be stored. This means that even if a meteorologist wants to study a specific event, and there happens to be data in the area at that time, the data typically isn't at a high enough resolution to actually be able to make an indications about things like fluctuations in air movements or changes in wind gusts. Because of this meteorologists trying to advance their understanding typically need to target large predictable formations and move higher resolution sensors to the area before they occur. This brings to light the next problem with prediction data, where it is collected. Most data comes from mesonet tower networks or stand alone weather stations. These towers aren't tall enough to give meaningful data about what is happening in the boundary layer. Methods like radar that are used for tracking weather aren't able to penetrate most formations and are unable to tell what is actually happening inside weather formations. Weather balloons offer the ability to take vertical profiles of data that can prove to be extremely insightful, but because they climb so quickly they're only able to capture data at each altitude for seconds at a time. The most promising methods for measuring the boundary layer are using unmanned aerial systems (UAS). These vehicles can fly directly anywhere data needs to be taken, but currently there are very few sensors that are compatible with UAS.

Based on these facts it can be said there exists a gap where the data that could be the most insightful to researchers cannot actually be obtained. Currently there are few reliable

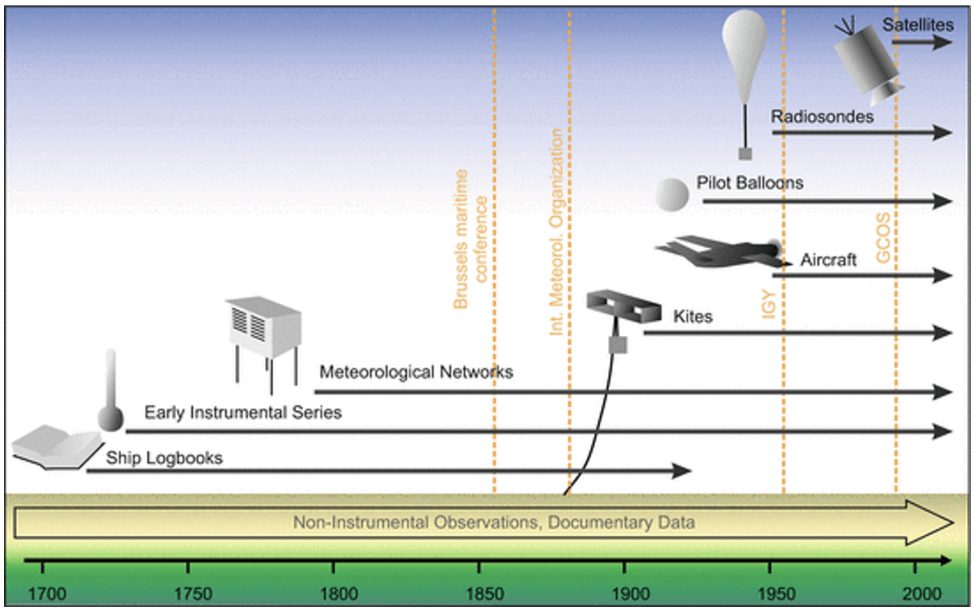

Figure 4: More Systems for Atmospheric Observations [24]

methods of directly collecting wind data during severe weather. On top of that there is the potential for existing weather monitoring systems to be upgraded to be able to capture addition wind information that may prove useful to weather forecasters. This could be improved with the development of a new instrument for measuring wind speed and direction that is easily integrated into unmanned aerial systems.

1.2.1 Measuring Gusts

When measuring gusts it could be helpful to understand how they are defined and what type of measurements are taken. Generally the speed of a wind gust, U_{max}, is defended as the maximum horizontal wind speed reached over a loner term sampling period (T). This can be described mathematically as the maximum of the moving averages with a moving average window length equal to the gust duration (t_g). Meteorologists generally report gusts in the

wind forecasts issued using a gust duration $f_g = 3$ s and a sample length T = 10 min.

Two other helpful parameters are the gust and peak factors. The gust factor is the ratio of the wind gust speed, U_{max}, to the mean horizontal wind speed, U.

$$G = \frac{U_{max}}{U} \tag{1.2.1}$$

The peak ratio is the difference of the wind gust speed from the mean normalized by the standard deviation of the horizontal wind speed (σ_u).

$$g_x = \frac{U_{max} - U}{\sigma_u} \tag{1.2.2}$$

G is proportional to the inverse of the mean wind speed. This means that in practice the measured G values show a large variability at low mean wind speeds. This generally smooths out at strong wind speeds (neutral conditions) where G reaches a nearly constant value that is characteristic of the measurement site. The average G value is influenced by environmental factors, such as the measurement height, surface roughness, atmospheric static stability, and on the temporal scales used in the gust definition, the gust duration t_g and the sample period, T.

In contrast, peak factor, the peak factor can be considered to be invariant of environmental conditions. This is because of it's normalized by the standard deviant, so instead it is a function of the gust time scales, t_g and T. Being able to measure the gust and peak factor helps meteorologist characterize gusts. These are the kinds of measurements that would be helpful to get from severe weather environments. [23]

1.3 Goals and Objectives

Based on the problems discussed above there arises the need for the development of a new wind sensing solution. In order to be helpful on a wider scale the new sensor will need to improve on existing technology in a variety of ways. The end product of this project must meet the following goals:

- Simple to operate in high stress or dangerous environments
- Competitive with the monetary cost of comparable sensors
- Capable of variable data collection rates
- Capable of accurate collection rates high enough to measure winds gusts
- Operate in severe weather environments (rain, extreme winds, hail, etc.)
- Durable enough to operate while sustaining impacts from debris

The goal is to produce a new scientific instrument that can collect high rate scientific data inside of severe weather, while still being able to operate at the capacity of a standard weather prediction anemometer.

1.4 Outline

This paper will discuss the design and testing of a new type of anemometer that uses differential pressure sensing to measure both the speed and direction of rapidly changing winds. It will begin with a more in depth examination of currently available anemometers. It then

continues discussing the fundamentals of this type of sensing followed by the unique application of these methods. Then the paper will cover the enactment of these new methods to design a new type of scientific instrument. Next the process of designing this new sensor and the considerations it has been afforded will be discussed. After this, the paper will examine results of its initial testing and comment on these initial findings. Possible applications of this technology are then explored with specific attention paid to the benefits of it's use in combination with mobile mesonet stations and unmanned aerial systems (UAS). Finally, it will then conclude with final thoughts and plans for future work.

CHAPTER II

LITERATURE REVIEW

2.1 Current Wind Measurement Solutions

Modern meteorologist use a variety of instruments to make observations regarding the wind. They come in a wide range of sizes each with different accuracy, cost, etc. While there have been numerous different solutions for measuring the wind, this portion of the literature review will only discuss those that are relevant to the designs presented later in this thesis.

2.1.1 Cup Anemometer and Weather Vane

Figure 5: Cup Anemometer and Weather Vane

Cup anemometers and weather vanes are almost a standard when it comes to measurements to enable weather prediction. The cup anemometer supplies wind speed by using the movement imparted on the device by the wind. A weather vane works in a similar way, but instead produces directional measurements. The instruments are independent of one another and can be used separately but are very commonly deployed in tandem since often both speed and direction are desired. They offer several benefits. First, they are a robust proven system that is easy to operate. They are also able to provide wind measurements in any direction along the two-dimensional plane they are oriented in. Second, they are extremely simple. Both parts of the system can be calibrated using a linear calibration curve. This makes it much faster and easier to calibrate them, which contributes to their last benefit: price. They are both relatively cheap to manufacture and certify. This allows them to be more readily obtained and deployed.

The cup anemometer and weather vane are not without downsides, however. Because they are reliant on external moving parts, they become stuck over time and can be damaged by high winds. In addition, because these sensors are reliant on the wind to impart its momentum on them, they inherently have slow response times. This means that gusts and rapid changes in the wind are often missed and not recorded. On top of this, they can suffer from what is known as "over speeding". This is when the wind speed decreases, but the anemometer's cups continue to spin at a higher rate due to their momentum. This causes the device to report wind speeds higher than what actually occurred. Cup anemometers and weather vanes are good tools for weather monitoring; but are often unable to provide the accurate data necessary for research due to these shortcomings.[12]

2.1.2 Propeller Anemometer

Figure 6: Propeller Anemometer

Propeller anemometers work using similar principals to cup anemometers and weather vanes. The biggest difference being that the propeller anemometer consists of a cup anemometer and a weather vane combined into one. The cups of the anemometer are replaced with a propeller and are mounted directly to the front of the weather vane. This leads to even simpler logistics and set up since only one device is needed. However, this means the propeller anemometer suffers from many of the same issues that were previously discussed. Because everything is still inertial based, slow response rate is still an issue. The propeller blades are less susceptible to over speeding than cups, but don't eliminate the problem completely. There is also a new issue that arises with the addition of the anemometer to the weather vane. The propeller itself isn't omni-directional and relies on the weather vane to correctly align it with the wind. This means the wind speed readings are even further slowed as the propeller can't start matching speed with the wind until the weather vane corrects. Compli-

cations can be also come from the interaction between the weather vane and the propeller. In the absence of wind, the rotating of the weather vane can cause the propeller to spin and report false wind readings. The interaction of these parts also means the system requires a more complex second order calibration. Similar to the cup anemometer and weather vane, the propeller anemometer is a great tool for weather monitoring but generally falls short for highly scientific measurements.[13]

2.1.3 Hot Wire Anemometer

Figure 7: Probe of a Hot Wire Anemometer

Hot wire anemometers work by placing a heated wire in a fluid flow. This allows them to take wind speed measurements without any moving parts. They come in two varieties that operate in different manners. Constant temperature hot wires use a variable current to heat the wire to a constant temperature, as the name implies; they can then correlate the amount of current used to heat the wire with the wind speed. In contrast, constant current hot wires supply the wire with a constant current and measure the wire's change in temperature as its exposed to the flow. Both options are extremely sensitive and as such,

can accurately produce extremely high frequency measurements. If calibrated properly, this method can even be used to measure turbulence. This is something that is extremely difficult due to the high resolution required to do so. Unfortunately, its high sensitivity is also its downfall. Special care must be given when placing them, as hot wire anemometers produce very erroneous readings if not correctly aligned with the flow. This is the first of many flaws that make hot wire anemometers difficult to use outside of a laboratory setting. They can also be affected by dust or water in the air, altering the heat exchange relationship between the air and the wire. Additionally, this method means that the heated wire is very delicate, and the device still doesn't yield any information on wind direction. Overall hot wires are great scientific instruments but have several short comings that make them difficult to use for field measurements.[6]

2.1.4 Ultrasonic Anemometers

Figure 8: Young 81000 Ultrasonic Anemometer

Ultrasonic anemometers consist of a precisely designed array of speakers and microphones. These devices use a concept known as inverse time transit difference.[18] The speakers emit ultrasonic pulses that are then measured by the microphones. As air moves through the anemometer, it changes the frequency of the pulses and the amount of time they take to travel from speaker to microphone. This can then be used to find not only wind speed, but direction as well. This method comes with a lot of advantages. It can take both 2D and 3D direction measurements, depending on the configuration of the pulse emitters and microphones. These anemometers can capture data at high enough frequencies for scientific measurements. They don't rely on moving parts and are generally much more compact than other devices that measure both speed and direction. They are not without their own pitfalls, however. They are generally much more expensive than other anemometers. In addition, they don't measure accurately if there is any dust, debris, or water in the air. They can also be quite delicate, depending on their design. Any misalignment of the pulse sources and microphones will drastically impact their accuracy and require them to be re-calibrated. These sensors are extremely helpful in taking scientific grade field measurements but are not suitable for use in severe weather. Their high cost also limits how many can feasibly be procured by researchers.

2.1.5 Pitot Tubes

Pitot tubes are a widely used tool to measure wind speed and is the first discussed in this paper that works on the principle of differential pressure sensing. While this will be discussed in more detail later, pitot tubes work by measuring the difference between the total pressure and the static pressure associated with a flow. This difference can then be used to find the

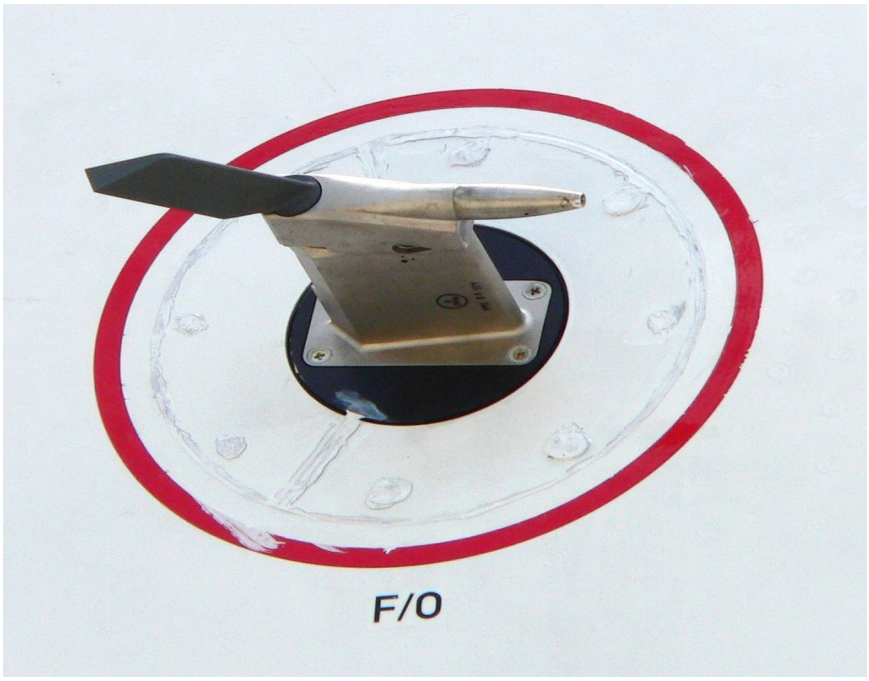

Figure 9: Pitot Tube combined with an Angle of Attack Sensor

speed of the air passing over it. They have a variety of uses, with the most common being air speed sensors for either operating aircraft or aerodynamic research. Due to their simplicity and lack of moving parts, they are extremely reliable and are often combined with other sensors, such as the angle of attack sensor on an airplane. They work in a large regime of flow speeds and can even be heated to prevent icing in sub-freezing temperatures. The most common limitation for a pitot tube is that they must be properly oriented with the flow. They are also prone to giving false readings if their pressure ports become clogged with water or debris. Even though they are a very widely used and robust sensor, because of their need to be properly oriented with the wind and inability to provide direction measurements, they are scarcely used for collecting meteorological data.

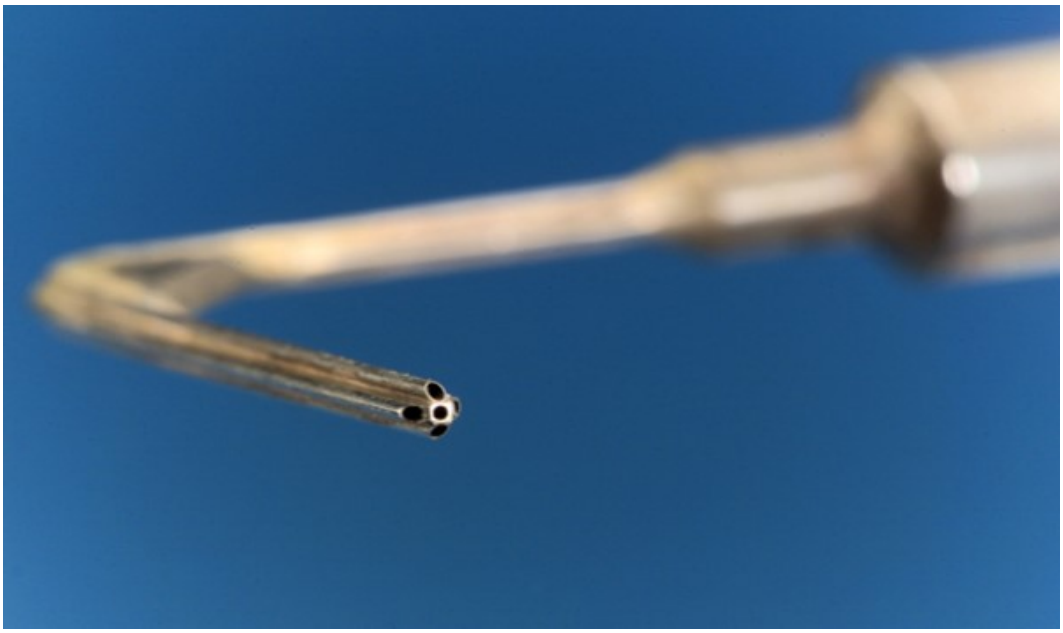

Figure 10: 5 Hole Probe from NASA's Lewis Research Center

2.1.6 Five Hole Probes

The five hole probe (5HP) is based on the pitot tube. It was created in 1915 by Admiral Taylor with the goal of allowing give pitot tubes the ability to measure wind direction on top of speed for use on ships. Similar to the pitot tube, it uses differential pressure to calculate wind speed. The difference is the addition of extra pressure ports on the tip of the probe. It is the first example covered in this paper of a multi-hole probe (MHP). The idea behind multi-hole probes is to use the differential pressure sensing to calculate both a flows speed and direction. The five hole probe, as the name suggests, uses five stagnation pressure ports in combination with a static port to take enough pressure measurements to calculate limited 3D directional measurements. Using the differences in pressures of all fives These probes are able to operate in a large range of Reynolds numbers but utilizing different geometric design of the probe's tip. These probes begin to address inabilities present in a pitot probe with

regards to weather measurements but unfortunately don't do enough to make them a viable option in most situations. Due to only having forward facing stagnation pressure ports the wind directions other can measure are limited, generally in a range from -25 to 25 degrees in any direction.

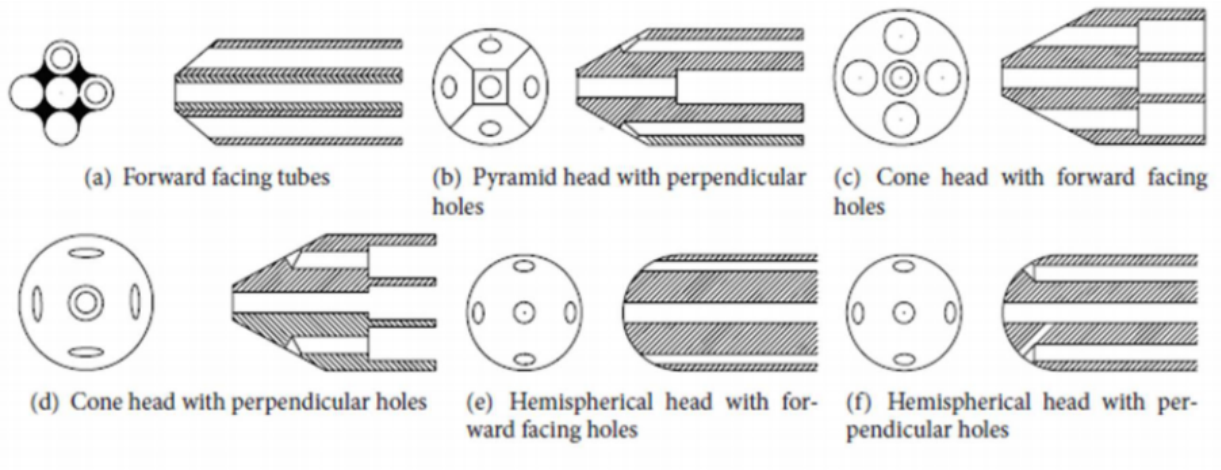

Figure 11: Types of Five Hole Probe Tip Geometries[8]

In addition, their geometry needs to be optimized for each different range of Reynolds numbers they need to operate in, prohibiting one probe from working across all flow regimes. Because of these short comings five hole probes aren't able to be widely implemented into a weather sensor platform and are instead mainly a useful upgrade to pitot tubes on fixed wing aircraft to grant additional functionality.[25]

2.1.7 Flush Air Data Sensors

The flush air data sensors (FADS) were developed for situations where traditional pitot probe aren't viable. This can be for a myriad of reasons ranging from super sonic flow applications to stealth aircraft. FAD systems are a multi-hole differential pressure sensing system. The number of pressure ports they use and their exact configuration vary depending on the

Figure 12: FADS system on the nose of NASA's F/A-18 Systems Research Aircraft[7]

application. They are typically an integrated systems of an aircraft used to collect air speed and attitude data. They can be seen on several advanced aircraft such as the Northrop B-2 Spirit and NASA's F-18 based High Angle-of-Attack Research Vehicle (HARV). Similar to other multi-hole probes they use multiple holes in a known configuration to be able to take airspeed and 3D limited directional measurements. They are a relatively new technology that is still being heavily researched. They have the potential to be extremely useful sensors but their effectiveness is closely tied to their integration into the vehicle and the aerodynamics associated with it. Similar to the five hole probe, while the FADS system is a promising sensor for measuring airflow, but its specific implementation means it isn't helpful for specific

meteorological applications.[22]

2.2 Pressure Based Systems

2.2.1 MEMS Based Cylinderical Sensor

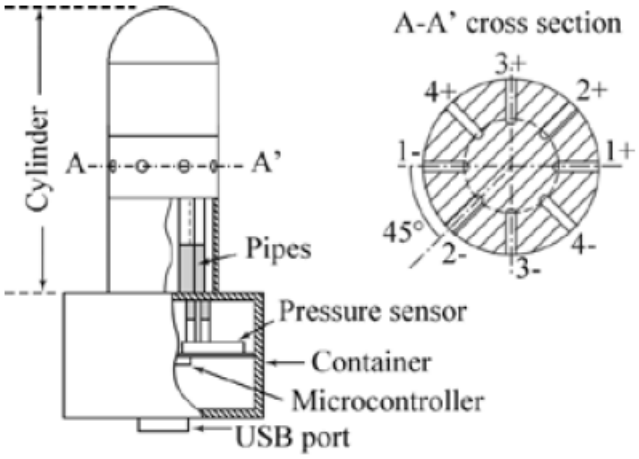

Figure 13: Muilthope probe based on MEMS Sensors[15]

As mentioned, the use of differential pressure sensing to monitor wind velocity is not a new concept. It will be helpful to look at what has previously been done in the area. Microelectromechanical system (MEMS) based transducers have been used before to build a similar system to the one purposed in this paper. This concept has show that it can be effective at measuring wind speed and direction. This MEMS based sensor used eight pressure ports around a metal cylinder where the differential is taken across each 45° pair. The plots in figure 14 show the error of this system for flows coming from each direction.

With generally under 3% error in both speed and direction this system proves that differential sensing can be used for this application. The MEMS system does fall short in some areas however. It is not optimized for severe weather can could fall prone to similar

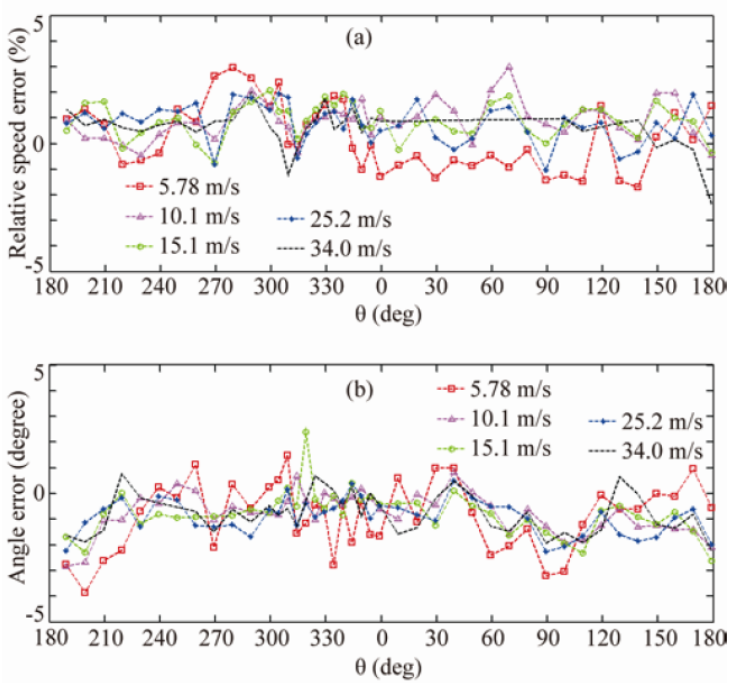

Figure 14: Error Across Directional Range of MEMS based Anemometer. Sensors[15]

issues that pitot probes face. In addition the probes all metal probe design is requires precision machining, making it difficult and expensive to build. [15]

2.2.2 Extreme Turbulence Probe

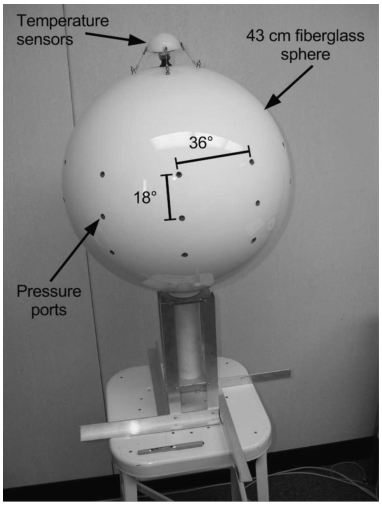

Figure 15: The Extreme Turbulence Probe. Sensors[9]

The Extreme Turbulence(ET) probe was a 43 cm fiberglass spherical differential pressure probe. It used 30 pressure ports placed around the sphere with differential transducers to find wind speed and gust parameters. A specific design point of this system was to study turbulence in severe weather. This design worked well and was able to capture both wind speed and find several gust parameters. It also had an integrated system to use compressed air to clear the pressure ports using compressed air. Aside from its inability to measure wind direction it also suffer from being extremely cumbersome. The probe was very large and could only log data to a desktop computer. This meant that it's setup required a full desktop computer to be used and powered in a hurricane. [9]

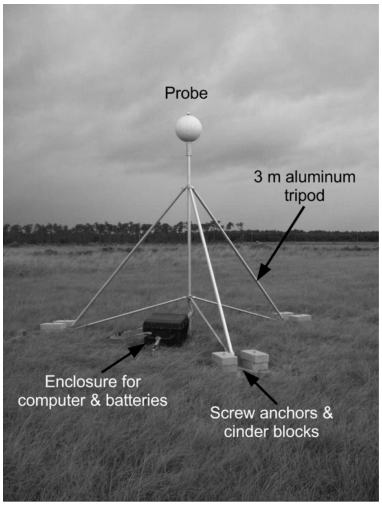

Figure 16: An Extreme Turbulence Probe Field Setup. Sensors[9]

2.3 Current Systems for Atmospheric Observation

This section will discuss systems for atmospheric observation on a slightly larger scale. These systems are general tools that are available to meteorologists and can gather a variety of data, not just wind speeds. They will work in a multitude of different environments with each having its own strengths and weaknesses.

2.3.1 TOtable Tornado Observatory

Figure 17: The TOtable Tornado Observatory[5]

The TOtable Tornado Observatory (TOTO) was a one off weather station built by the National Oceanic and Atmospheric Administration (NOAA) Environmental Technology Laboratory (ETL) and the National Weather Service (NWS). It was used from 1979-1987 so monitor severe weather. TOTO was a 250-300lb barrel that was outfitted with a variety of weather instruments. These included mostly anemometers, pressure sensors and humidity sensors. To deploy TOTO, it would be rolled out of the back of a customized pickup truck using metal wheel ramps. TOTO then had to be tipped to its vertical position and oriented facing north for accurate wind direction readings. It was outfitted with several different

sensors during its lifetime but generally used cup anemometers and weather vanes for wind speed and direction. It was placed into the path of a tornado near Ardmore, Oklahoma in 1984 but unfortunately was tipped over by the wind. TOTO was an early attempt to measure severe weather that brought to light a lot of the difficulties of taking measurements in these extreme conditions.[5]

2.3.2 Weather Towers

Figure 18: Mesonet Tower in Goodwell Oklahoma

Weather towers are a staple for data collection to support both weather prediction and research. Once set up they offer a platform to run a wide range of sensors that can be run continuously and have their data accessed remotely. Weather towers can operate inde-

pendently of one another or configured to work as part of a network of towers that allow meteorologists to access data from across bigger areas easily. For this paper we will mostly be discussing towers that are associate with the Oklahoma Mesonet. The mesonet is a network of 121 weather towers that are spread across the state and spread to ensure there is at lease one in each county. Each tower is outfitted with a suite of sensors and is configured to take and upload measurements remotely every five minutes.

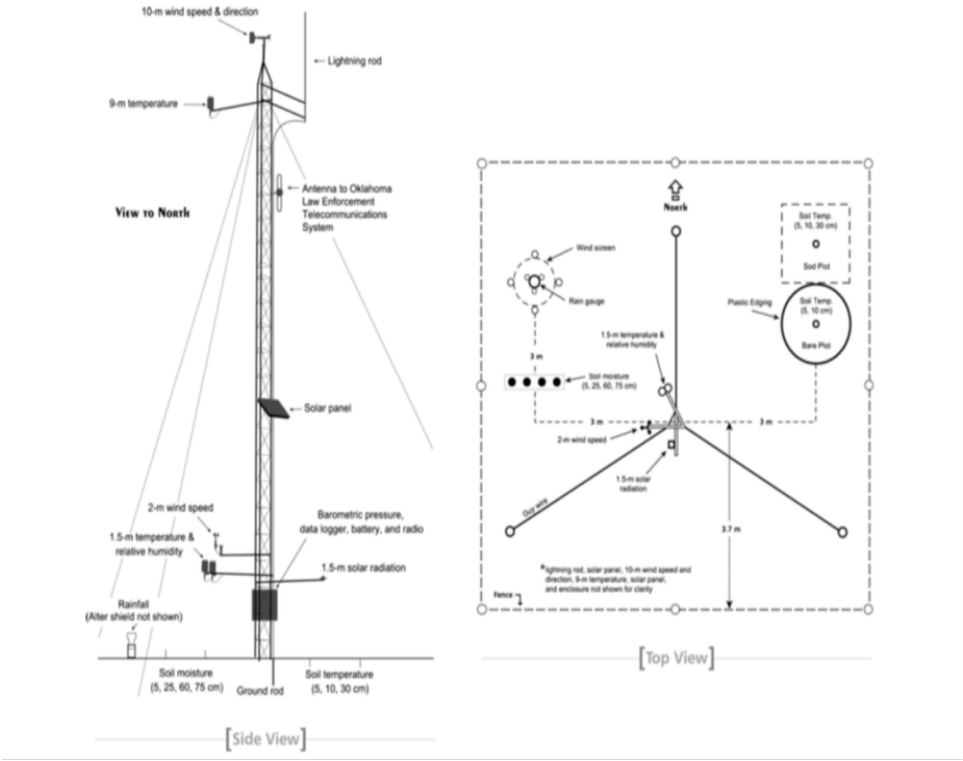

Figure 19: Standard Layout for Oklahoma Mesonet Towers[16]

Figure 19, shows the standard layout of a Oklahoma Mesonet tower. The towers are 10 meters tall and support instruments to take wind speed and direction, temperature, pressure, humidity, rainfall, solar radiation, as well several different soil quality readings. Each tower actually takes two wind measurements, one at 10 meters and one at 2 meters. At the top of the tower a R.M. Young 5103 anemometer is employed with a R.M. Young 3101 anemometer

being used 2 meters from the base. They are designed to remotely send data a centralized serve and be resistant to severe weather.[16]

2.3.3 Weather Balloons

Figure 20: USRI Student Researchers Filling a Weather Balloon

Another one of the most common tools for data collection in meteorology are weather balloons. Since they were created in the 1930's they have been one of the easiest and most accessible way to take measurements across the entire vertical profile of the atmosphere. They generally consist of a latex or synthetic rubber balloon that is filled with hydrogen or helium attached to a sensor system known as a Radiosonde that is then released to float up into the atmosphere. This allows them to take measurements on the way up until the balloon pops, usually at around 115,000 ft. The balloon's payload is able to measure

pressure, temperature, and humidity, all while being able to be tracked using either GPS or radar. This data over the entire vertical profile is able to be used to create a Skew-T plot, a quintessential tool for surveying atmospheric conditions.[1]

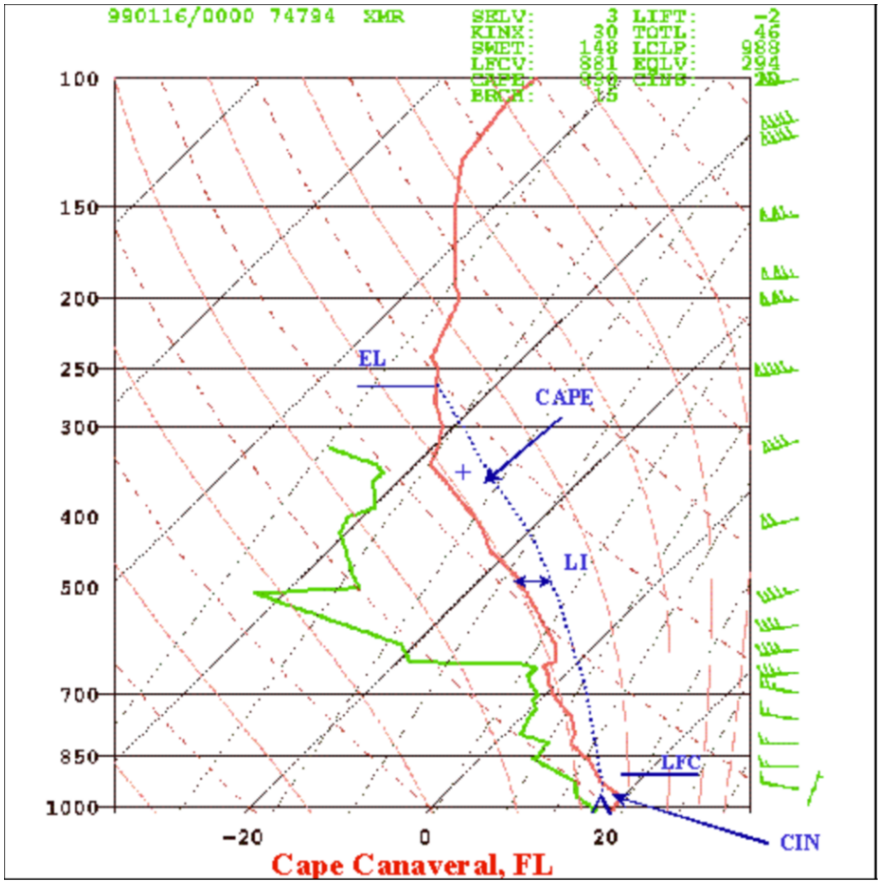

Figure 21: Skew-T Plot showing the results of a sounding[1]

2.3.4 Mobile Mesonets

Mobile mesonet stations are similar to the mesonet towers that have be discussed previously. What sets them apart is that they are build into a vehicle. They were originally created by in 1992 by technicians as the National Severe Storm Lab (NSSL) branch of the National Oceanic and Atmospheric Administration (NOAA). There is no standardized design for mobile mesonet and several have been created with varying configurations. Typically however,

Figure 22: Mobile Mesonet used by the National Severe Storm Laboratory[21]

mobile mesonets carry an instrument suite capable of capturing nearly the same data as full towers: wind speed and directions, temperature, pressure, humidity, rainfall, solar radiation. While they lack the height of towers they are able to collect data at rates high enough to support weather research. Unlike their tower counterparts, the entire purpose is to further meteorological research by collecting the right data in the right places. Mobile mesonets are generally used to gather research about severe weather and other atmospheric events that researchers wish to study.[**?**]

2.3.5 Manned Aircraft

Manned aircraft have always been an obvious choice for capturing atmospheric data but there were steep technical hurdles that originally prevented them from playing a large role in meteorological data collection. Most of these were overcome with the implementation of

Figure 23: A Multihole Probe Equiped on a NCAR C130Q Hercules[14]

inertial navigation systems (INS) in the 1960s. These systems allowed for precise tracking of a planes movements and allowed for them to be corrected for in any measurements made on board the aircraft. Originally this system required the processing of data a lengthy process, requiring more labor and therefore money. Over time data processing has become more manageable and helped to reduce manned aircraft's biggest drawback, cost. These have provided important data pertaining to large scale meteorological events such as severe weather.[14]

2.3.6 Unmanned Aerial Systems

Unmanned Aerial Systems (UAS) are an exciting technology being used for atmospheric observation. First used in 1970, the application of modern UAS it atmospheric sensing allows for an unparalleled amount of flexibility.[11] Most systems used for weather measurement

Figure 24: DJI Matrice 600 outfitted with a Young 8100 and aerosol sensor[4]

don't require the size of a fully manned plane, thus equipping them on a UAS provide many benefits. Firstly, UAS are cheaper to operate in almost every scenario than manned aircraft. This means lowers the barrier to entry and allows researchers to collect much more data with the same budget. Secondly, be it fixed wing or multi-rotor, UAS are able to maneuver more precisely and operate in ways manned aircraft simply cannot. This helps researchers ensure their able to get the right data in the right places. Finally, they're able to do this all without risking human life. UAS are able to be operated remotely, ensuring operators have more distance between them and hazardous conditions such as severe weather.

CHAPTER III

OVERVIEW OF DIFFERENTIAL PRESSURE SENSING

3.1 What is Differential Pressure Sensing

The concept of observing fluid flow using pressure is by no means a new idea. Pitot tubes are commonly used for airspeed measurements in both aviation and wind tunnels, and newer applications like flush air data sensors have been in use since the 90's. These all work on the concept of differential pressure sensing. The basis of this concept relies on understanding dynamic pressure. Dynamic pressure, is the portion of a flow's pressure associated with it velocity or kinetic energy. Since it is directly related, dynamic pressure can be easily used to find the speed of a flow if density is known using equation 3.1.1.

$$P_{dynamic} = \frac{1}{2}\rho V^2 \tag{3.1.1}$$

$$P_{total} = P_{dynamic} + P_{static} \tag{3.1.2}$$

The only obstacle is that there is no practical way of truly directly measuring the dynamic pressure. Instead, it can be calculated based on other pressure measurements using Bernoulli's equation. The static pressure is the opposite of the dynamic pressure; it is the pressure that is not associated with the flows velocity. It can easily be measured by taking a pressure measurement of the flow that is free of influence from the flow's movement. This is

usually achieved by having a pressure port either somewhere entirely outside of the flow or one oriented tangential to the flow direction. The total pressure (also known as stagnation pressure) is, as the names suggests, the total of all the pressures associated with the flow and is the sum of the static and dynamic pressures. Similar to static pressure, total pressure can be measured directly. This is done but putting a pressure port directly into the flow. This forces the flow to come of to complete stop (known as stagnation) inside the port and converts the kinetic energy of the flow entirely into pressure that adds to the static pressure and is then measured. The relationship of these three pressures is the basis of differential pressure sensing.

Based on this concept there have been several instruments created over time that can pseudo-directly measure differential pressures and thus can be used to find the dynamic pressure without having to individually find static and total pressures. Possibly the first example is the U-tube manometer. This device connects the ports to each side of a U shaped tube filled with a liquid of a known density. The difference of those pressures then cause the fluid to move up one side of the U. The difference in height of the columns of liquid can then be measured and used to directly calculate the differential pressure. While quite simple, U-tube manometers are still commonly used to this day to measure the airspeed in wind tunnels.

Today, the most common type of differential pressure sensors are strain gauge pressure transducers. These transducers expose a diaphragm to a different pressure on each side and then measure the strain of the diaphragm to calculate the differential pressure. These sensors are convenient as they are able to efficiently produce directly digital measurements to relatively high degree of precision.

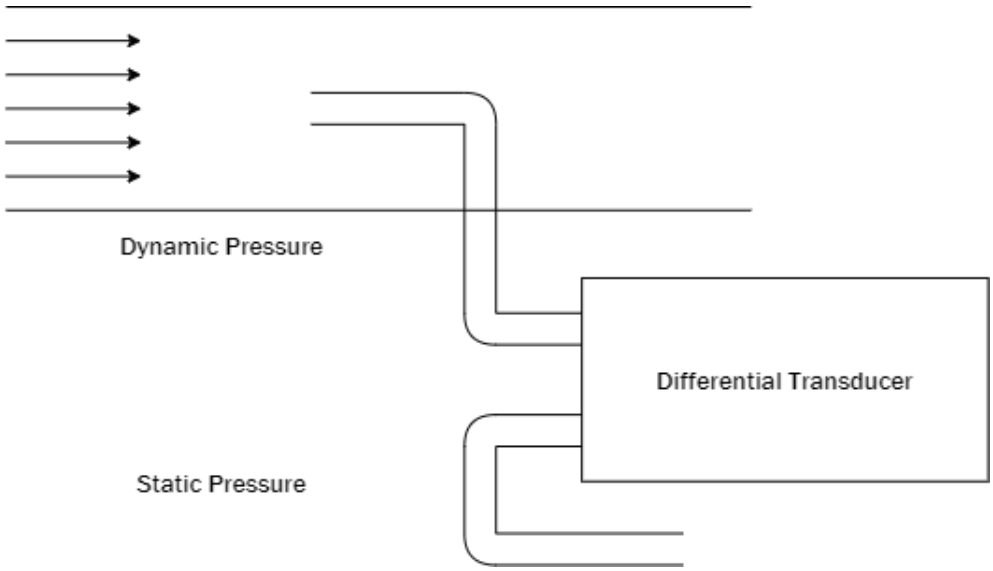

Figure 25: Basic Differential Pressure Configuration for finding Airspeed

The dynamic pressure of a flow can be found by taking the difference between the total and static pressure measurements, then that dynamic pressure is used to calculate a flow's velocity. This is the foundation of differential pressure sensing.

3.2 Current Applications

3.2.1 Pitot Tubes

Pitot tubes are probably the most straight forward application of differential pressure sensing. They employ a tube with a pressure port in the middle to measure they total pressure and a ring of pressure ports around the outside of the tube that are all connected internally to measure the static pressure. These ports are connected to a transducer (sometimes known as an airspeed sensor in aviation) that measures the dynamic pressure. As previously discussed this can then be used to directly calculate the velocity using a combination of equations 3.1.1 and 3.1.2 as shown in equation 3.2.1.

$$V = \sqrt{\frac{2(P_{total} - P_{static})}{\rho}} \tag{3.2.1}$$

3.2.2 Five Hole Probes and Flush Air Sensors

FHP and FADS systems are both a step up in complexity from the traditional pitot probe. Both of them use the base differential pressure concept to find airspeed but incorporate additional pressure ports to allow for limited directional flow measurements. Both also generally allow for 3D directional measurements which can be extremely insightful. A downside to these multi-hole probe concepts is that the math behind them can vary widely due to the geometries associated with their designs. For 5HP this is the design of the probe, specifically its tip geometry. With FADSs this involves the specific integration and the vehicles aerodynamics as a whole. While the specific application of the concept can vary widely in detail there are some basic concepts though that hold true for any differential pressure based sensor.

3.3 360° Differential Sensing

So far we have discussed strengths and weaknesses of existing wind measuring instruments and the basic theory behind differential pressure sensing and current multi-hole probe devices. The rest of the discussion in this paper will be focused around the creation of a new instrument based on these concepts. 360° differential sensing is a novel application of differential pressure sensing to create a new type of multi-hole probe.

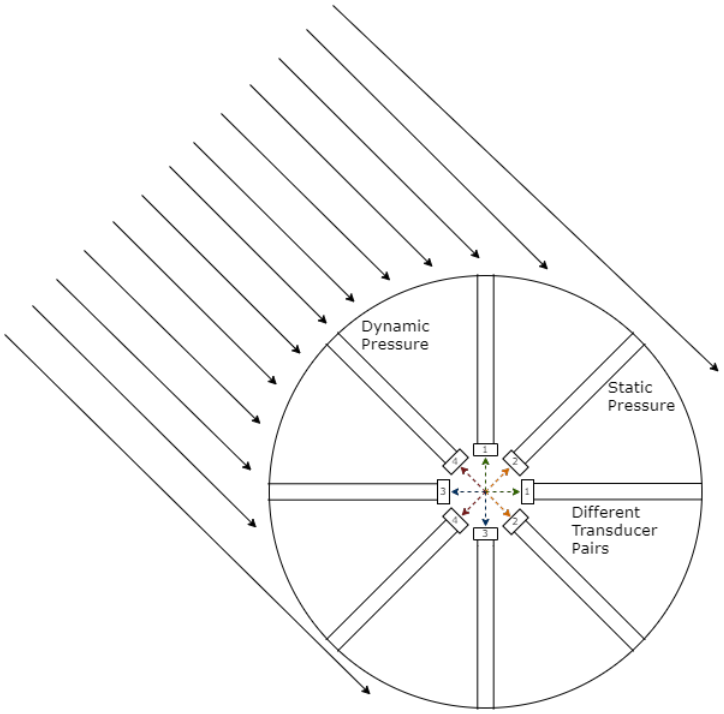

Figure 26: 360° Differential Pressures Concept

3.3.1 Concept

The concept of 360° differential pressure sensing is based on the fundamentals used in multi-hole probes the pitot probe, 5HP, and FADS but inspired by their inability to take omni-directional measurements. This concept overcomes those shortcomings by using a series of pressure ports oriented in an outward facing ring. This idea allows for wind velocity to be taken using the differential pressure between the port most aligned with the flow and the port most tangent to it. Speed can then be calculated using the same dynamic pressure relationship as discussed before using those two ports. Direction calculations can made using multiple methods, the easiest of which can estimate direction by looking at which ports have the highest and lowest pressures. They port that is most aligned with the wind will have the highest pressure. So if we know the position of each port we can then say the flow

is originating in the direction of the port with the highest pressure and is most tangent. This is because it will the closest be measuring the stagnation pressure which based on Bernoulli's equation will always be the highest pressure associated with a flow. Like wise the lowest pressure will be the one most tangent to the flow. Since all ports are exposed to the atmosphere they will measure the static pressure but the one most tangent to the flow will measure closest to the true the static, which according to Bernoulli's equation will be lower than thew stagnation pressure. It is also important to note that these two ports will always be at 90° angles to one another if there are an even number of ports and they are radially equally spaced (since the port direction with the lowest pressure will always be tangent to the one with the highest). This method is extremely simple but isn't without drawbacks. The resolution of directional measurements using this method without interpolation is directly tied to the number of pressure ports used as this method can only give directions that precisely line up with each port. Any direction between the port positions will be measured as the port direction it is closet too. This means that the directional resolution given is directly tied to the number of ports each probe has. While this isn't ideal, this method will be used for initial testing for the sake of simplistic initial calculation and directional interpolation or other calculation methods will be discussed later on. This problem does also bring to light the next drawback of this probe design as a whole.

This method of measuring the wind velocity requires a high number of pressure transducers. Using this method means that each port will need its own transducer. This is a problem inherent to all multi-hole probe designs but is exaggerated with the 360° concept because the pressure port configuration must cover a larger range of directions. This can be minimized however; based again on Bernoulli's equation. Since we know that we are only interest in the

ports with the highest and lowest pressures, and that those ports will always be at 90° angles from one another, we could use one differential transducer to measure the difference between ports at 90° angles. This means the transducer with the highest magnitude of measurement is measuring the flows dynamic pressure. Then, whether the differential pressure is positive or negative will determine which of the two ports the flow is aligned with and which it is tangent to. This method yields the same results but allows for one transducer to monitor two ports, reducing the number of transducers need in half. It also eliminated the need for a universal static port for the differential transducers to use as a reference.

3.3.2 Direction Calculation Techniques

As mentioned before, wind direction measurements are done by determining which transducer is reading the higher differential pressure. After this, the sign of the measurements will indicate if the flow is most aligned with. Positive differential reading will mean that the air is most directly flowing into the port connected to the positive input on the transducer while a negative reading would mean the opposite. This method is extremely simple and effective. It's main downside is that it's resolution is limited by the number of ports used. In a configuration of equally spaced ports covering a full 360° circle, the resolution (noted as R) is giving using equation 3.3.1 where n is the number of pressure ports used.

$$R = \frac{360}{n} \tag{3.3.1}$$

This makes the calculation of accuracy related to the method relatively simple. Because the ports are equally spaced the accuracy is the same in every direction. Since this method assumes the flow is perfectly aligned with one of the ports the amount of error from this

method depends on how far in between the ports the true wind direction is. The point with the maximum error will be where the true flow direction is exactly halfway between two of the ports. This means that max absolute error (e_{maxabs}) of a measurement is exactly half of the resolution.

$$e_{maxabs} = \frac{R}{2} \tag{3.3.2}$$

This absolute error can then be used to find the error relative to the range of measurements with the following:

$$e_{maxrel} = \frac{e_{maxabs}}{360} \tag{3.3.3}$$

These these three equations can be applied to the purposed design that uses 16 pressure ports ($n = 16$).

$$R = \frac{360}{16} = 22.5$$
$$e_{maxabs} = \frac{22.5}{2} = 11.25$$
$$e_{maxrel} = \frac{11.25}{360} = 0.03125$$

This means that theoretically using that using this method without accounting for error in the transducers there a maximum absolute error of 11.25° and a max error relative to the entire measurement range of 3.125%. This is outside of the goal of ±5° for directional accuracy based on the requests of the CLOUD-MAP researchers. However, this method will yield sufficient data for initial tests and can be improved upon using interpolation or entirely different methods that are discussed in further sections.

Alternate Direction Calculation Methods

A large shortcoming of the data presented in this paper is the low resolution in directional measurements from the basic calculation method. Using this method allowed for a larger scope of design and testing during this thesis but before the CMHP is able to take research grade measurements, a better algorithm for finding direction from pressures needs to be created. A benefit of the data logging setup is that it records raw transducer signals that are then converted to measurements in post processing. This means that any data set collected can be reprocessed in the future with improved algorithms. This section will discuss the need for a better algorithm and then purpose one such method that could be implemented in the future.

It is obvious that the method for directional calculations needs to be improved because of it's low resolution but essentially the same simplistic method is able to to be used for velocity without as many drawbacks. This can be explained by the examining the surface pressure coefficients from flow around a cylinder.

Figure 27 shows a plot of surface pressure coefficients at different points along the surface of a cylinder. Curves A, B, and C show real flow at different Reynolds numbers around a cylinder and the solid line shows the theoretical values if it was modeled as a potential flow. The surface pressure coefficient, C_p, is a ratio of the difference in surface pressure on the cylinder and the stagnation pressure of the flow to the dynamic pressure of the flow. This can be found using either equation 3.3.4 or the version given in Figure 27.[19]

$$\frac{C_p = P - P_\infty}{\frac{1}{2}\rho U_\infty^2} \tag{3.3.4}$$

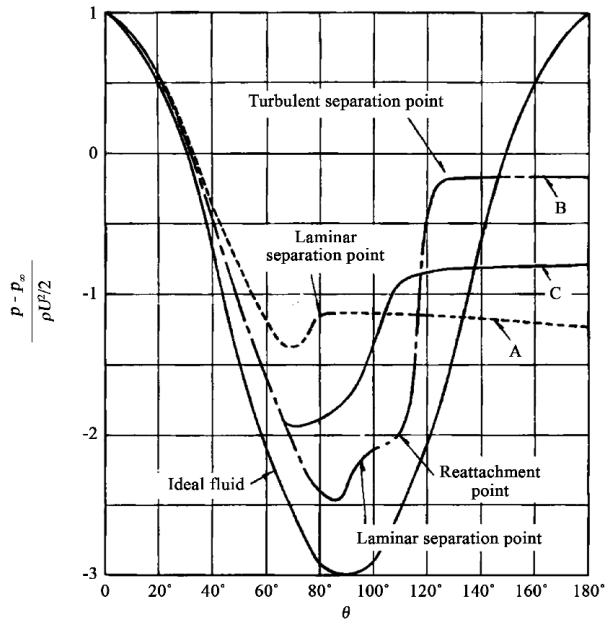

Figure 27: Surface Pressure Coefficients for Flow around a Cylinder [17]

The theoretical values from modeling this as an ideal potential are unrealistic because eventually the flow will detach from the cylinder and the pressure coefficients will never rise back up to 1 so instead the given equations should be used. The point at which the flow will detach is dependent on the Reynolds number of the flow. On this plot, $\theta = 0$ is the stagnation point where the flow first contacts the cylinder. The stagnation point will always have a C_p of 1. Since this is the pressure being measured for velocity calculations, by assuming the port is always aligned we are assuming the C_p of this point is always equal to one. While this assumption isn't true, it is close enough to get accurate wind speed measurements. This is because with 22.5° spacing the most a port can be dis-aligned and still have the max pressure is 11.25°. We can see that on this plot, for every curve, there is very little change in C_p from $\theta = 0$ to $\theta = 11.25$. If extreme accuracy is required in wind speed measurements this change should be accounted for, but the amount of accuracy required for that is beyond what is covered in the scope of this paper.

These this relationship between C_p and θ could also be useful in finding the wind direction. One purposed method of doing this is to match the C_p values measured around the probe to those one a known curve. This is complicated slightly as since we are measuring differential pressure between the points on the cylinder. To combat this we can manipulate equation 3.3.4 to find ΔC_p.

$$\Delta C_p = \frac{P_0 - P_\infty}{\frac{1}{2}\rho U_\infty^2} - \frac{P_{90} - P_\infty}{\frac{1}{2}\rho U_\infty^2} \tag{3.3.5}$$

Here, P_0 is the pressure at a point on the cylinder and P_{90} is the pressure 90° from that point. Since the difference between these pressures is what is measured by our transducers we can re arrange the equation to be in terms of that. We can also cancel out the sstagnation pressure which is helpful.

$$\Delta C_p = \frac{P_0 - P_{90}}{\frac{1}{2}\rho U_\infty^2} \tag{3.3.6}$$

We can also use the fact that $P_{dynamic} = \frac{1}{2}\rho U_\infty^2$ to further simplify the equation since we are using the assumption that the highest pressure differential is the dynamic pressure.

$$\Delta C_p = \frac{P_0 - P_{90}}{P_{TRmax}} \tag{3.3.7}$$

Now that we have this relation we can tell what the differential should be at any port on the cylinder if it was perfectly aligned. This is a tool we can use to find the direction of the flow.

First use the velocity that was calculated using the basic method that was described previously. From this velocity the Reynolds number of the flow can be found using the

probes diameter as a characteristic length and using the standard viscosity of air. The Reynolds number will then allow us to find the C_p curve for this flow. This can be done by interpolating between a number of curves at different Reynolds number or by simply choosing the closest one. After that the ΔC_p values can be taken from this plot to match what the differentials should be from the ports on either side of the port we are assuming has a C_p equal to 1. Next, find the ΔC_p values for those ports actually are using the differential pressure measurements and equation 3.3.7. Next compare these ΔC_p with what they should be. The direction calculated using the basic method can then be corrected using the difference between the expect ΔC_p and the actual ΔC_p values.

This method should greatly improve both the resolution and accuracy of directional measurements of the CMHP. It is a requirement that either this method or another like it is used to improve directional measurements before the CMHP can be helpful in collecting real meteorological data.

CHAPTER IV

INSTRUMENT DESIGN

4.1 Design Requirements

It is important to establish design requirements that will ensure the instrument produced is able to actually helpful for the meteorological research. Goals were established that would help ensure the instrument's relevance. These goals were as follows:

- Simple to operate in high stress or dangerous environments
- Competitive with the monetary cost of comparable sensors
- Capable of variable data collection rates
- Capable of accurate collection rates high enough to measure winds gusts
- Operate in severe weather environments (rain, extreme winds, hail, etc.)
- Durable enough to operate while sustaining impacts from debris

These items are meant to help keep the design focus of to producing a new instrument capable of collecting high rate scientific data inside of severe weather systems that is still able to operate at the capacity of a standard weather prediction anemometer. While these items are helpful to keep the design centered on the overall goal, more specific metrics need to

be established to have meaningfully targets to hit on a more technical level. For this, it can be helpful to look at what weather researchers themselves are targeting for data collection. Shown below in Table 1 are the desired sensor accuracy for different weather parameters targeted by the CLOUD-MAP project.

National Weather Service Desired Sensor Specifications

Measurement Variables and Accuracy		**Sensor Response Time**	
Temperature	±0.2 °C	Time	<5s (<1s preferred)
Relative Humidity	±5%	**Operating Conditions**	
Pressure	±1.0 hPa	Temperature	-30 to 40 °C
Wind Speed	±0.5 m/s	Relative Humidity	0 to 100%
Wind Direction	±5 Degree Azimuth	Wind Speed	0 to 45 m/s

Table 1: Suggested Benchmarks for Meteorological Data Collection [10]

The CLOUD-MAP was a federally funded large scale collaborative project that focused on using modern technology to study the atmospheric boundary layer (ABL). This is the exact kind of research the new instrument should be targeting to support so these requirements are perfect targets. For wind measurements this means we need to target and accuracy to within ±5 degrees for directional measurements and ±0.5 m/s for speed accuracy. These scientific level standards and can be challenging to achieve outside of a laboratory environment. The instrument also needs to be able to operate in a temperature range from -30 to 40°C, humidity range from 0 to 100%, and wind speed range from 0 to 45 m/s. These environmental environments span from standard everyday weather to lower level extreme weather such as tornadoes and hurricanes. Because this project aims to help with specifically extreme weather measurements these values will be a priority to be as high as possible but this table

gives a good minimum requirements. Using these requirements will help ensure that the product of the design process will actually able to fill the discussed data gap and be helpful to researchers.

4.2 Probe Development

Figure 28: Cylindrical and Spherical Probe Designs.

Based on the previously discussed concepts and requirements two prototypes were constructed. The cylindrical probe (shown above on the left) is designed to use the basic 360° differential pressure sensing concept to take 2-D wind velocity measurements. It is the primary design proposed and evaluated by this paper. The spherical probe (shown above on the right) is a prototype showcasing potential application of the concept to taking 3D flow velocity measurements. It utilizes eight pressure ports in a spherical configuration to theoret-

ically take 3D measurements in a near-omni-directional fashion. While more analysis would be required to develop a more complex algorithm to take accurate measurements, conceptually this probe would use the same basic multi-hole differential pressure concept to find 3-D velocity measurements. Due to the need to have someway of mounting the probe, measurements made in a cone aligned with the mounting shaft below the sensor suffer from decreased accuracy. Theoretically the measurements could still be taken however their accuracy may suffer unless corrected.

Figure 29: Spherical Probe Internals

The sphere was created using two 2.5in O.D., 0.04in thick hollow steel spherical domes. To create the pressure ports, brass tubes were soldered to holes drilled in the sphere in the designed pattern. 1/8 inch tubing is then able to connect to these brass tubes to carry pressure to transducers. Additionally, holes were drilled vertically in the center of each dome. The holes allow for each dome to mount onto a hollow internal shaft with a bolt that holds them in place. This internal shaft extends out of the bottom of the sphere and

attaches to the mounting shaft. The hoses then run through the inside of this interior shaft out of the sphere and to their transducers. This probe was created as a initial prototype and technology demonstrator. While this probe is theoretically functional with only eight ports it is unknown how much accurate it could be without more advanced velocity calculating attempts.

The rest of this paper will be focused on the development and testing of the cylindrical probe design. First the design of the probe itself will be discussed followed by the development of the supporting electronics. After this, initial data and calibration will covered and then finally data from additional tests will be analysed.

4.2.1 General

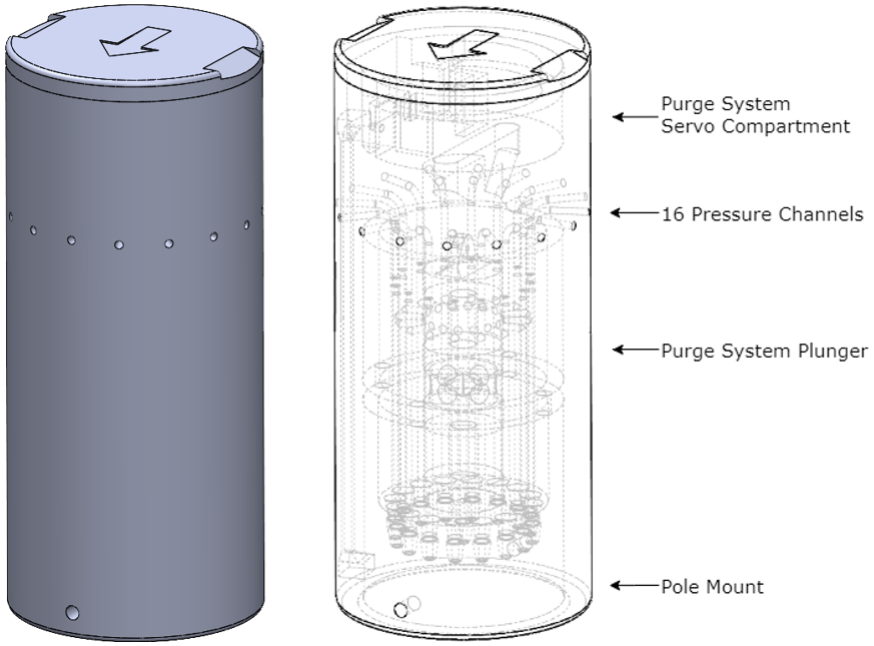

Figure 30: Cylindrical Multi-hole Probe Design

Shown above is probe design for the cylindrical multi-hole probe, full CAD drawings are given in the appendix. Starting with ease of use, the probe was designed to mount easily

on the end of any piece of standard 1.5in schedule 40 pipe with pressure hosing able to run inside of the mounting pipe. This allows for easy physical integration into anything from a tripod to a weather tower, or even unmanned aerial vehicles (UAV). The design includes sixteen, one twentieth inch diameter pressure ports that are routed down to the bottom of the probe where pressure hosing can be attached. The eighteen inch section of one eighth inch pressure tubing then connect the probe to a custom designed bank of eight differential pressure transducers. Using sixteen ports equally spread across a full circle means that direction measurements are able to be made using the direct calculation method discussed to a resolution of 22.5 degrees.

Pressure hosing friction fits onto nubs that are incorporate into the 3D-printed design. This allows for the hosing to be attached without any additional hardware being added. This connection method has worked successfully during all phases of testing and hasn't warranted any changes. Strain relief should always be with the pressure hosing to prevent stress on the connection but it is feasible that probes that are roughly treated or used in extreme environments could see the connection nubs break off. If this at any point begins to hamper the probes use the design allows for metal tubing to be inserted and glued inside of the nubs to reinforce them. This quick modification should give the nubs enough strength to resist breaking any reasonable strain caused by the pressure hosing and prevent the nubs from becoming brittle in extreme cold.

A key component of the purge system is an internal compartment that houses a servo motor that actuates an internal plunger. This compartment has a mount for a high torque metal geared micro servo and a slot leading to the probe's internal channel that allows for a servo linkage to the purge plunger. There is also an external pass-through for the servo's

Figure 31: Cylindrical Multi-hole Probe's Internal Compartment.

three pin cable. This servo cable is then routed down an inset passage along the outside of the probe to its base. The cable then enters a another pass-through that allows it to exit the bottom of the probe inside of it's mounting pole along with the probe's pressure tubing. The servo cable then runs to the CMHP's electronics

This design is compact, sturdy, and features no externally moving parts. It allows for pressure measurements to be taken in a variety of environments and also supports a purge system that allows for all ports to be cleared of water and other blockages.

4.2.2 Purge Mechanism

As discussed a large driving force behind the design of the cylindrical multi-hole probe was the need for a way to take measurements in extreme environments. A well documented

weakness of differential pressure sensing is that pressure port can become clogged rather easily. This is a persistent problem caused by precipitation, icing, and foreign object debris (FOD) that is frequently seen in aircraft pitot tubes. Similar issues are likely to be seen for differential pressure sensors put into severe weather environments do to the combination of high rain, high winds, and several sources of debris. This means that a way to address pressure port clogging became a requirement during the probes design. Firstly the pressure ports were made as small as was feasible to limit debris that could enter them. Next, a purge system was designed to allow clogs in the ports to be cleared remotely without having to remove the sensor from use.

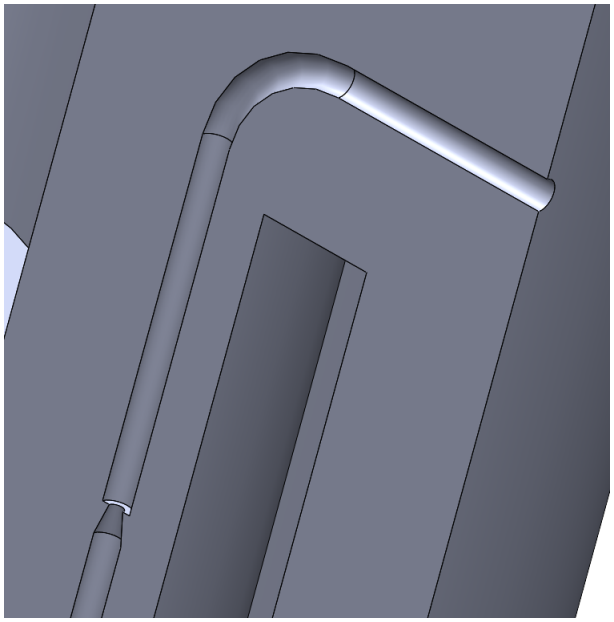

Figure 32: Cutaway view of a pressure port showing a debris trap

The CMHP's purge system uses two main features, a debris trap and an internal air release plunger. The debris trap sits around three quarters of an inch below the opening of the pressure port. It features a sharp change in the port's diameter that results in a ledge to keep debris near the opening of the pressure port. The trap (shown in Figure 32) the

bottom left reduces the pressure port's diameter by 50% has no effect on the pressure moving through the air, but will stop or slow most or all water or debris. This allows the second part of the system to then remove the blockage. The internal air release plunger is a plunger that travels up and down a central channel of the probe. This plunger is actuated by a servo motor that sits an internal cavity at the top of the probe, which can be seen in Figure 30. Halfway through the probe, each pressure port branches off to this central channel. The plunger has two different zones of it's exterior. The top and bottom sections have a rubber seal that press against the walls of the probe's central channel and form an airtight seal. The middle section of the plunger has is open to a channel running down it's own interior. The plunger's interior channel runs to a nub at it's base similar to the nubs on the pressure ports. The plunger's however, works in the opposite manner and is instead attached to a source of high pressure air.

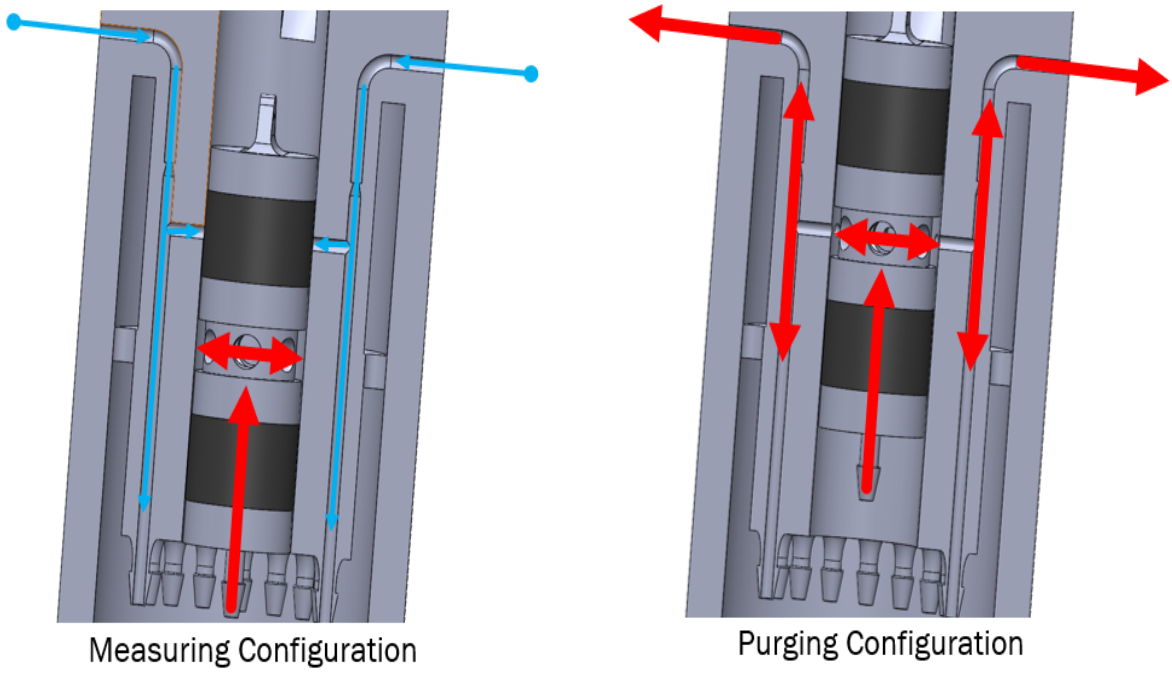

Figure 33: Purge Mechanism Diagram

When the plunger is lowered in the probes measurement configuration the rubber seal on the top section of the plunger isolates each of the sixteen pressure ports from each other and allows them to carry pressure from the air to the transducers. When the plunger is raised the seal slides away from the opening to the pressure ports and exposes them to the plunger's middle section. This pushes high pressure air backwards through all sixteen of the pressure ports simultaneously. Shown in Figure 33, this exposes any blockages to the full force of the supplied high pressure air. This should be able to push any debris stuck in the debris trap back out through the opening of the port and unclog it.

This design is not without downsides however. The pressure ports teeing off to the central channel allows for overall simplicity but means that the transducers attached to the pressure ports are also exposed to the high pressure air; which limits the pressure that can be used without damaging the transducers in the event that debris perfectly seals the opening of the pressure port. This however should not pose major issue however as the transducers used, 5 inH_2O Honeywell HSC series differential transducers, have an over pressure rating of 300 inH_2O (about 10.8 psi). This should be ample pressure to remove any blockages. The other issue caused by this system is that if any FOD makes it through the debris trap and past the T junction, the pressure will force it deeper into the system resulting in it passing into the pressure tubing and possible the transducers. This would likely effect all pressure readings in that port until the pressure hosing is manually removed and cleared of blockages. This shouldn't happen as most FOD should get trapped near the opening or the pressure port but if it is found this happens often then the debris trap could be resigned to combat the problem. Changing the "ledge" style debris trap to a "S bend" style design should easily improve the traps effectiveness at the cost of some added complexity. Lastly which purging,

the CMHP is unable to take accurate measurements. This is unfortunate but will have little impact on performance as the purge system should only be activated periodically or if obviously errant pressure readings are detected.

This purge system addresses the challenges of the extreme weather environment and should ensure the successful operation of the CMHP in such environments. The internal plunger system was shown to work in simple informal tests at 10psi but it's effectiveness at actually removing the debris was not evaluated. While it is an important part of the system's overall design, only the testing of the 360° differential pressure concept is covered in the scope of this thesis. All testing discussed in this paper was done with a simplified version of the CMHP where the pressure ports are not connected to the central channel and thus isolated from the purge system.

4.2.3 Manufacturing

The probe design makes use of relatively complex internal geometries to allow for it's overall compact size. These internal structures would make it difficult to build using traditional manufacturing methods. For that reason it was decided to create the CMHP using stereolithography 3-D printing. stereolithography (SLA) printing used a laser to harden layers of resin that form the final product. This method is especially great for complex geometries as each layer is supported by the surrounding liquid resin as it prints. This helps prevent deformation that can occur with more common filament disposition modeling (FDM) 3-D printing methods. This also makes the manufacturing process relatively simple as parts can made directly from a CAD file and has the added benefit of the final product being watertight.

Figure 34: A Semicircular Aeration Hole Located below the Pressure Ports

The use of SLA methods for manufacturing lead to a few notable to additions to the design. First there was a hollow cavity added inside the outer wall of the probe. This hollow area reduces the amount of resin needed to make each probe, the amount of time it takes to print, and make the finished probe weigh less. Second there were a number of aeration hole added to the holes. These holes allow air to enter the internal geometry as the print processes and prevents the occurrence of "cupping". Cupping takes place when a SLA print is raised out of the pool of liquid resin as it's printed if the print contains cavities that will not be open to the atmosphere at any point in the printing process. As these cavities are raised out of the pool they for a vacuum since outside air isn't able to fill the newly created empty spaces. The vacuum can then pull uncured liquid resin up into these areas that causes problems when the part is post processed which effects the finish of the print and can even

lead to uncured resin being trapped inside of the finished part. These holes also allow air to enter the cavities as the print to stop a vacuum from forming. These holes also allowed liquid resin to drain from the hollow areas that were incorporated to reduce the amount of material used. An example of an aeration hole is shown in Figure 34.

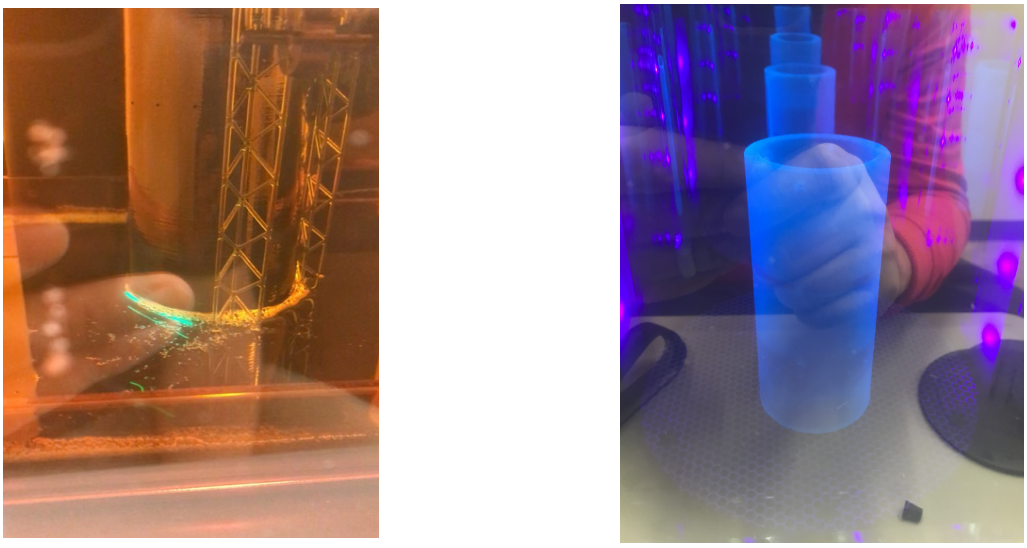

Figure 35: **Left:** SLA 3-D Printing **Right:** UV Curing Post Process

The probe was printed in SLA tough resin which provides strong material properties to resist physical damage. The print was then post processed to archive a final product. The probe was washed in a bath of 99% isopropyl alcohol where ultrasonic waves are used to make sure any remaining uncured resin is dissolved. The washed partially cured print was then put in a UV cure chamber to fully harden. The result was a fully formed and ready to use cylindrical multi-hole probe.

4.3 Electronic Development

This design using the 360° differential pressure concept requires the use of eight differential pressure transducers. This is much more than is typically used for other applications with

pitot tubes only using one and five hole probes using three. Because of this, there are very few options to get the required supporting electronics off the shelf. After searching for options the only commercial off the shelf (COTS) solution was banks of pressure transducers meant to operate a suite of pitot tubes across an entire wind tunnel. These systems do have enough transducers but since they're meant to be built into a wind tunnel they are much to large and heavy to be used for field measurements. This means to make the CMHP function during operations in real world environments custom supporting electronic had to be developed. For the initial testing covered later in this paper, pressure measurements were done using a Scanivale DSA3217 pressure scanner. This allowed for the concept to be proven before custom supporting electronics were full built. This section details the design and manufacture of the support electronics that were required for field testing.

4.3.1 Sensor Selection

Differential pressure transducers are the basis of the 360° probe concept so selecting proper transducers is of the utmost importance. For this iteration it was decided to use TruStability Board Mount Pressure Sensors made by Honeywell. Specifically the Honeywell TruStability High Accuracy Silicon Ceramic (HSC) Series was identified to be suitable for the project. These sensors use a piezoresistive silicon membrane to that is able to convert pressures directly into voltages. The HSC series offers transducers that are calibrate from -20°C to 85°C and calibrated for temperatures from 0°C to 50°C. This means they can meet the upper end of the required 40°C range identified from the design requirements listed in Table 1 without any other considerations needed. They do fail to meet the requires on the lower end however. The -30°C requirement is below both the standard calibration and operating

Figure 36: HSCMRRN005ND7A3 Differential Pressure Transducer

ranges. Calibration issues can be fixed by performing additional calibrations for the lower temperature range and the self-heating of the rest of the electronics will keep the transducers in their operational range in colder environments. If necessary a heater could also be added to the electronics compartment to further allow for operations in the cold.

They also output digital measurements at approximately 1kHz. This speed is much higher than what is required and will not even come close to slowing down the overall sampling rate of the CMHP. Specifically the HSCMRRN005ND7A3 sensor was selected. This is a surface mount differential pressure transducer with a pressure range of ± 5 inH_2O and 1% error over that range. Taking those numbers and using Equation 3.2.1 we find that at standard temperature and pressure (STP) this corresponds to ± 2 m/s ($\sim$4.5 mph) rate of error and can measure up to a max wind speed of 45.5 m/s ($\sim$101.7 mph). This meets the requirements for maximum wind speed but the error falls short of the desired 0.5 m/s accuracy. This isn't ideal but these transducers can use an auto-zero calibration to reduce their error rate to 0.5% to get closer to the requirements and the standard 1% accuracy will be enough to prove the concept of this probe. These transducers then communicates its output over digital I^2C protocol.

4.3.2 Printed Circuit Board Design

Electrical Design

The decision to use I^2C communication protocol allows for keeping keeping communication simple since it can allow all eight of the transducers to communicate with the data logger with just two shared connections between all of them. This brings to light the leading design challenge for the electrical design of this project. I^2C protocol works based on each sensor having a unique I^2C address. The problem is these addresses are built into each transducer and the manufacture only offers each transducers with 2-3 different I^2C address and due to shortages it was difficult to even get eight transducers with the same pressure range regardless of their I^2C address. To combat this, it was decided that eight transducers with the same address would be used and an I^2C address translator would be used to shift each address in order to deconflict each transducer. This custom printed circuit board (PCB) will be designed to allow all eight transducers and their respective I^2C translators to interface with a PJRC Teensy 4.1 that will be used as a data logger. In addition it needs to provide for an external power source to power the entire system and a way for other external sensors to be integrated into the same Teensy 4.1.

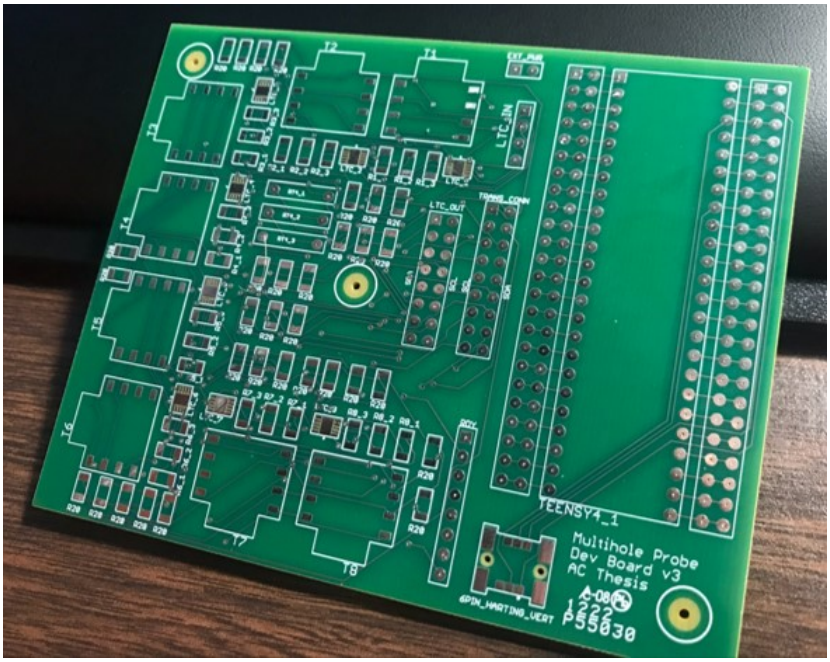

Figure 37: Multi-hole Dev Board v3 with Manufacturing Errors.

The PCB was designed as a two layer board with trace connections on the top and bottom layers and all components being on the top layer. The board utilized a combination of surface mount devices and through hole connections. The board went through three design iterations before arriving at the final design. Version 1 was iterated upon during the design phase and almost scrapped entirely due issues with it's component layout. It's initial layout resulted in the routing of traces being impossible. Version 2 improved upon this by moving several of the resistors supporting the I^2C address translators to the outer edge of the board and reducing both the minimum trace width and spacing. The layout change allowed for several of the longer traces to route along the outer edge of the board and reduced congestion in the middle of the board. The tracing changes helped by allowing traces to be more densely packed and route better through the gaps between through-holes.

The smaller width does impact the circuit however. Both the width and the thickness

of the trace impact how much current is can transfer without melting. Version 1 used a standard 10 mill trace width which is rated for 1 amps of current, but as the circuit is expected to experience less than 0.2 amp version 2 was able to be safely changed to use 8 mill traces [2]. The reducing trace spacing can lead to the current moving through the traces causing emf interference in the signal of each trace but this design only uses digital signals which are resistant to this kind of interference. This second board was sent of to be manufactured however upon their return it was discovered a dimensions mistake had been made in the design files the rendered them unusable. This was corrected for the third and final version. The first shipment of these boards however were stricken with manufacturing errors. The mistake (shown if Figure 37) was that a machine had been tooled incorrectly and each of the 1mm through holes were drilled to be 0.254mm. After this was corrected by the manufacturer, the version 3 boards were tested and found to work as expected. Board schematics for this final version can be seen in the appendix.

I^{2}C Address Translation

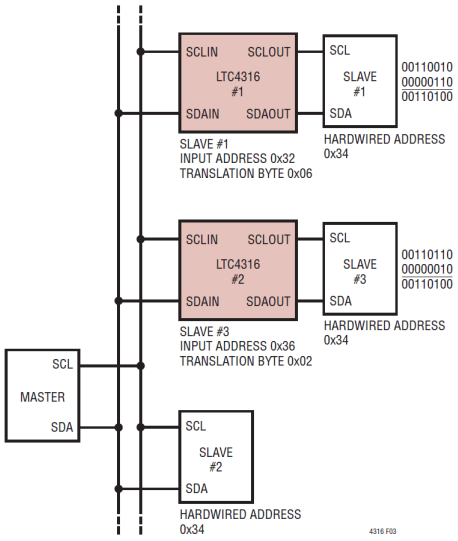

Figure 38: Example LTC4316 Setup

I^{2}C translation is handled by done by an LTC 4216. This is a small integrate circuit (IC) that is able to be placed on the I^{2}C clock and data lines between the slave device (transducers) and the master device (teensy 4.1) and convert all communications between the two to the proper I^{2}C addresses. The LTC works by using a voltage divide to set a translation byte. This byte is set when the LTC is powered on and then is added to the bytes of section of the commands and responses that contain the I^{2}C addresses. This process is shown in Figure 38. For our board, each LTC needed to be set to different translation bytes in order to make sure all eight transducers had different addresses. This was done using the wiring design showing in in Figure 39 where values resistor values R_1,R_2, and R_3 (shown in the figure as R4_1,R4_2, and R4_3) were changed for each transducer. The resistances used for each voltage divider are listed in appendix textcolorredadd appendix number and translation table with all pull-up resistors having a resistance of 2 kΩ.

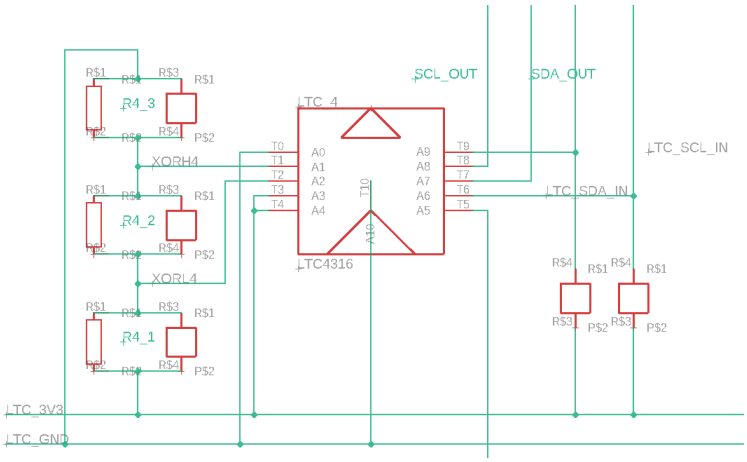

Figure 39: LTC4316 Wire Diagram

Development Board Features

The purpose of the boards build for this project are to test the concept and be a launch point for a final product design in the future as such the board was designed with several "development" features to help isolate any problems should they arise in testing. To make trouble shooting easier, each section of the board is separated from each other with traces going to header pins rather than the other terminal of it's desired connection. This means that connections can be made between different sections of the board with jumper wires. This allows for a lot of flexibility during testing because it means that every component can be easily rewired to different zones of the board while retaining any supporting pull-up resistors they need. For example, if a transducers isn't logging data it could be rewired to skip it's translator to see if it functions on it's own or even through a different translator that is verified to be working. The entire Teensy 4.1 that is used as a data logger can even be easily swapped out.

In addition some extra ease of use features were also added. A stand alone Harting flex

connector was added and traced to the main I^2C ports so that additional I^2C devices can be easily integrated into existing hardware. Extra header connectors were added to each pin of the Teensy that allow for easy integration of any other device that it can interface with. This feature was used to implement a rotary encoder over a serial connection for wind tunnel testing and the integration of a Pixhawk auto pilot that was used to pull GPS timestamps over Mavlink. This later addition will also allow for these electronics to be easily integrate into an existing Pixhawk controlled UAS and add any information onboard information from the autopilot directly into it's data logs. Finally there was a terminal added to allow the entire system to be powered externally by a regulated five volt power source. This was used to power the unit with a lithium polymer (LiPo) through a battery eliminator circuit (BEC) for all tests of the CMHP system.

4.4 Manufacturing

textcolorredadd stencil picture

Successfully soldering large amounts of surface mount components can be very difficult and time consuming if done incorrectly. To assemble our supporting electronics a stencil was created that had apertures that lined up with each surface mount pad on the board. The stencil can then placed on top of the bare PCB and when spread solder paste is spread across it the apertures allow the paste to apply uniformly across every pad on the board simultaneously. The design of this stencil is relatively simple however there are a few general rules that should be followed. The amount of solder paste placed on each pad is a function of the area of the aperture opening and the thickness of the stencil. If too much solder is applied it can short connections with the surrounding pads, if there is too little then there

won't be a sufficient electrical connection. The SMT pads for the LTC address translator are extremely small (0.889x0.305 mills); this means that the LTC will be very sensitive to a incorrectly sized stencil. Stencils need to maintain two ratios, the aspect and area ratios[3]. These rules are given in the following equations where L and W are the length and width of the aperture with T being the thickness of the stencil:

$$AspectRatio = W/T \geq 1.5 \tag{4.4.1}$$

$$AreaRatio = \frac{LxW}{(2(L+W)xT)} \geq 0.66 \tag{4.4.2}$$

A stencil was developed using these rules from 1 mill stainless steel. This was used to apply solder paste to all SMT pads. After this surface mount components were then individually placed by hand on their respective pads. Then the board and components are placed in an infrared oven. The oven is programmed to follow a specific temperature profile. This profile heats components hot enough to boil of the paste and melt the solder for it to re-flow but without overheating or melting any components. After this the boards are inspected and continuity tested to ensure every connection is good.

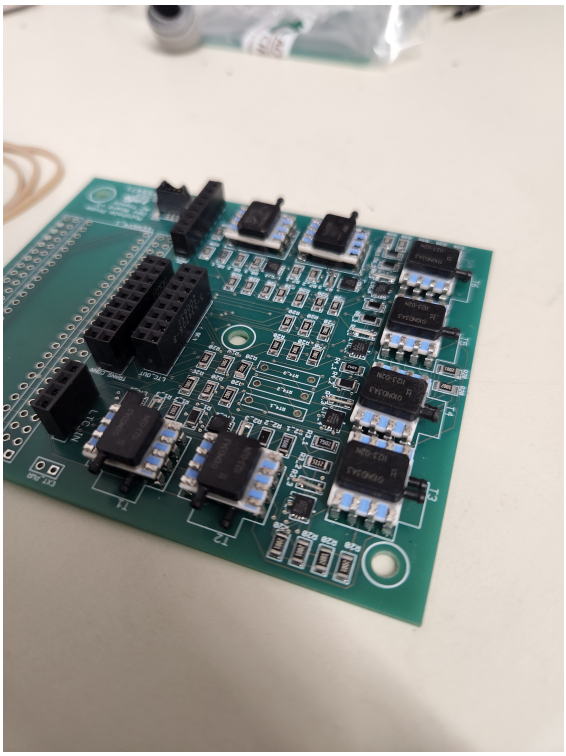

Figure 40: Fully Populated PCB

4.5 Multiplexers and Design Changes

During the initial testing of the completed electronics it was discovered that the LTC address translators could all be replaced by a singular I^2C multiplexer. The I^2C multiplexer acts as a electrically controlled switch that can be used to rapidly connect and disconnect to the signal wires of each transducer. This greatly simplifies the overall circuit and slightly reduces component costs. Because of the ease at which sections of the board could be rerouted transducers were rewired to pass their signal to the data logger through an Adafruit TCA9548A I2C Multiplexer Breakout. The TCA multiplexer was then soldered to a breadboard that was mounted to the existing PCB. This maintained all of the original functionality while simplifying the board and while reducing cost and the amount of SMT components that

needed to be soldered.

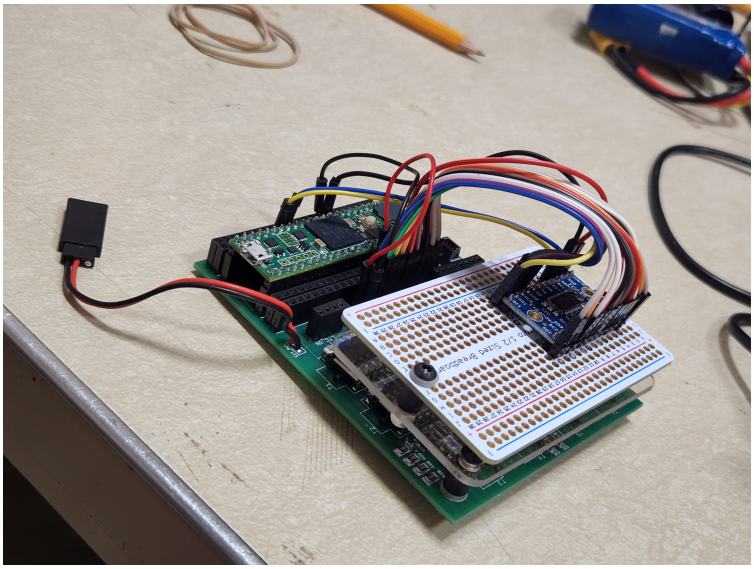

Figure 41: Finished Board with TCA9548

4.6 Calibration

Once the new electronics were created they needed to be calibrated. For simplicity these prototype transducers will be calibrated using a static calibration technique. The transducers were entirely disconnected from the pressure hosing and placed in a box that would ensure the transducers were exposed to no differential pressure. A data set of the raw signal output from each transducer was then taken over a period of five minutes. The signal output for each transducer was then averaged over that time. Then a static calibration coefficient was made using the average signal output. This coefficient can be found from equation 4.6.1 where S_{Pcal} is the signal associated with the calibration pressure (zero in this case), $S_{measured}$ is the average raw signal measured by the transducer, and C_{static} is the static calibration coefficient.

$$S_{Pcal} = S_{measured} + C_{static} \tag{4.6.1}$$

Since $S_{Pcal} = 0$, C_{static} can easily be defined as $C_{static} = -S_{measured}$. The found C_{static} values for each transducer are shown in Table 2. These values were found the be effective at calibrating measured pressures near the calibration point.

This static calibration helps make sure all of the transducers read the same value at the calibration point, 0 Pa differential pressure. The problem with this method is that as the measured pressure moves farther away from the calibration point it becomes less effective. This effectively means that as velocity increases the transducer's calibration becomes less accurate. This would be solved by using a dynamic calibration curve rather than a static calibration point.

Static Calibration Coefficients

Transducer	Coefficient
1	1029
2	1007
3	1022
4	1025
5	1014
6	1038
7	1022
8	1007

Table 2: Calibration Values

A dynamic calibration curve uses a calibration coefficient that changes based on the raw signal of each transducer. This allows the transducer be correctly calibrated over an entire range of pressures rather than a single pressure. The effects of not using a dynamic calibration curve has a clear effect on the data collected, especially at higher wind speeds, however the static calibration is suitable for proving the 360° concept work. A full dynamic calibration will be done at a future point in time.

CHAPTER V

INITIAL TESTING AND PROOF OF CONCEPT

5.1 Proof of Concept Testing

To help prove the 360° concept before time and resources were spent on fully developing it's support electronics some initial testing was done where the probe was setup to use a Scanivale DSA3217 pressure scanner to handle measurements and data logging. This DSA3217 uses a set of sixteen differential pressure ports that all measure pressures in reference to a single, shared reference port. In order to support the 360° concept each of these sixteen transducer ports was connected to one of the sixteen pressure hoses from the CMHP with their shared reference port being left open to the atmosphere. This meant the transducers acted more like absolute pressures measuring gauge pressure rather than differential pressures between the ports. While the the raw data output would differ slightly from the output of final data logger used in future testing, will have little impact as the differential can be calculated in the post processing of the data.

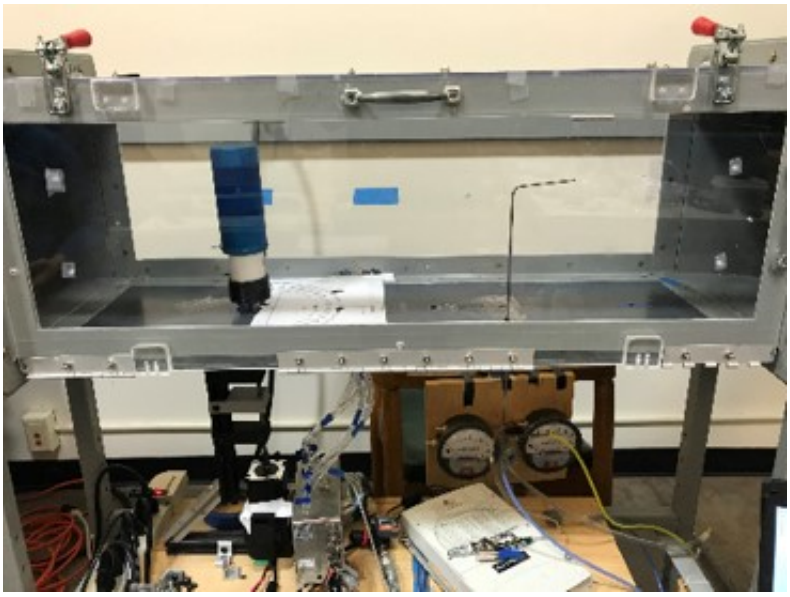

Figure 42: CMHP Initial Testing Setup

This initial wind tunnel testing was conducted in a GDJ FLOTEK 1440 wind tunnel. This To simulate changes in wind direction the probe was mounted on a stepper motor that could rotate the probe with respect to the flow in the tunnel. In addition a pitot probe was set up upstream of the CMHP that was attached to an analog differential pressure to monitor flow in the tunnel. This setup, shown in Figure 42, was use to run several trials of two different tests.

5.2 Initial Test Result

Static tests were run where the probe was rotated in 22.5° steps with ten seconds spent at each position after a full rotation the probe was then rotated back to its starting position in the same manner. This meant that each port was aligned with the flow for several second twice. The airspeed was was kept constant throughout entire range of motion and the wind tunnel was then turned off. Data collection was stopped once the tunnel's pitot tube read

zero differential pressure.

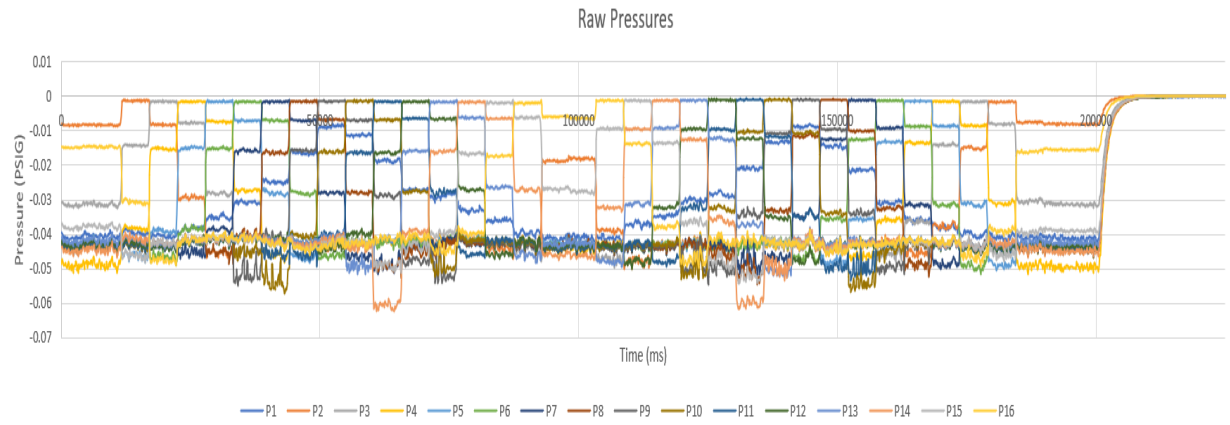

Figure 43: Static Test Raw Pressures

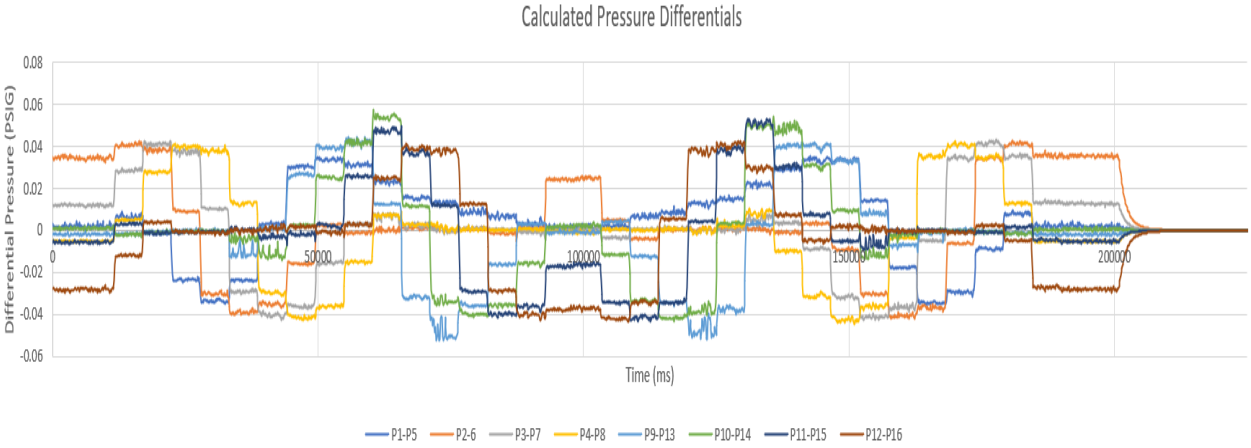

Figure 44: Static Test Calculated Differential Pressures

Shown are plots of the data from a static test at each step of processing to convert it from raw pressures to wind speed and directions. Figure 43 shows the raw pressures from each transducers. For this initial test data set these are the gauge pressures of for each of the sixteen pressures. Our final design uses eight differential pressures in 90° offsets, we can take the raw pressures can calculate the differentials manually. These calculated differentials are shown in Figure 44. These both of these plots are the first indication that our probe is

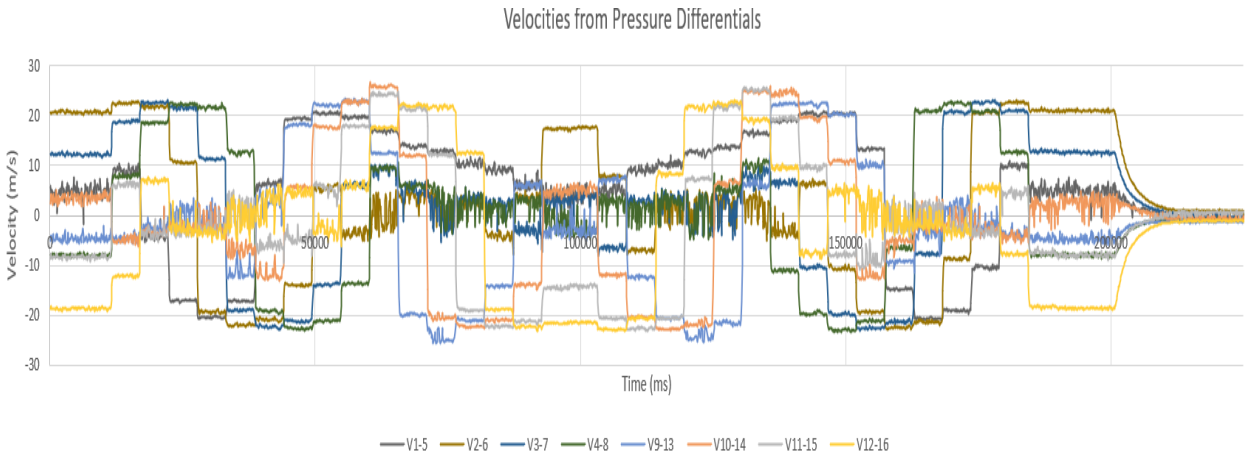

Figure 45: Static Test Calculated Velocities by Transducer

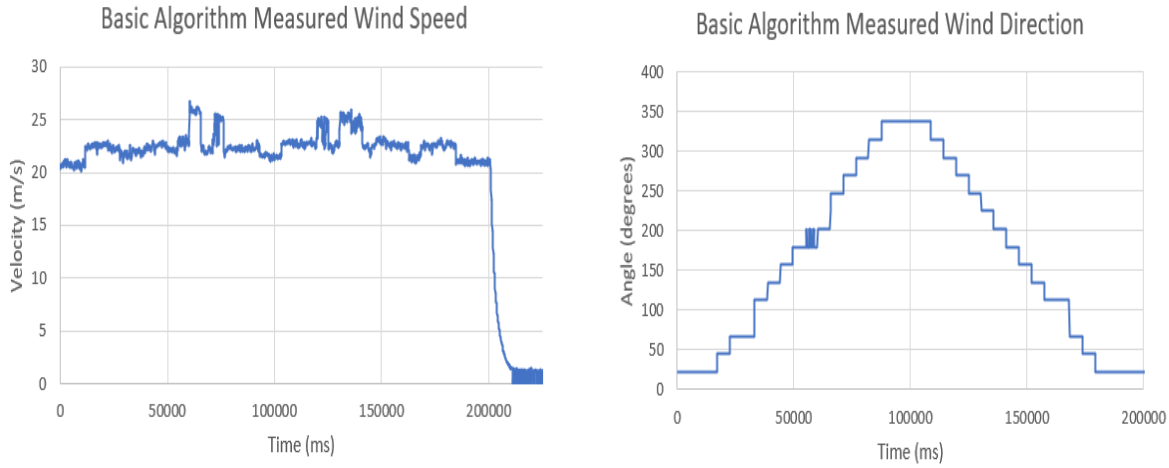

Figure 46: Cylindrical Multi-hole Probe's Initial Test Measurements.

working correctly. In Figure 43 we can see that as the probe is snapped to each position, the pressure of the port in that position raises to stagnation pressure of the flow.

CHAPTER VI

TESTING AND RESULTS

The completed CMHP was tested in three main types of trials. Firstly, the same initial wind tunnel tests were performed on the probe with its new electronics. These were conducted to collect data sets in a controlled environment where both the wind speed and direction are known and confirm the new data acquisition system (DAQ) is functioning properly. After wind tunnel testing confirmed the new electronics were working correctly, the systems was run in two types of field tests where data was able to be compared to existing systems.

6.1 Wind Tunnel Trials

The same wind tunnel tests from the probe's initial testing were repeated with the same probe using its new custom built DAQ unit. This included the same two static and dynamic tests that have been previously discussed however in addition to the probe data, data from a rotary encoder was taken to measure the probes actual position.

6.1.1 Static Tests

Shown are plots of the data from a static test of both the raw pressures as well as the calculated wind speed and directions. In Figure 47, the differential pressures from each transducer during the test are shown. These pressures behave as expected. As the probe

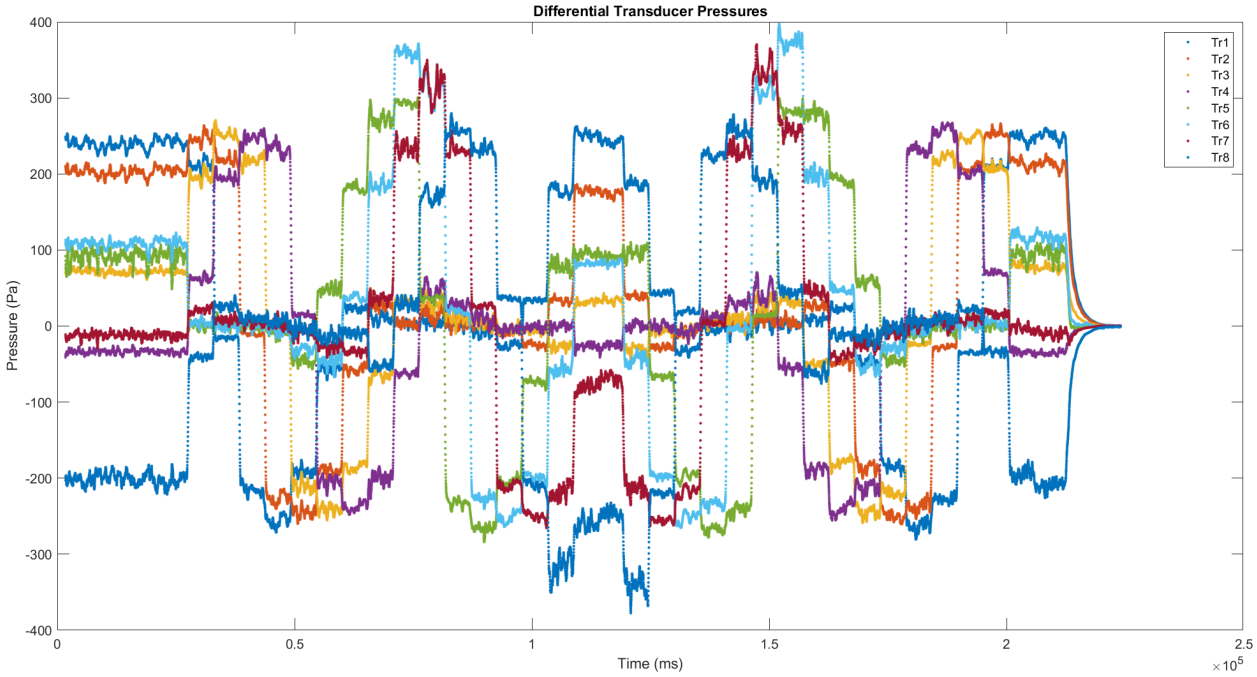

Figure 47: Static Test Raw Pressures

is snapped between positions the probe in each positron registers a rise in pressure up to the stagnation pressure of the flow. It is notable that some transducers (mostly six, seven, and eight) rise above the stagnation pressure when aligned with the flow. This is most likely explained by the lack of a full dynamic calibration. These transducers read more inline with their expected pressure values as their differential pressure gets closer to their static calibration point at 0 Pa. A dynamic calibration should increase the accuracy of these transducers along the entire operational pressure range for the probe. Figure 48 shows the calculated wind speeds based on the maximum observed differential pressures. Again this plot shows the expected behavior of a nearly constant wind speed with just a few variations. The most notable issues appear when the transducers with before-mentioned higher pressures become aligned with the flow. This causes the measured wind speeds to peak around 4 m/s higher than expected. As this is caused by the inaccuracies in pressure differentials from the

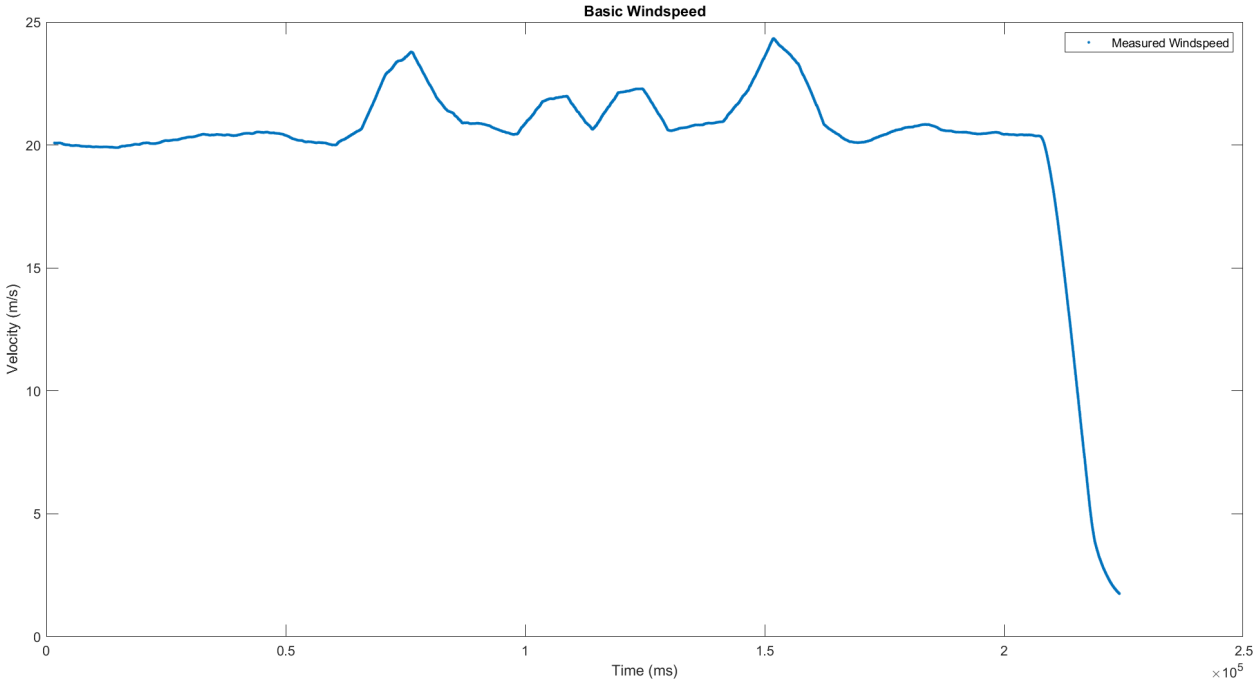

Figure 48: Static Test Calculated Velocities

transducers it is expected that these peaks should disappear after a dynamic calibration is performed. Next, Figure 49 shows the calculated wind directions using the basic method that was described earlier along with the measured positions from the rotary encoder. The probe's directional calculations form the low resolution stair step pattern that is expected from the basic directional calculation method. These measurements are actually an improvement over the results of the initial testing. Because all the transducers are working properly on the new electronics, there are no positions that are missed like what is in the initial test results. In this figure the probe measured wind direction is shown in blue and the positional data from encoder is shown in green. This encoder was originally planned to be used to empirically find the sensors response time, however it is clear that this system did not work correctly. We can see that there is an error build up on in the positional data. In this test, the probe is rotated at a high speed between positions. This movement could have either been faster

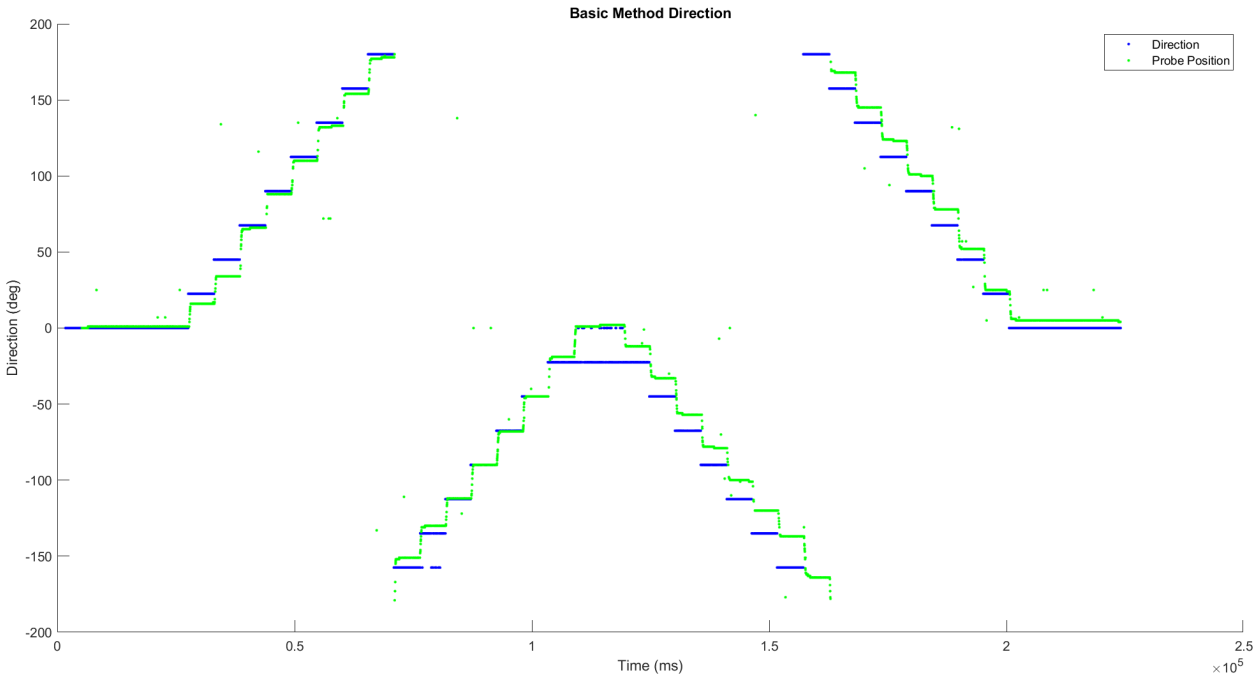

Figure 49: Static Test Calculated Direction

than the encoder was able to register leading to unmeasured movement, or been fast enough that it caused the linkage pairing the probe and the encoder to slip. This can be fixed in later tests by lowering the rotation speed between positions but does mean that we aren't able to reliably measure the response time of the probe. While there are some problems that could be corrected, overall this data proved that the system does work. In addition to the changes listed, further testing would also benefit from some other changes. If future testing is done in a wind tunnel with a larger cross section the probe could be free from wall effects of the tunnel. Future analysis would also benefit from accounting for the effects the tunnel probe has on probe measurements.

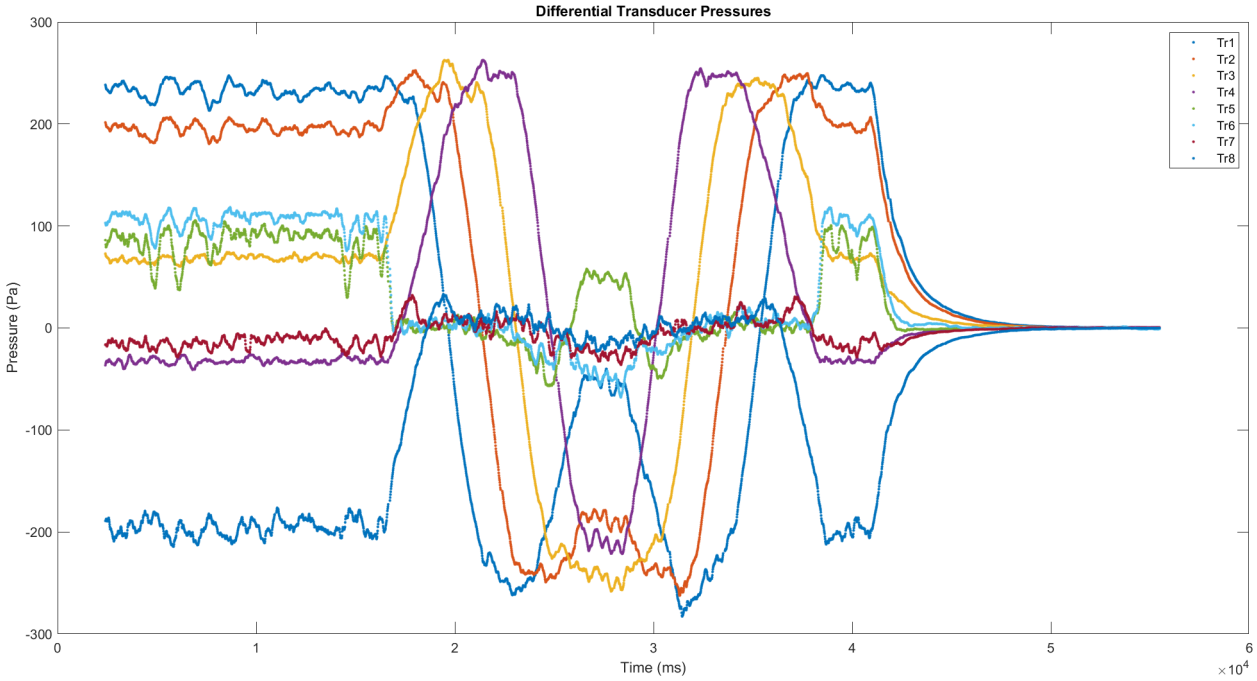

Figure 50: Dynamic Test Raw Pressures

6.1.2 Dynamic Tests

Similarly, the same dynamic test from the initial trials was run on the system using the new DAQ system. This test simulates a constant wind speed changing direction at a steady rate. Shown in Figures 50-52 are the test results from these trials. These results also match what we expected from initial testing. It also shows some similar issues to the static test results. The raw pressures again switch off matching the stagnation pressure as they come into and out of alignment with the flow. This time since the rotation is more gradual we see this happen in a series of overlapping arches rather than sharp jumps. Also notable is that in this smaller range of movement the transducers that appeared to be sensitive more sensitive to the lack of a dynamic calibration never become fully aligned with the flow. This means they won't cause inaccuracies in the calculated velocities.

Shown in Figure 51, the calculated velocities match our expectations of a nearly constant

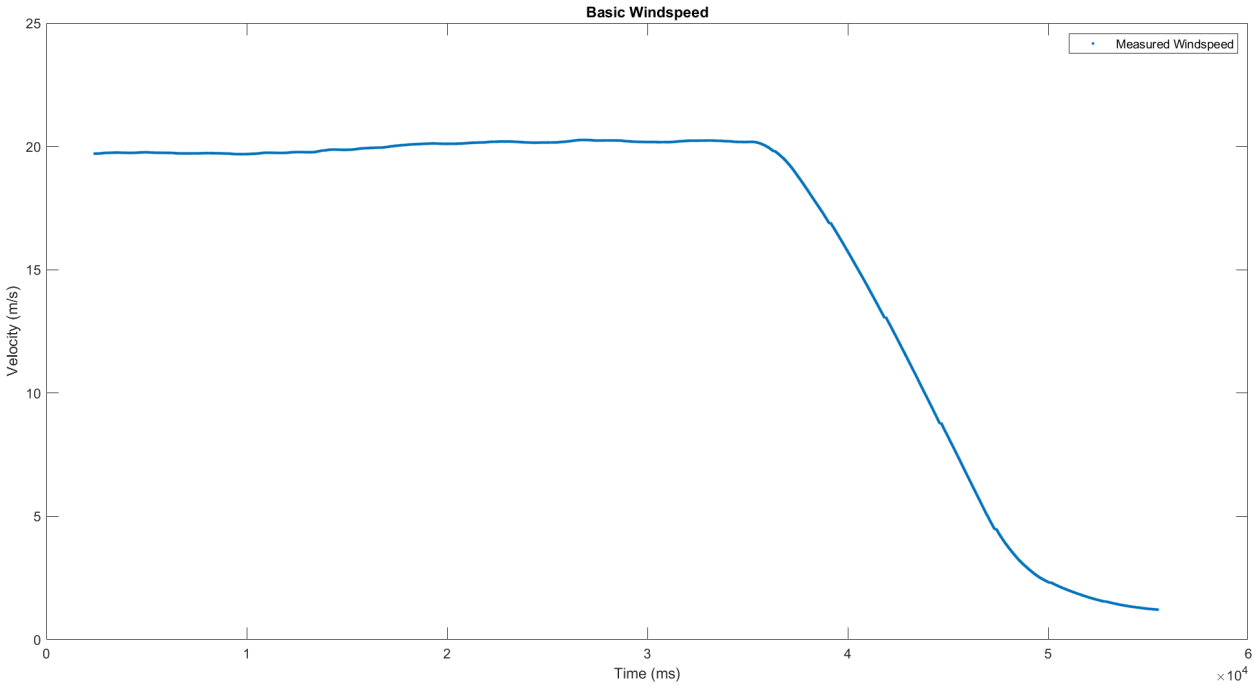

Figure 51: Dynamic Test Calculated Velocities

velocity even better than the static test data. As mentioned this plot is free from the unexpected peaks in velocity caused by the lack of calibration over the full range of speeds. While a full dynamic calibration would increase they accuracy of these velocities these results show that the methods used by the probe do work.

Next, Figure 52 shows the calculated direction. This shows similar results to the static test but does have a few notable features. We see a steady stair-step change in direction just as in previous data sets. An improvement however, is since the probe is rotated more slowly it appears that the encoder was able to more accurately measure the position throughout the test. The measured data is able to follow the movement effectively however it does emphasize the shortcomings of the basic direction calculation method. This method's poor resolution is really only useful as an approximation of direction. Before the probe is fully usable for weather measurements a better method must be implemented.

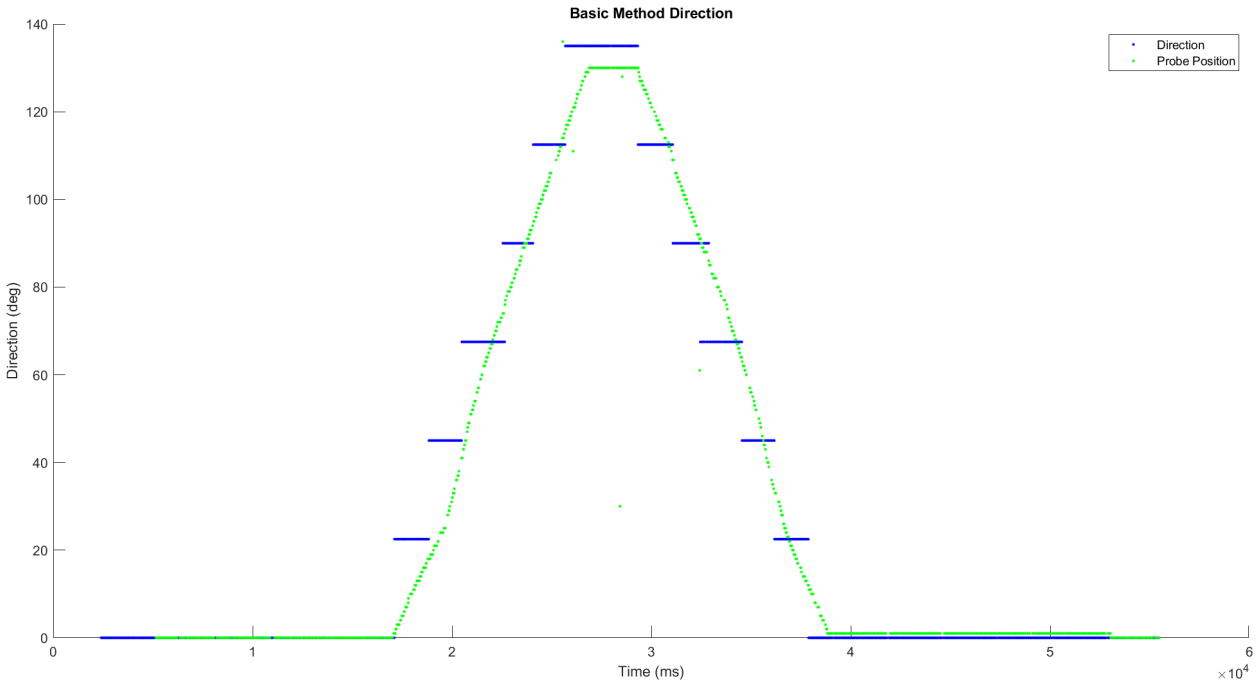

Figure 52: Dynamic Test Calculated Direction

6.2 Mesonet Tower Test

Once the wind tunnel testing confirmed that the system was working at a base level the CMHP was tested in the field against other comparable systems. The first of these test was a validation test to compare the data from the CMHP to the young 92000, the young 81000, and the Marena Mesonet tower. The Marena Mesonet is part of the the Oklahoma Mesonet Network. This is a network of one hundred and twenty mesonet towers used to take weather observations across the state of Oklahoma. The Marena Mesonet is located in Payne county, just south of Lake Carl Blackwell. It provides time averaged weather observations every five minutes.

To get comparable data data sets were collected with the CMHP and one of the Young anemometers were mounted on top of a Blue Sky Mast mobile tower and raised to a height of 10 meters. This puts these sensors at the same elevation as the anemometer on the mesonet

Figure 53: Test Tower set up next to Marena Mesonet Site

tower. All sensors were oriented so that a measured direction of zero degrees is due North to make comparisons easier. Two data sets over 30 minutes long were taken, one with each of the Young sensors alongside the CMHP and the Mesonet data. The instruments were also configured to use GPS to be able to timestamp data sets in unix time which allows for all captured data sets to be compared to one another and to the mesonet data. Unfortunately, due to issues with this setup, no accurate timestamps were not collected for the Young anemometers so their data cannot be meaningfully compared to the data from the other sensors.

The mesonet reports the average wind speed and direction over every five minute period along with the maximum wind speed measured during each period. To make the CMHP data more comparable with this lower resolution it needed to smoothed. The CMHP in its current setup does not log at a constant data rate, but does average around 75 hz. Based

Figure 54: CMHP and a Young 81000 on a Blue Sky Mast Tower

on this, each data point from this trial was smoothed using a moving median average over 10,000 points which will make each point the average over roughly the last 5 minutes of data. This is shown in Figures 55 and 56.

In Figure 55 we can see that the velocities match with the mesonet rarely differing more than 1 m/s and never differing from the average more than around 2.5 m/s. These points again suffer from the lack of a dynamic calibration but even so this data is less effected than the wind tunnel data as the experienced wind speeds are at a lower speed and closer to its static calibration point. With the implementation of a dynamic calibration we should see these magnitudes match up even better. Figure 56 shows the directional data from the same trial. This plot matches up almost exactly as over the time of the trial the wind direction stayed relatively constant. This plot still suffers from the poor directional resolution of the basic calculation technique but proofs that the wind direction can be accurately found using the CMHP concept in field data collection.

Figure 55: Smoothed CMHP Wind Speed Data vs Marena Mesonet

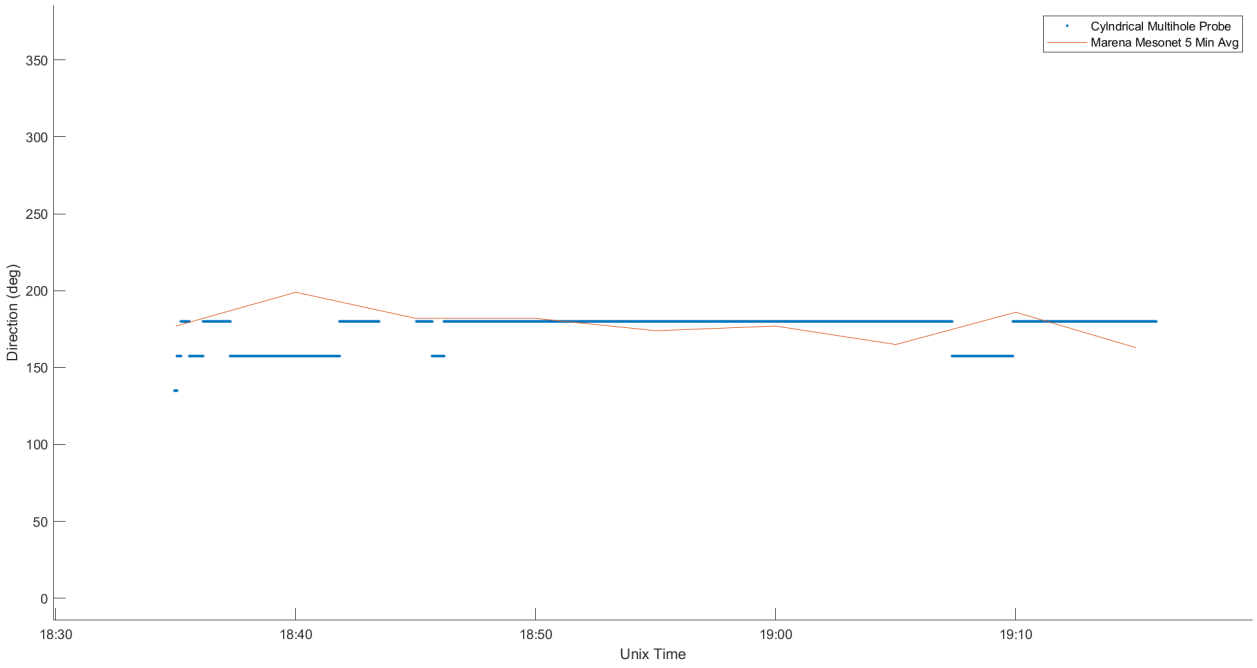

Figure 56: Smoothed CMHP Wind Direction Data vs Marena Mesonet

6.3 Mobile Mesonet Test

The final test covered in the thesis is a simulation of a mobile mesonet. In this test the CMHP is set up on top of a vehicle alongside Young ultrasonic anemometers. Each trial consisted

Figure 57: Vehicle Setup to Simulate Mobile Mesonet

of the CMHP being tested alongside one Young anemometer with trails run using the Young 81000 and the Young 92000. All instruments were oriented so that zero degrees of direction was from the front of the vehicle. Each trial consisted of the vehicle being driven through town and on the highway at a variety of speeds up to around 65 mph (29 m/s). Similarly to the mesonet tower tests, each sensor was setup to use GPS to timestamp data with unix time to make comparison easier. Velocity data from one of these trials is shown in Figure 58. This plot shows the downside of a static calibration extremely well. We can see that as the velocities go farther from zero (the static calibration point) the CMHP readings begin

to stray further and further from the Young measurements. Overall however this test shows that the CMHP is able to produce results that are very comparable to existing ultrasonic anemometers.

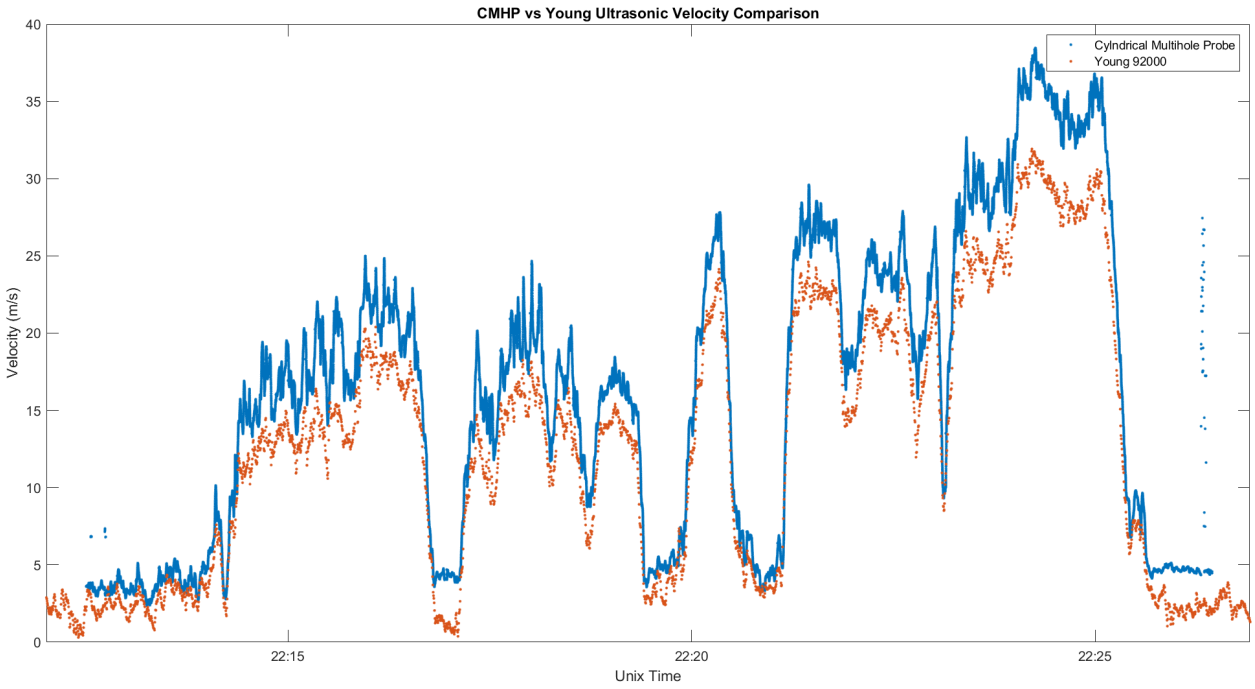

Figure 58: CMHP Wind Direction Data vs Young 92000 Mobile Mesonet

CHAPTER VII

CONCLUSION

This paper has covered a novel application of differential pressure sensing from concept, to design, to initial testing and evaluation. While it is clear that some areas still require significant improvements, it is also clear that the concept shows significant promise. The Cylindrical Multi-hole Probe has clearly shown that with refinement it could become a viable low-cost option for taking scientific grade wind measurements in severe weather environments.

7.1 Comparison to Existing Instruments

The CMHP shows promise, but it is important to look at how it compares to other similar wind measurement systems. Ultrasonic anemometers are the closest fit to the role of monitoring severe weather. Figure 59 shows several notable parameters for these instruments. As it currently stands, the CMHP falls short of the young anemometers in accuracy for both speed and direction, as well as resolution for directional measurements. As previously discussed, those short comings are mainly due to limitations of the first prototype and can be improved upon in future iterations.

Next is cost. The Young 81000 and 92000 are available off the shelf for $2,950 and $1,996 respectively. The CMHP prototype uses $625 in parts with the option of lowering the cost if

Parameter	Wind speed	Wind Direction
Range	1 to 45.4 m/s	0 to 360°
Resolution	0.002 m/s	22.5°
Accuracy	±2 m/s	±11.25°

Parameter	Wind Speed	Wind Direction	Temperature
Range	0 to 40 m/s	0 to 360°	-50 to 50 °C
Resolution	0.01 m/s	0.1°	0.1 °C
Accuracy	± 1% (0 to 30 m/s) ± 0.05 m/s (30 m/s to 40 m/s) ± 3% (<40 m/s)	± 2° (0 to 30 m/s) ± 5° (30m/s to 40 m/s)	± 2 °C (> 30m/s)

Parameter	Wind Speed	Wind Direction	Temperature	Relative Humidity
Range	0 to 70 m/s	0 to 360°	-40 to 60 °C	0 to 100%
Resolution	0.01 m/s	0.1°	0.1 °C	0.10%
Accuracy	± 2% (0 to 30 m/s) ± 3% (30 to 70 m/s)	± 2°	± 0.3°C (-20 to 50°C) ± 0.3°C (50 to 60°C)	± 2% (5 to 95%)

Figure 59: Comparison of the CMHP with other Anemometers

cheaper transducers can be sourced. This means it could realistically be sold for comparable prices profitably. This is extremely impressive considering the advantages of the CMHP.

Where the CMHP pulls ahead is it's robustness and ability to operate reliably in adverse conditions. Being able to capture high quality data in severe weather allows it to be extremely useful for those doing meteorological research. Overall, this means that the CMHP, while lacking in accuracy and resolution in it's current form, could prove to be a system that is able to capture comparable data for a comparable price in a much large range of environments.

7.2 Advantages and Shortcomings

The Cylindrical Multi-hole Probe holds several advantages over other sensors that are able to take high resolution wind measurements. Firstly, it has shown the concept to be refined to take data similar to ultrasonic anemometers, but cost significantly less than these other off the shelf options. The CMHP is also able to work in a wider variety of environments

and is significantly more durable than ultrasonic anemometers. In addition, it also has the ability to work for any fluid if it's transducers are swapped out. This means it could be used for more than just an anemometer and could be used to measure water currents as well. The design is obviously without its own pitfalls however. It relies on a high number of pressure transducers, which can be difficult to source. This reliance could also prove problematic as increasing the operational pressure range will have an effect on its accuracy. It could also be difficult to find transducers with high enough accuracy to improve the system's velocity accuracy by a useful amount. In it's current form, it's measurements are far less accurate than ultrasonic anemometers. For velocity measurements, the probe appears to only be lacking a full dynamic calibration, but to improve its directional accuracy will require the development of a better algorithm to convert pressures to direction.

7.2.1 Future Work

While the research covered in this paper was a good start, the concept will require much more work before it is viable as a push button replacement for existing systems. Firstly, the implementation of better wind direction algorithms is a must to be able to take data that is comparable to other systems. A full range dynamic calibration needs to be done for each of the transducers to ensure that pressure readings are accurate throughout their entire range. Additionally, further testing needs to be carried out to fully characterize the system as a whole. The development of a final iteration of electronics that eliminate the need for pressure tubing will greatly reduce the response time of the sensor. The response time for both velocity and directional measurements should also be validated empirically.

It would also help to perform more in depth comparisons of other wind sensors. The most

comparable sensors for the CMHP's application are ultrasonic anemometers, but it would be fruitful to compare to different types of systems, such as cup anemometers and hot wires. Finally, a study on the effectiveness of the purge system and it's impact on pressure readings will need to be performed in the future.

7.2.2 Final Thoughts

Overall the 360° degree differential sensing concept shows great promise in providing useful tools to weather researchers. Once refined to a push button solution, this design could have a great impact in many different wind sensing applications. The CMHP is an improvement on other similar systems that have been created in the past. It is more compact and usable than systems like the extreme turbulence probe, but more weather hardened than systems like the MEMS based system. The Cylindrical Multi-hole Probe prototype has shown that with further development it could become a viable low-cost option for taking scientific grade wind measurements in hazardous weather environments.

REFERENCES

[1] Skew-T Parameters and indices. https://www.weather.gov/source/zhu/ZHU_Training_Page/convective_parameters/skewt/skewtinfo.html.

[2] PCB trace width calculator. https://www.7pcb.com/trace-width-calculator.php, 2017.

[3] How to choose the suitable thickness of SMT stencils? https://www.smartsmttools.com/test1/, Nov 2019.

[4] OSU drone expertise is supporting the exploration of earth and the final frontier. https://www.okcommerce.gov, Jul 2021.

[5] Greg Carbin. Toto (totable tornado observatory, 2019.

[6] G Comte-Bellot. Hot-wire anemometry. *Annual Review of Fluid Mechanics*, 8(1):209–231, 1976.

[7] Monroe Conner. Air data sensing system: Capturing critical data for flight control. https://www.nasa.gov/centers/dryden/multimedia/imagegallery/F-18SRA/EC97-43936-9.html, Jun 2015.

[8] Geoffrey W. Donnell, Jordan A. Feight, Nate Lannan, and Jamey D. Jacob. Wind characterization using onboard imu of suas. *2018 Atmospheric Flight Mechanics Conference*, 2018.

[9] Richard Eckman, Ronald Dobosy, David Auble, Thomas Strong, and Timothy Crawford. A pressure-sphere anemometer for measuring turbulence and fluxes in hurricanes. *Journal of Atmospheric and Oceanic Technology - J ATMOS OCEAN TECHNOL*, 24, 06 2007.

[10] Jamey Jacob, Phillip Chilson, Adam Houston, and Suzanne Smith. Considerations for atmospheric measurements with small unmanned aircraft systems. *Atmosphere*, 9(7):252, 2018.

[11] T. G. Konrad, M. L. Hill, and J. R. Rowland. A small, radio-controlled aircraft as a platform. `https://www.jhuapl.edu/Content/techdigest/pdf/APL-V10-N02/APL-10-02-Konrad.pdf`, 1970.

[12] L. Kristensen. *The cup anemometer and other exciting instruments*. PhD thesis, Oceanic Technol, 1993.

[13] Leif Kahl Kristensen. Cups, props and vanes. 1994.

[14] D H Lenschow. Bulletin No. 23 MEASUREMENT TECHNIQUES: AIR MOTION SENSING, NCAR, 1989.

[15] Cheng Liu, Lidong Du, Zhan Zhao, Zhen Fang, Cheng Liu, and Liang Li. A directional anemometer based on mems differential pressure sensors. *The 9th IEEE International Conference on Nano/Micro Engineered and Molecular Systems (NEMS)*, Sep 2014.

[16] Renee A. McPherson, Christopher A. Fiebrich, Kenneth C. Crawford, James R. Kilby, David L. Grimsley, Janet E. Martinez, Jeffrey B. Basara, Bradley G. Illston, Dale A. Morris, Kevin A. Kloesel, and et al. Statewide monitoring of the mesoscale environment: A technical update on the oklahoma mesonet. *Journal of Atmospheric and Oceanic Technology*, 24(3):301–321, 2007.

[17] Shigeru Ogawa and Yusuke Kimura. Performance improvement by control of wingtip vortices for vertical axis type wind turbine. *Open Journal of Fluid Dynamics*, 08:331–342, 01 2018.

[18] A Alberigi Quaranta, G C Aprilesi, G De Cicco, and A Taroni. A microprocessor based, three axes, ultrasonic anemometer. *Journal of Physics E: Scientific Instruments*, 18(5):384–387, 1985.

[19] Mustafa Sarioglu and Tahir Yavuz. Subcritical flow around bluff bodies. *AIAA Journal*, 40(7):1257–1268, Jul 2002.

[20] Adam B Smith. Billion-dollar weather and climate disasters, Mar 2020.

[21] John Smith. Mobile Mesonet. `https://www.nssl.noaa.gov/projects/torus/`, 2012.

[22] Ankur Srivastava, Andrew Meade, and Kurtis Long. Learning air data parameters for flush air data sensing systems. *Journal of Aerospace Computing, Information, and Communication*, 9, 11 2012.

[23] Irene Suomi and Timo Vihma. Wind gust measurement techniques—from traditional anemometry to new possibilities. *Sensors*, 18(4):1300, 2018.

[24] Jean-Noël Thépaut. New technologies and applications of satellite data ... - earth.esa.int, 2012.

[25] A. L. Treaster and A. M. Yocum. The calibration and application of five-hole probes. 1978.

APPENDICES

Design Drawings and Schematics

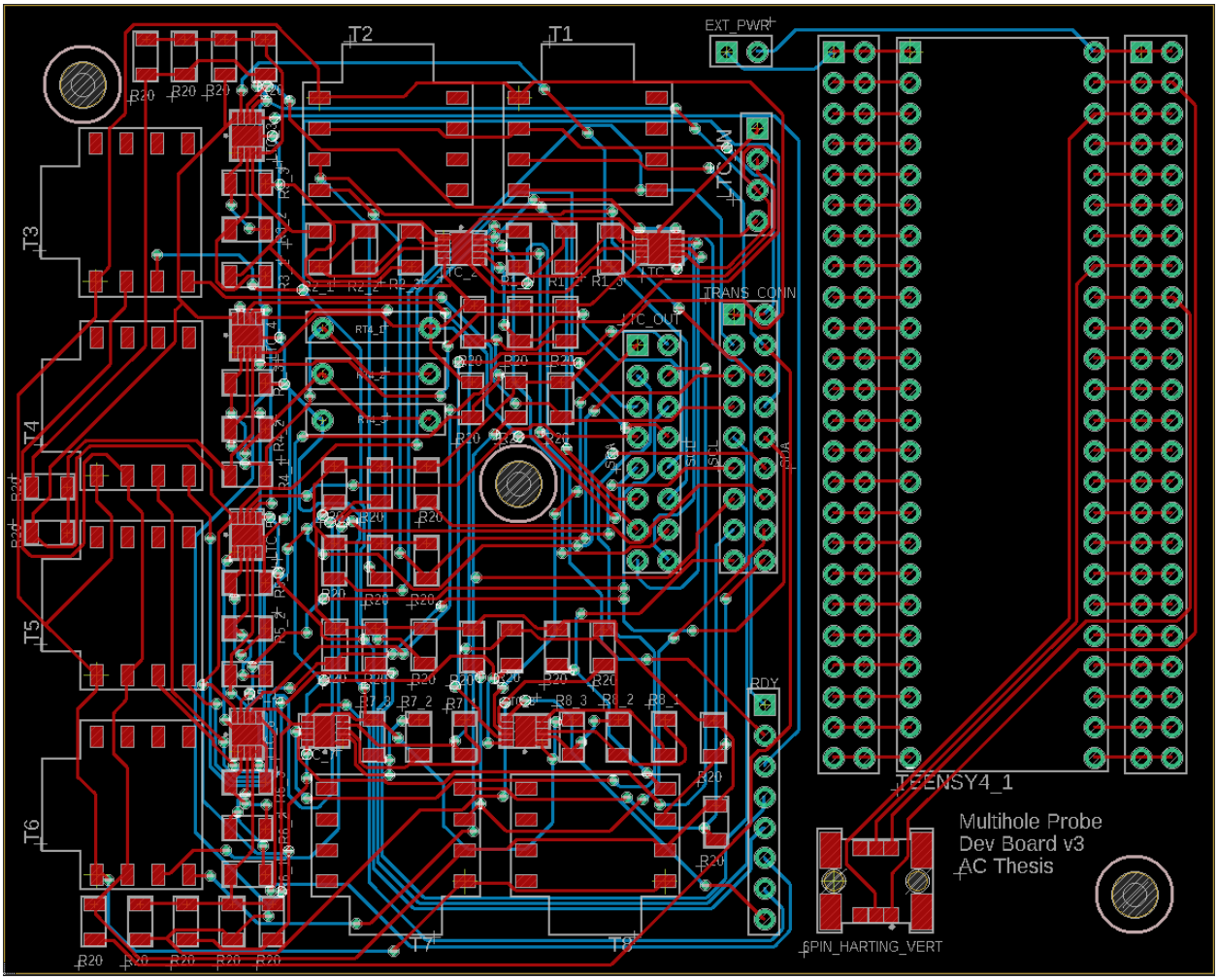

Figure 60: PCB layout.

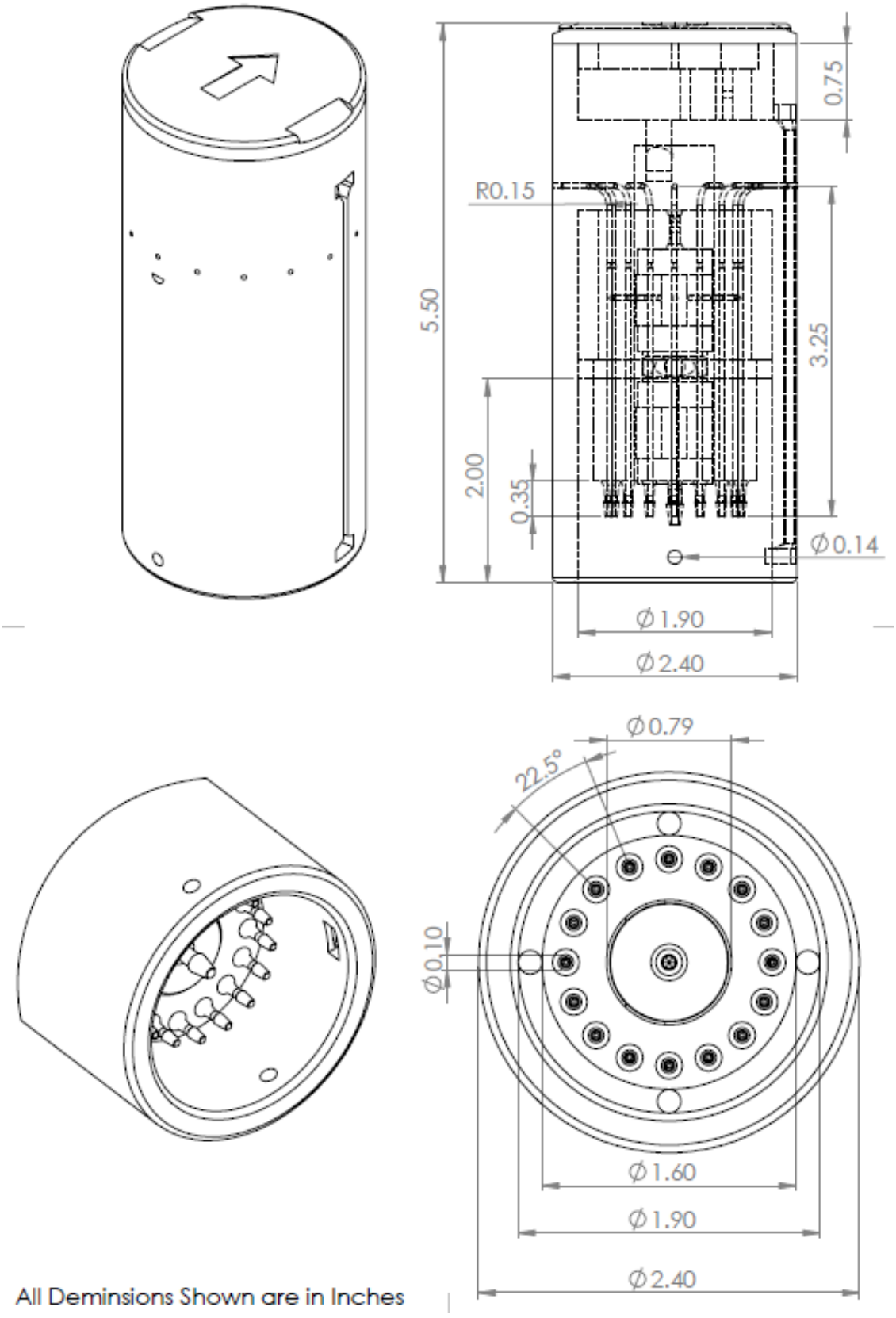

Figure 61: Anemometer CAD.

Data Logger Code

```
/* CMHP Sensor Suite Package
   Oklahoma State University

   Created by: Andrew Cole        | andrew.cole11@okstate.edu
   Based on Code by: Levi Ross    | levi.ross@okstate.edu

   Last edit:  06/26/2022

   Code for Teensy 4.1 that logs data from 8 differential pressure transducers
      that function as a Multi-Hole Probe over I2C connections. Allows for
      Pixhawk timing data to be recorded for post-processing and data fusion.

   For access to the development environment needed to run this code, email Levi Ross
      and he will send a portable Arduino installation (currently 1.8.15) with all
      libraries and configurations set for proper functionality.

   All libraries needed to operate code and process data can be found on the GitHub:
   https://github.com/LeviRoss2/USRI_Sensor_Suite

*/
```

```
21
22 /////////// SET VALUES HERE ///////////////////////
23 #define Pixhawk true      // true for MavLink message parsing
24 #define USB true         // true for full serial output, false for no
     serial output (Serial.print(), etc.)
25 #define MHP true          // true for Temp and Humidity calcs
26 #define encoder true          // true for rotary encoder
27
28 int MHP_freq = 200;       // Hz | MAX: 200, MIN: 2 | Five Hole Probe
     refresh rate
29
30 // Feel free to change description to match which angles being calculated
31 // Where the calcuation angles are based on specific trandsducer numbers
32 #define addressMHP1 0x78  // I2C address of first 5HP differential
     transducer
33 #define addressMHP2 0x29  // I2C address of second differential
     transducers
34 #define addressMHP3 0x38  // I2C address of third differential transducers
35 #define addressMHP4 0x38  // I2C address of fourth differential
     transducers
36 #define addressMHP5 0x38  // I2C address of fifth differential transducers
37 #define addressMHP6 0x38  // I2C address of sixth differential transducers
38 #define addressMHP7 0x38  // I2C address of seventh differential
     transducers
39 #define addressMHP8 0x38   // I2C address of eighth differential
     transducers
40 #define addressencoder 0x25 // I2C address of Rotary Encoder (for
```

```
    windtunnel Testing)
41 ///////// DO NOT EDIT BEYOND THIS POINT ///////////
42
43 ////////////////////// LIBRARY CONFIGURATION
44 #include "SD.h"       // Access to SD card data (read/write, USB file
    transfer)
45 #include <mavlink.h>  // Allows for connection with Pixhawk-type
    autopilots
46 #include <TimeLib.h>  // Required for datetime conversions
47 #include <Wire.h>     // Updated for Teensy 3.X and 4.X support
48 #include <SPI.h>      // Required for SD card functions
49 ////////////////////// END LIBRARY CONFIGURATION
50
51 ////////////////////// SD CARD CONFIG
52 const uint8_t chipSelect = BUILTIN_SDCARD;  // Choose the Teensy built-in
    SD card slot
53 char filename[12];  // initialize the filename used to write to SD files
54 File file;          // Initialize the variables used to access SD files
55 ////////////////////// END SD CARD CONFIG
56
57 ////////////////////// SHARED VARIABLES
58 int Time = millis();  // Initialize internal timer
59 #define led LED_BUILTIN   // Teensy built-in LED
60 ////////////////////// END SHARED VARIABLES
61
62 ////////////////////// PIXHAWK SETUP
63 float oldPixTime = 0;         // Previous Pixhawk GPS time, only run again
```

```
      if new data exists
uint32_t GPS_stat[1] = {0};   // GPS state (3D lock, 2D, none, etc.)
double PixTime[2] = {0, 0}; // Pixhawk system time (us) and time since
    boot (ms)
elapsedMillis oldTime;        // Timer for LED blink
#define pixSerial Serial2
//////////////////// END Pixhawk Setup

//////////////////// MHP SETUP
int MHPiter = 0;   // Interval frequency for 5HP measurement logging
int MHPlast = 0;   // Last time interval logged
int last_tot = 0;  // Last time the 1 second check was run (to verify loop
     speed)
int MHPcount = 0;  // Total times the 5HP data was run in the last second
int reading = 0;   // Send bit data to SD card and display in Serial
    Monitor
//////////////////// END MHP SETUP

//////////////////// DATE/TIME SETUP
uint16_t Year = 0000;
uint8_t Month = 00;
uint8_t Day = 00;
uint8_t Hour = 00;
uint8_t Minute = 00;
uint8_t Second = 00;
//////////////////// END DATE/TIME SETUP

```

```
87
88
89
90 #define TCAADDR 0x70
91
92 void tcaselect(uint8_t i) {
93   if (i > 7) return;
94
95   Wire.beginTransmission(TCAADDR);
96   Wire.write(1 << i);
97   Wire.endTransmission();
98 }
99
100
101 ////////////////////// Setup Loop
102 void setup() {
103   Serial.begin(115200);
104   pinMode(led, OUTPUT);      // Teensy build-in LED for reference
105
106 #if Pixhawk == true
107   pixSerial.begin(115200);
108   MavLink_receive();
109   Serial.println("Pixhawk data requested.");
110 #endif
111
112   // Allow all systems to catch up
113   delay(1000);
```

```
114
115   Wire.begin(); //i2c bus 1
116
117   if (MHP_freq < 2) {
118     MHP_freq = 2;
119   }
120   if (MHP_freq > 200) {
121     MHP_freq = 200;
122   }
123   MHPiter = 1000 / MHP_freq;
124
125   // Initialize SD card
126   if (!SD.begin(chipSelect)) {
127     while (1) {
128       digitalWrite(led, !digitalRead(led));
129       Serial.println("Teensy 4.1 SD fail. Reset card and try again.");
130       delay(2000);
131     }
132   }
133
134   Serial.println("SD card initialized.");
135
136   // Initialize filename variable
137   // Iterate over all known files to keep naming consistent but iterable
138   int n = 0;
139 #if Pixhawk == true
140   MavLink_receive();
```

```
141
142 snprintf(filename, sizeof(filename), "LIVE%03d.csv", n); // includes a
     three-digit sequence number in the file name
143 while (SD.exists(filename)) {
144   n++;
145   snprintf(filename, sizeof(filename), "LIVE%03d.csv", n);
146 }
147
148 // We can make it pause until GPS is acquired
149 // For now, turn off so test = real
150 if (USB == false) {
151   Year = year(PixTime[0]);
152   Month = month(PixTime[0]);
153   Day = day(PixTime[0]);
154   Hour = hour(PixTime[0]);
155   Minute = minute(PixTime[0]);
156   Second = second(PixTime[0]);
157 }
158 // Chose generic date, can be anything. Seeing this means no GPS
159 else {
160   Year = 2001;
161   Month = 01;
162   Day = 01;
163   Hour = 01;
164   Minute = 01;
165   Second = 01;
166 }
```

```
167
168 #else
169   snprintf(filename, sizeof(filename), "TEST%03d.csv", n); // includes a
      three-digit sequence number in the file name
170   while (SD.exists(filename)) {
171     n++;
172     snprintf(filename, sizeof(filename), "TEST%03d.csv", n);
173   }
174   // Chose generic date, can be anything. Seeing this means no GPS
175   Year = 2001;
176   Month = 01;
177   Day = 01;
178   Hour = 01;
179   Minute = 01;
180   Second = 01;
181
182 #endif
183
184   file = SD.open(filename, FILE_WRITE);
185   file.println("Board Time(ms),DS1-B1,DS1-B2,DS1-B3,DS1-B4,DS2-B1,DS2-B2,
      DS2-B3,DS2-B4,DS3-B1,DS3-B2,DS3-B3,DS3-B4,DS4-B1,DS4-B2,DS4-B3,DS4-B4,
      DS5-B1,DS5-B2,DS5-B3,DS5-B4,DS6-B1,DS6-B2,DS6-B3,DS6-B4,DS7-B1,DS7-B2,
      DS7-B3,DS7-B4,DS8-B1,DS8-B2,DS8-B3,DS8-B4, Unix Time (sec), Pix Boot
      Time (ms)");
186   file.close();
187
188
```

```
189   Serial.println(n);
190   Serial.println(filename);
191   Serial.println("File setup complete.");
192
193   //#if encoder == ture
194
195   //Setting up interrupt
196   //A rising pulse from encodenren activated ai0(). AttachInterrupt 0 is
       DigitalPin nr 2 on moust Arduino.
197   attachInterrupt(2, ai0, RISING);
198
199   //B rising pulse from encodenren activated ai1(). AttachInterrupt 1 is
       DigitalPin nr 3 on moust Arduino.
200   attachInterrupt(3, ai1, RISING);
201 }
202
203 //#endif
204
205 }
206
207 ////////////////////// Main Loop
208 void loop() {
209
210   Serial.println("Entering main data collection loop now...");
211   while (1) {
212
213     digitalWrite(led, !digitalRead(led));
```

```
214     file = SD.open(filename, FILE_WRITE);
215     Time = millis();
216     file.print((String) Time + ',');
217
218 #if USB == true
219     Serial.println(String("Time: ") + Time);
220 #endif
221
222
223 #if MHP == true
224     if ((Time - MHPlast) >= MHPiter) {
225       MHPlast = Time;
226       MHPcount++;
227       mhpCheck();
228     } else {
229       file.print((String)' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + '
       ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' +
      ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' +
       ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ','
      + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ','
       + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',');
230     }
231 #else
232     file.print((String)' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' '
       + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + '
      ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + '
       ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' +
```

```
      ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',' +
       ' ' + ',' + ' ' + ',' + ' ' + ',' + ' ' + ',');
#endif

#if Pixhawk == true
    MavLink_receive();

    if (PixTime[1] > oldPixTime) {
      file.print(PixTime[0]);
      file.print(',');
      file.print(PixTime[1]);
      file.print(',');
      oldPixTime = PixTime[1];
    }
    else {
      file.print((String)' ' + ',' + ' ' + ',');
    }
#else
    file.print((String)' ' + ',' + ' ' + ',');
#endif

    file.println();
    file.close();

#if USB == true
    if ((Time - last_tot) >= 1000) {
      Serial.println(); Serial.print("MHP count: "); Serial.println(
```

```
      MHPcount);
258       Serial.println();
259       last_tot = Time;
260       MHPcount = 0;
261     }
262 #endif
263   }
264 }
265
266 ////////////////////// Reads data from Pressure Transducers
267 void mhpCheck() {
268
269 #if USB == true
270   Serial.println();
271 #endif
272
273   // Hard code the read for each sensor due to time-sensitivities
274   // Down the road, convert to single for-loop
275   // Doesnt speed up code, but makes it more concise to read
276   // Pre-define "reading" as -1 to track bad reads
277   // Code can "get stuck" on a given read if it doesnt exist (device not
      connected)
278   // Feel free to comment out loops that are hanging for now, we can
      resolve later
279
280   int i = 0;
281
```

```
282   while (i < 8) {
283     tcaselect(i);
284
285     reading = -1;
286
287 #if USB == true
288     Serial.print( String("Tr") + (i + 1) + String(':'));
289 #endif
290
291     Wire.requestFrom(addressMHP1, 4);   // request 4 bytes from slave
      device
292     while (Wire.available()) { // slave may send less than requested
293       reading = Wire.read();   // receive a byte as int
294       file.print(reading + String(','));
295 #if USB == true
296       Serial.print(reading + String(" "));
297
298 #endif
299     }
300
301     i++;
302   }
303
304   //#if encoder == true
305
306   // Send the value of counter
307   if ( counter != temp ) {
```

```
308     pos = (float(temp) / 2.0) * (360.0 / 600.0);
309   };
310
311   if (counter < 0) {
312     counter = 1200 + counter;
313   }
314   if (counter >= 1200) {
315     counter = counter - 1200;
316   }
317
318   Serial.print (pos);
319   temp = counter;
320
321   //#endif
322 }
323
324 void MavLink_receive() {
325   mavlink_message_t msg;
326   mavlink_status_t status;
327
328   while (pixSerial.available())
329   {
330     uint8_t c = pixSerial.read();
331
332     //Get new message
333     if (mavlink_parse_char(MAVLINK_COMM_0, c, &msg, &status))
334     {
```

```
335
336         //Handle new message from autopilot
337         switch (msg.msgid)
338         {
339
340           case MAVLINK_MSG_ID_SYSTEM_TIME:  // #27: RAW_IMU
341             {
342               /* Message decoding: PRIMITIVE
343                     static inline void mavlink_msg_raw_imu_decode(const
      mavlink_message_t* msg, mavlink_raw_imu_t* raw_imu)
344               */
345               mavlink_system_time_t system_time;
346               mavlink_msg_system_time_decode(&msg, &system_time);
347
348               uint64_t Ptime = system_time.time_unix_usec;
349               PixTime[0] = Ptime / 1000000;
350               PixTime[1] = system_time.time_boot_ms;
351
352 #if USB == true
353               Serial.print("Unix Time: ");
354               Serial.print(PixTime[0]);
355               Serial.print("Pix Boot Time: ");
356               Serial.println(PixTime[1]);
357 #endif
358
359             }
360             break;
```

```
361
362         case MAVLINK_MSG_ID_HEARTBEAT:  // #0: Heartbeat
363           {
364
365             mavlink_heartbeat_t heartbeat;
366             mavlink_msg_heartbeat_decode(&msg, &heartbeat);
367
368             //armed = ((heartbeat.base_mode & MAV_MODE_FLAG_SAFETY_ARMED)
    ? true : false);
369
370           }
371           break;
372
373         case MAVLINK_MSG_ID_GPS_RAW_INT:  // #27: RAW_IMU
374           {
375             /* Message decoding: PRIMITIVE
376                    static inline void mavlink_msg_raw_imu_decode(const
    mavlink_message_t* msg, mavlink_raw_imu_t* raw_imu)
377             */
378             mavlink_gps_raw_int_t gps_raw_int;
379             mavlink_msg_gps_raw_int_decode(&msg, &gps_raw_int);
380
381             GPS_stat[0] = gps_raw_int.fix_type;
382
383           }
384           break;
385       }
```

```
386     }
387   }
388 }
389
390 void ai0() {
391   // ai0 is activated if DigitalPin nr 2 is going from LOW to HIGH
392   // Check pin 3 to determine the direction
393   if (digitalRead(3) == LOW) {
394     counter++;
395   } else {
396     counter--;
397   }
398 }
399
400 void ai1() {
401   // ai0 is activated if DigitalPin nr 3 is going from LOW to HIGH
402   // Check with pin 2 to determine the direction
403   if (digitalRead(2) == LOW) {
404     counter--;
405   } else {
406     counter++;
407   }
408 }
```

Post Processing Code

```
global parsedVars
global parseRangeUTC
global MHPNumber
global CpData
global directionmethod

% Read Cp vs. Theta data in as matrix
CpData = readmatrix('RE#_plot'); %this file must be in the same root
    folder
% Convert all NaN to -1
CpData(isnan(CpData)) = -1;


%//////////////Inputs//////////////////////
directionmethod = "BASIC"; %BASIC,LINEAR, or CPMATCH
%/////////////////////////////////////////
baseNameNoExt = "test1";
processTeensy(baseNameNoExt);

%% FUNC: processTeensy - Read Teensy data, determine which datasets exist
% Determine which of MultiHoleProbe or TPH suite exists
% INPUT
% * baseNameNoExt - Pix base file name with no extension or filepath
    details
function processTeensy(baseNameNoExt)
```

```
global MHPNumber
global TPHNumber
global TeensyNumber
global parsedTeensy

% Get the name of the file that the user wants to use.
[baseNameNoExtTeensy, ~, folderTeensy, fullInputMatFileNameTeensy]...
    = file2open('*.csv','Select a Teensy Suite .CSV file');
% Read data in as matri=x
data = readmatrix(fullInputMatFileNameTeensy);
% Convert all NaN to -1
data(isnan(data)) = -1;

fullFile = textscan(fopen(fullInputMatFileNameTeensy),'%s');
rawData = fullFile{1};
iter = 0;
for i=15:4:length(rawData)
    iter = iter + 1;
    row = rawData{i};
    rowData = split(row,',')';
    posarray(iter,1) = rowData(1,end);
end
posarray = string(posarray);

% Check if MHP data exists
if(max(data(:,4))>0)
    % If data exists, iterate MHP number (in case more than one is used)
```

```
51      MHPNumber = MHPNumber+1;
52      outputMHP = processMHP(data, folderTeensy, baseNameNoExt,posarray);
53  else
54      % If no data exists, output as empt
55      outputMHP = [];
56  end
57
58  end
59
60  %% FUNC: processMHP - Parse and process MHP data
61  % Parse MHP data based on parseRangeUTC data from Pixhawk
62  % Output external table (DateUTC, TimeUTC) to user workspace
63  % Preseve internal table (DatenumUTC) for other uses
64  % INPUT
65  % * data - Teensy full dataset
66  % * folder - folder containing raw Teensy data
67  % * baseNameNoExt - Pix base file name with no extension or filepath
        details
68  % OUTPUT
69  % * outputMHP - Internal version of resulting table (DatenumUTC)
70  function outputMHP = processMHP(data, folder, baseNameNoExt,posarray)
71
72  global parsedVars
73  global parseRangeUTC
74  global MHPNumber
75
76  % Get number of Rows and Columns of the data
```

```
77 nrows = length(data(:,1));
78 ncols = length(data(1,:));
79
80 % Air desnity, pre-set for now but will be calc'ed from TPH data
81 rho=1.197; % kg/m3
82
83 % Find location of IMU, CTUN, GPS, and NKF1 tables (all required)
84 % for j=1:length(parsedVars)
85 %     if(strcmpi('IMU_table',char(parsedVars{j,1})))
86 %         locIMU=j;
87 %     end
88 %     if(strcmpi('CTUN_table',char(parsedVars{j,1})))
89 %         locCTUN=j;
90 %     end
91 %     if(strcmpi('GPS_table',char(parsedVars{j,1})))
92 %         locGPS=j;
93 %     end
94 %     if(strcmpi('NKF1_table',char(parsedVars{j,1})))
95 %         locNKF1=j;
96 %     end
97 %     if(strcmpi('XKF1_table',char(parsedVars{j,1})))
98 %         locNKF1=j;
99 %     end
100 % end
101
102 % Convert the parsedVars tables to arrays
103 % IMU = table2array(parsedVars{locIMU,2});
```

```
% CTUN = table2array(parsedVars{locCTUN,2});
% GPS = table2array(parsedVars{locGPS,2});
% NKF1 = table2array(parsedVars{locNKF1,2});

% Convert rawData into arrays of relevant data
time = data(:,1);
TR1B1 = data(:,2);
TR1B2 = data(:,3);
TR2B1 = data(:,6);
TR2B2 = data(:,7);
TR3B1 = data(:,10);
TR3B2 = data(:,11);
TR4B1 = data(:,14);
TR4B2 = data(:,15);
TR5B1 = data(:,18);
TR5B2 = data(:,19);
TR6B1 = data(:,22);
TR6B2 = data(:,23);
TR7B1 = data(:,26);
TR7B2 = data(:,27);
TR8B1 = data(:,30);
TR8B2 = data(:,31);
UnixT = data(:,34);
PixT = data(:,35);
pos = data (:,37);

% Backfill Pixhawk BOOT and UNIX times via linear interpolation
```

```
% filledPixT(:,1)=timeInterpolation(time(:,1),PixT(:,1));
% filledUnixT(:,1)=timeInterpolation(time(:,1),UnixT(:,1));

% Convert MHP data to Alpha, Beta, and Pitot base values
TR1Count = ((TR1B1*256)+TR1B2);
TR2Count = ((TR2B1*256)+TR2B2);
TR3Count = ((TR3B1*256)+TR3B2);
TR4Count = ((TR4B1*256)+TR4B2);
TR5Count = ((TR5B1*256)+TR5B2);
TR6Count = ((TR6B1*256)+TR6B2);
TR7Count = ((TR7B1*256)+TR7B2);
TR8Count = ((TR8B1*256)+TR8B2);

TR1Cal = 1029;
TR2Cal = 1007;
TR3Cal = 1022;
TR4Cal = 1025;
TR5Cal = 1014;
TR6Cal = 1038;
TR7Cal = 1022;
TR8Cal = 1007;

% Convert base values to pressure values (inH2O)
TR1_inH2O=((TR1Count-1638)*(5+5))/(14745-1638)-1;
TR2_inH2O=((TR2Count-1638)*(5+5))/(14745-1638)-1;
TR3_inH2O=((TR3Count-1638)*(5+5))/(14745-1638)-1;
TR4_inH2O=((TR4Count-1638)*(5+5))/(14745-1638)-1;
```

```
158 TR5_inH2O=((TR5Count-1638)*(5+5))/(14745-1638)-1;
159 TR6_inH2O=((TR6Count-1638)*(5+5))/(14745-1638)-1;
160 TR7_inH2O=((TR7Count-1638)*(5+5))/(14745-1638)-1;
161 TR8_inH2O=((TR8Count-1638)*(5+5))/(14745-1638)-1;
162
163 % Convert pressure values to Pascals - 1 inH2O = 0.0360912 psi = 248.84 pa
164 TR1_pa=TR1_inH2O*248.84;
165 TR2_pa=TR2_inH2O*248.84;
166 TR3_pa=TR3_inH2O*248.84;
167 TR4_pa=TR4_inH2O*248.84;
168 TR5_pa=TR5_inH2O*248.84;
169 TR6_pa=TR6_inH2O*248.84;
170 TR7_pa=TR7_inH2O*248.84;
171 TR8_pa=TR8_inH2O*248.84;
172
173 % Do math here ------------------------------------------------
174
175
176 %Fix Position Data
177 %Remove Fragmented Data
178 for i = 2:length(posarray)
179 if strlength(posarray(i,1)) ~= 3
180 posarray(i,1) = NaN;
181 end
182 end
183
184 posarray = str2double(posarray);
```

```
185 last = 0;
186 for i = 2:length(posarray)
187 if posarray(i,1)>360||posarray(i,1)<0
188 posarray(i,1) = NaN;
189 elseif posarray(i,1)==360
190 posarray(i,1) = 0;
191 end
192 if isnan(posarray(i,1))
193 else
194 last = posarray(i,1);
195 end
196 posarray(i,1) = 360 - posarray(i,1);
197 end
198
199 %Unwrap direction
200 for i = 1:length(posarray)
201 if posarray(i,1)>180
202 posarray(i,1) = posarray(i,1)-360;
203 end
204 end
205
206 % windowSize = 1;
207 % b = (1/windowSize)*ones(1,windowSize);
208 % a = 1;
209 % posarray = filter(b,a,posarray);
210
211 % Add all relevant data to MHPData matrix
```

```
MHPData(:,1)=time;          % Sensor board time
% MHPData(:,2)=filledPixT;  % Pix board time
% MHPData(:,3)=filledUnixT; % Pix Unix Time (GPS)
MHPData(:,4)=smoothdata(TR1_pa-TR1Cal,'movmean',20);
MHPData(:,5)=smoothdata(TR2_pa-TR2Cal,'movmean',20);
MHPData(:,6)=smoothdata(TR3_pa-TR3Cal,'movmean',20);
MHPData(:,7)=smoothdata(TR4_pa-TR4Cal,'movmean',20);
MHPData(:,8)=smoothdata(TR5_pa-TR5Cal,'movmean',20);
MHPData(:,9)=smoothdata(TR6_pa-TR6Cal,'movmean',20);
MHPData(:,10)=smoothdata(TR7_pa-TR7Cal,'movmean',20);
MHPData(:,11)=smoothdata(TR8_pa-TR8Cal,'movmean',20);
MHPData(:,12)=posarray;

%Calculate Windspeed and Direction
[vel, dir] = CMHPCalc(MHPData,rho);

%Unwrap direction
for i = 1:length(dir)
if dir(i,1)>180
dir(i,1) = dir(i,1) -360;
end
end

MHPData(:,13) = smoothdata(vel);
MHPData(:,14) = dir;


```

```
239
240
241 %Average every two datapoints
242 MHPDataTemp = zeros(size(MHPData));
243
244 for c = 4:11
245 MHPDataTemp(1,c)= MHPData(1,c);
246 for i = 2:length(MHPData(:,c))-1
247 MHPDataTemp(i,c) = (MHPData(i+1,c)+MHPData(i,c)+MHPData(i-1,c))/3;
248 end
249 MHPDataTemp(i+1,c)= MHPData(i+1,c);
250 end
251
252 for c = 4:11
253 MHPData(:,c) = MHPDataTemp(:,c);
254 end
255
256
257 % Cut out trailing data that wasnt interpolated
258 %MHPData(isnan(MHPData(:,3)),:)=[];
259
260 % Generate DateUTC and TimeUTC datasets for user view
261 % MHP_DateTime=datetime(MHPData(:,3),...
262 %     'ConvertFrom','posixTime','Format','MMM-dd-yyyy HH:mm:ss.S');
263 % MHP_Date=datestr(MHP_DateTime,'mmm-dd-yyyy');
264 % MHP_Time=datestr(MHP_DateTime,'HH:MM:SS.FFF');
265 %
```

```
% % Generate DatenumUTC for internal use
% datenumMHP = datenum(MHP_DateTime);

% Find MHP start time from Pixhawk parse

TO_MHP = 1;


% Find MHP end time from Pixhawk parse

LND_MHP = length(MHPData);


% Parse all MHP data based on Pixhawk parsing
MHP_entry = MHPData(TO_MHP:LND_MHP,:);
% MHP_Date = MHP_Date(TO_MHP:LND_MHP,:);
% MHP_Time = MHP_Time(TO_MHP:LND_MHP,:);
MHP_time_out = (MHP_entry(:,1)-min(MHP_entry(:,1)))/1000;
% datenumMHP = datenumMHP(TO_MHP:LND_MHP,:);

% Create external table for user view
MHP_tableExternal = table(MHP_entry(:,1),MHP_entry(:,2),MHP_time_out,...
    MHP_entry(:,4),MHP_entry(:,5),MHP_entry(:,6),MHP_entry(:,7),MHP_entry
    (:,8),MHP_entry(:,9),MHP_entry(:,10),MHP_entry(:,11));...
    'VariableNames'; {'Board Time from PowerUp (msec)',...
    'Pix Time from PowerUp (msec)','Pix time from parse',...
    'TR1 (Pa)','TR2 (Pa)','TR3 (Pa)','TR4 (Pa)','TR5 (Pa)','TR6 (Pa)', '
```

```
    TR7 (Pa)','TR8 (Pa)', 'Position (deg)'};
MHP_tableExternal.Properties.Description = sprintf('MHP%d',MHPNumber);
% Output table to user workspace ----------------------------asignin
    --------
assignin('base',sprintf('MHP_table%d',MHPNumber),MHP_tableExternal);
% Save tabele for user review
table2saveCSV(baseNameNoExt, folder, MHP_tableExternal)

outputMHP = MHP_entry;

% Create new figure, iterate number to prevent overwriting
clf;
figIter = 0;
figIter = figIter+1;
Figs{figIter}=figure(figIter);
Figs{figIter}.Name = sprintf('CMHP Pressures.',MHPNumber);
% Prevent autoUpdate of axis bounds
set(Figs{figIter},'defaultLegendAutoUpdate','off');

%Plot recovered velocity values
plt = plot(MHP_entry(:,1), MHP_entry(:,4),'.',MHP_entry(:,1),MHP_entry
    (:,5),'.',MHP_entry(:,1),MHP_entry(:,6),'.',MHP_entry(:,1),MHP_entry
    (:,7),'.',MHP_entry(:,1),MHP_entry(:,8),'.',MHP_entry(:,1),MHP_entry
    (:,9),'.',MHP_entry(:,1),MHP_entry(:,10),'.',MHP_entry(:,1),MHP_entry
    (:,11),'.');
title('Differential Transducer Pressures')
xlabel('Time (ms)');
```

```
313 ylabel('Pressure (Pa)');
314 legend({'Tr1','Tr2','Tr3','Tr4','Tr5','Tr6','Tr7','Tr8'});
315
316 % yyaxis right
317 % plt = plot(MHP_entry(:,1), MHP_entry(:,12),'LineWidth',5);
318 % ylabel('Position (deg)');
319
320 % Create new figure, iterate number to prevent overwriting
321 figIter = figIter+1;
322
323 Figs{figIter}=figure(figIter);
324 Figs{figIter}.Name = sprintf("Basic Windspeed and Direction",MHPNumber);
325 % Prevent autoUpdate of axis bounds
326 set(Figs{figIter},'defaultLegendAutoUpdate','off');
327
328 %Plot recovered velocity values
329 plt = plot(MHP_entry(:,1), MHP_entry(:,13),'.');
330 title('Basic Windspeed')
331 xlabel('Time (ms)');
332 ylabel('Velocity (m/s)');
333 legend({'Measured Windspeed'});
334
335 figIter = figIter+1;
336 Figs{figIter}=figure(figIter);
337 Figs{figIter}.Name = sprintf('Probe Direction vs Position',MHPNumber);
338 % Prevent autoUpdate of axis bounds
339 set(Figs{figIter},'defaultLegendAutoUpdate','off');
```

```
340
341 hold on
342 plt = scatter(MHP_entry(:,1), MHP_entry(:,14),'.','b');
343 plt = scatter(MHP_entry(:,1), MHP_entry(:,12),'.','g');
344 title('Basic Method Direction');
345 xlabel('Time (ms)');
346 ylabel('Direction (deg)');
347 legend({'Direction','Probe Position'});
348 hold off
349 % figIter = figIter+1;
350 % Figs{figIter}=figure(figIter);
351 % Figs{figIter}.Name = sprintf('Probe Position',MHPNumber);
352 % % Prevent autoUpdate of axis bounds
353 % set(Figs{figIter},'defaultLegendAutoUpdate','off');
354 %
355 % %Plot recovered velocity values
356 % plt = plot(MHP_entry(:,1), MHP_entry(:,12));
357 % title('Probe Prosition')
358 % xlabel('Time (ms)');
359 % ylabel('Position (def)');
360 % legend({'Direction'});
361 end
362
363 %% FUNC: timeInterpolation - Uses board time to interpolate other time
364 % externTime is assumed to be the same length as board time
365 % Back fills empty data in externTime by:
366 % * Using boardTime as the constant interpolated against
```

```
% * Using partial externTime as the interpolant, or truth data
% INPUT
% * boardTime - highspeed board time
% * externTime - same length as boardTime, but different timing variable
% OUTPUT
% * interpolatedArray - externTime with empties filled

function interpolatedArray = timeInterpolation(boardTime,externTime)

% Initialize the counter for output array
Interpcount=0;

% Interpolation BuildUp
for i=1:length(boardTime(:,1))
    boardTimeInt = boardTime(i);   % Logger time (Teensy, Arduino, etc.)
    externTimeInt = externTime(i); % External time (Pix, Unix, GPS, etc.)

    % If valid timing value, add to interpolation array
    if(externTimeInt~=-1)
        Interpcount=Interpcount+1;
        InterpData(Interpcount,1)=boardTimeInt;   % Sensor board time
        InterpData(Interpcount,2)=externTimeInt;  % External Time
    end
end

% Remove Duplicate External Time interpolation points
NewVals=unique(InterpData(:,2));
```

```
394 % Concatenate data based on unique datapoints only
395 for i=1:length(NewVals(:,1))
396     TempVal = find(InterpData(:,2)==NewVals(i,1),1,'first');
397     conCat(i,1) = InterpData(TempVal,1);
398     conCat(i,2) = NewVals(i,1);
399 end
400
401 % Backfill gaps in full dataset
402 for j = 1:length(boardTime(:,1))
403     if(externTime(j,1) == -1)
404         externTime(j,1) = interp1(conCat(:,1),conCat(:,2),boardTime(j,1),'
      linear');
405     end
406 end
407
408 % Output the interpolated External Time array
409 interpolatedArray = externTime;
410
411 end
412
413 %% FUNC: file2open - Open File Based on File Input Type
414 % Prompt user to open a file of type "TYPE" with a heading of "TEXT"
415 % Saves file data as output
416 % INPUT
417 % * Type - File type to isolate for user to choose. Can be an array.
418 % * * Types: .csv, .txt, .xlsx, etc
419 % * * User will only be open the types provided
```

```
% * Text - Title of the UI figure that opens
% OUTPUT
% * baseNameNoExt - base file name with no extension or filepath details
% * baseName - base file name with extension but no filepath
% * folder - filepath to the file, but without any file information
% * fullInputMatName - file name with filepath and extension added
% * * These are used as passthrough to other functions
function [baseNameNoExt, baseName, folder, fullInputMatName] = file2open(
    type,text)

global startingFolder

% Get the name of the file that the user wants to use.
defaultName = fullfile(startingFolder,type);
% Grab baseName and folder directly from the loading process
[baseName, folder] = uigetfile(defaultName, text);

% Redefine starting folder as current folder
% Allows next file open to start in same folder
startingFolder = folder;

if baseName == 0
    % User clicked the Cancel button.
    return;
end

% Remove extension from the baseName
```

```
446 [~, baseNameNoExt, ~] = fileparts(baseName);
447
448 % Recombine all parts to recreate the full name
449 fullInputMatName = fullfile(folder, baseName);
450
451 end
452 %% FUNC: file2save - Save File Based on file2open Details
453 % Take file components from file2open and save in same location
454 % INPUT
455 % * baseNameNoExt - base file name with no extension or filepath details
456 % * folder - filepath to the file with no file information
457 % * varToSave - variable with data to save externally
458 function file2save(baseNameNoExt, folder, varToSave)
459
460 % Get the name of the intput.mat file and save as input_parsed.mat
461 baseFileName = sprintf('%s_Parsed.mat', baseNameNoExt);
462 % Generate output file name using folder details and new name
463 fullParsedMatFileName = fullfile(folder, baseFileName);
464 % Save file with parsed data as the original filename plus the added
      portion
465 save(fullParsedMatFileName,'varToSave');
466 end
467 %% FUNC: table2saveCSV - Save Specific Tables from Workspace To .CSV File
468 % Must be table (no arrays or structures)
469 % Before loading, define 'yourTable.Properties.Description = yourVarName;'
470 % * yourVarName will be appended after DFL file name as new .CSV file
471 % * * Ex: DFL Name: NimbusFlight2_5_27_2021.bin
```

```
% * *      varName : GPS
% * *      output  : NimbusFlight2_5_27_2021_varName.csv
% INPUT
% * baseNameNoExt - base file name with no extension or filepath details
% * folder - filepath to the file with no file information
% * tableToSave - table with data to save externally
function table2saveCSV(baseNameNoExt, folder, tableToSave)

% If Description of table is empty, use preset name. Else, use the name
if(isempty(tableToSave.Properties.Description))
    varName = 'undefinedVar';
else
    varName = tableToSave.Properties.Description;
end

% Save file as "PixhawkDFLname_VARNAME.csv"
% VARNAME can be undefinedVar if description is not set
baseFileName = sprintf('%s_%s.csv', baseNameNoExt, varName);
fullOutputMatFileName = fullfile(folder, baseFileName);
% Write data to .csv file
writetable(tableToSave, fullOutputMatFileName);

end
%% FUNC: CMHPCalc

function [vel, dir] = CMHPCalc(MHPData,rho)
global directionmethod
```

```
vel = zeros(size(MHPData(:,4)));
dir = zeros(size(MHPData(:,4)));
for i = 1:length(MHPData(:,4))
    pressure = [MHPData(i,4),MHPData(i,5),MHPData(i,6),MHPData(i,7),
    MHPData(i,8),MHPData(i,9),MHPData(i,10),MHPData(i,11)];
    posP = abs(pressure);
    spacing =
    [0;22.5;45;67.5;90;112.5;135;157.5;180;202.5;225;247.5;270;292.5;315;337.5];

    [Pmax,MaxTR] = max(posP);
    vel(i,1) = sqrt((2*Pmax)/rho);
    if Pmax<5&&i>1
        dir(i,1)=dir(i-1,1);
    elseif directionmethod == "BASIC"

        if MaxTR == 1
            if MHPData(i,3+MaxTR)>0
                dir(i,1) = 0;
            else
                dir(i,1) = 90;
            end
        elseif MaxTR == 2
            if MHPData(i,3+MaxTR)>0
                dir(i,1) = 22.5;
            else
                dir(i,1) = 112.5;
            end
```

```
523         elseif MaxTR == 3
524             if MHPData(i,3+MaxTR)>0
525                 dir(i,1) = 45;
526             else
527                 dir(i,1) = 135;
528             end
529         elseif MaxTR == 4
530             if MHPData(i,3+MaxTR)>0
531                 dir(i,1) = 67.5;
532             else
533                 dir(i,1) = 157.5;
534             end
535         elseif MaxTR == 5
536             if MHPData(i,3+MaxTR)>0
537                 dir(i,1) = 180;
538             else
539                 dir(i,1) = 270;
540             end
541         elseif MaxTR == 6
542             if MHPData(i,3+MaxTR)>0
543                 dir(i,1) = 202.5;
544             else
545                 dir(i,1) = 292.5;
546             end
547         elseif MaxTR == 7
548             if MHPData(i,3+MaxTR)>0
549                 dir(i,1) = 225;
```

```
            else
                dir(i,1) = 315;
            end
        elseif MaxTR == 8
            if MHPData(i,3+MaxTR)>0
                dir(i,1) = 247.5;
            else
                dir(i,1) = 337.5;
            end
        end

 elseif directionmethod == "LINEAR"
     pressure = abs([MHPData(i,4);MHPData(i,5);MHPData(i,6);MHPData(i
,7);MHPData(i,4);MHPData(i,5);MHPData(i,6);MHPData(i,7);MHPData(i,8);
MHPData(i,9);MHPData(i,10);MHPData(i,11);MHPData(i,8);MHPData(i,9);
MHPData(i,10);MHPData(i,11)]);
     spacing =
[0;22.5;45;67.5;90;112.5;135;157.5;180;202.5;225;247.5;270;292.5;315;337.5];


            if MaxTR==1
                diff1 = abs(pressure(MaxTR)-pressure(8));
                difftot=abs(pressure(MaxTR+1)-pressure(8));
                if diff1>difftot
                dir(i,1)=spacing(MaxTR);
                else
                dir(i,1)=(diff1/difftot)*45+spacing(8);
```

```
572                 end
573             else
574                 diff1 = abs(pressure(MaxTR)-pressure(MaxTR-1));
575                 difftot=abs(pressure(MaxTR+1)-pressure(MaxTR-1));
576                 if diff1>difftot
577                 dir(i,1)=spacing(MaxTR);
578                 else
579                 dir(i,1)=(diff1/difftot)*45+spacing(MaxTR-1);
580                 end
581             end
582     elseif directionmethod == "CPMATCH"
583             refcp = getCP(vel(i,1),0.0508); %for a 2in probe
584             refdiff = refcp(46,2)-refcp(length(refcp),2);
585
586             if MaxTR > 1 && MaxTR < 8
587
588                 cp1 = pressure(MaxTR-1)/Pmax;
589                 cp2 = pressure(MaxTR+1)/Pmax;
590                 diff1 = abs(cp1-refdiff);
591                 diff2 = abs(cp2-refdiff);
592
593             elseif MaxTR == 1
594
595                 cp1 = pressure(8)/Pmax;
596                 cp2 = pressure(MaxTR+1)/Pmax;
597                 diff1 = abs(cp1-refdiff);
598                 diff2 = abs(cp2-refdiff);
```

```
599             elseif MaxTR == 8
600
601                 cp1 = pressure(MaxTR-1)/Pmax;
602                 cp2 = pressure(1)/Pmax;
603                 diff1 = abs(cp1-refdiff);
604                 diff2 = abs(cp2-refdiff);
605             end
606
607                 if diff1>diff2
608                     [c,n]=min(abs(cp1-refcp(:,2)));
609                     diradj = refcp(n,1);
610                     if diradj > 22.5
611                         diradj = 22.5;
612                     end
613                     if pressure(MaxTR) > 0
614                         if MaxTR <=4
615                             dir(i,1) = spacing(MaxTR)-diradj;
616                         else
617                             dir(i,1) = spacing(MaxTR+4)-diradj;
618                         end
619                     else
620                         if MaxTR <=4
621                             dir(i,1) = spacing(MaxTR+4)-diradj;
622                         else
623                             dir(i,1) = spacing(MaxTR+8)-diradj;
624                         end
625                     end
```

```
626
627
628                 else
629                     [c,n]=min(abs(cp1-refcp(:,2)));
630                     diradj = refcp(n,1);
631                     if diradj > 22.5
632                         diradj = 22.5;
633                     end
634                     if pressure(MaxTR) > 0
635                         if MaxTR <=4
636                             dir(i,1) = spacing(MaxTR)+diradj;
637                         else
638                             dir(i,1) = spacing(MaxTR+4)+diradj;
639                         end
640                     else
641                         if MaxTR <=4
642                             dir(i,1) = spacing(MaxTR+4)+diradj;
643                         else
644                             dir(i,1) = spacing(MaxTR+8)+diradj;
645                         end
646                     end
647                 end
648
649             if dir(i,1) > 360
650                 dir(i,1) = dir(i,1)-360;
651             elseif dir(i,1) < 0
652                 dir(i,1) = dir(i,1) + 360;
```

```
653             end
654
655     end
656 end
657 end
658
659 %% FUNC: getCP
660
661 function [cp] = getCP(vel,L)
662 global CpData
663
664 kv = 0.00001488; %kenimatic viscoity in m^2/s
665
666 % % Read Cp vs. Theta data in as matrix
667 % CpData = readmatrix('RE#_plot'); %this file must be in the same root
      folder
668 % % Convert all NaN to -1
669 % CpData(isnan(CpData)) = -1;
670
671 Re = (vel*L)/kv;
672
673 [val,idx]=min(abs(CpData(1,2:end)-Re));
674
675 theta = CpData(3:end,1).';
676 refcps = CpData (3:end,1+idx).';
677 % inttheta = 0:0.5:112.5;
678 % intcps = interp1(theta,refcps,abs(inttheta));
```

```
679
680 inttheta = 0:0.5:112.5;
681 intcps = interp1(theta,refcps,abs(inttheta));
682
683 %theta = [0;25;30;40;50;60;70;75;80;85;90;95;100;110;120;140;150;160];
684 %refcps =
      [1;0.8;0.6;-0.375;-0.95;-1.6;-1.8;-1.74;-1.625;-1.5;-1.45;-1.42;-1.43;-1.46;-1.48;-
685
686 %ploting options
687 %plot (theta,refcps) %uncomment to see just selected Cp curve
688
689 % hold on %uncomment until hold off to plot all Cp data
690 % i=0;
691 % leg = strings(1,length(CpData(1,2:end)));
692 % for i = 2:length(CpData(1,:))
693 % cplot = CpData(3:end,i).';
694 % plot (theta,cplot);
695 % leg(1,i-1) = append("Re = ",string(CpData(1,i)));
696 % end
697 % legend (leg)
698 % hold off
699
700 cp = [inttheta;intcps].'; %put a breakpoint here to see Cp graphs.
701 end
```

```
global parsedVars
global parseRangeUTC
global MHPNumber
global TeensyNumber
global parsedTeensy
global startingFolder
global rawDFL
global rawVars
global arduPilotType
global redactStructDFL
global figIter
figIter = 0;
global Figs
global CMHPData
global YoungData
global MHP_tableExternal
global Young_tableExternal
global directionmethod

%//////////////Inputs//////////////////////
datatype = "BOTH";  %CMHP,YOUNG,or BOTH
mesonetdata = "FALSE"; %Mesonet Comparision (CMHP only)
mesoset = 2; %Which mesonet dataset to use
directionmethod = "BASIC"; %BASIC or LINEAR or CPMATCH
%////////////////////////////////////////////

%Mesonet Data
```

```
MesonetT1 = [datenum(2022,6,23,18,35,00);datenum(2022,6,23,18,40,00);
    datenum(2022,6,23,18,45,00);datenum(2022,6,23,18,50,00);datenum
    (2022,6,23,18,55,00);datenum(2022,6,23,19,00,00);datenum
    (2022,6,23,19,05,00);datenum(2022,6,23,19,10,00);datenum
    (2022,6,23,19,15,00)];
MesonetVel1 =[4.5;4.0;4.9;4.1;3.6;4.1;3.5;4.8;4.9];
MesonetDir1 =[177;199;182;182;174;177;165;186;163];
MesonetVMax1 = [7.1;5.8;6.8;6.4;5.5;5.7;5.5;6.4;6.4];

MesonetT2 = [datenum(2022,6,23,19,20,00);datenum(2022,6,23,19,25,00);
    datenum(2022,6,23,19,30,00);datenum(2022,6,23,19,35,00);datenum
    (2022,6,23,19,40,00);datenum(2022,6,23,19,45,00);datenum
    (2022,6,23,19,50,00);datenum(2022,6,23,19,55,00);datenum
    (2022,6,23,20,00,00);datenum(2022,6,23,20,05,00);datenum
    (2022,6,23,20,10,00)];
MesonetVel2 =[4.7;4.5;4.5;5.4;4.2;3.5;4.5;4.2;5.4;4.5;4.8];
MesonetDir2 =[163;145;144;167;173;141;145;139;133;144;159];
MesonetVMax2 = [6.2;6.6;6.7;7.3;5.8;6.3;7.1;7.2;7.3;6.3;6.7];



if mesoset == 1
    Meso=[MesonetT1,MesonetVel1,MesonetDir1,MesonetVMax1];
elseif mesoset == 2
    Meso=[MesonetT2,MesonetVel2,MesonetDir2,MesonetVMax2];
end

```

```
46 if datatype == "BOTH"
47
48
49
50     figIter = figIter+1;
51     baseNameNoExt = "test1";
52     processTeensy(baseNameNoExt);
53     baseNameNoExt = "test2";
54     processTeensy4young(baseNameNoExt)
55
56     %Unwrap direction
57     for i = 1:length(Young_tableExternal.Direction)
58         if Young_tableExternal.Direction(i,1)>180
59             Young_tableExternal.Direction(i,1) = Young_tableExternal.
    Direction(i,1) -360;
60         end
61     end
62
63     Figs{figIter}=figure(figIter);
64     Figs{figIter}.Name = sprintf("CMHP vs Young Ultrasonic",MHPNumber);
65     % Prevent autoUpdate of axis bounds
66     set(Figs{figIter},'defaultLegendAutoUpdate','off');
67
68     %Plot recovered velocity values
69     plt = plot(MHP_tableExternal.datenumMHP,MHP_tableExternal.Velocity,'.'
    , Young_tableExternal.datenum,Young_tableExternal.Velocity,'.');
70     datetick('x')
```

```
71      title('CMHP vs Young Ultrasonic Velocity Comparison')
72      xlabel('Unix Time');
73      ylabel('Velocity (m/s)');
74      legend({'Cylndrical Multihole Probe','Young 92000'});
75
76      figIter = figIter+1;
77      Figs{figIter}=figure(figIter);
78      Figs{figIter}.Name = sprintf('CMHP vs Young Ultrasonic Direction
        Comparison',MHPNumber);
79      % Prevent autoUpdate of axis bounds
80      set(Figs{figIter},'defaultLegendAutoUpdate','off');
81      clf
82      hold on
83      plt = scatter(MHP_tableExternal.datenumMHP,MHP_tableExternal.Direction
        ,'.');
84      plt = scatter(Young_tableExternal.datenum,Young_tableExternal.
        Direction,'.');
85      datetick('x')
86      ylabel('Direction (deg)');
87      xlabel('Unix Time');
88      legend({'Cylndrical Multihole Probe','Young 92000'});
89      hold off
90  elseif datatype == "CMHP"
91
92      if mesonetdata == "TRUE"
93
94          baseNameNoExt = "test1";
```

```
95          processTeensy(baseNameNoExt);
96          figIter = figIter+1;
97          Figs{figIter}=figure(figIter);
98          Figs{figIter}.Name = sprintf("CMHP ",MHPNumber);
99          % Prevent autoUpdate of axis bounds
100         set(Figs{figIter},'defaultLegendAutoUpdate','off');
101
102         clf
103         hold on
104         %Plot recovered velocity values
105         plt = plot(MHP_tableExternal.datenumMHP,MHP_tableExternal.Velocity
    );
106         plt = plot(Meso(:,1),Meso(:,2),Meso(:,1),Meso(:,4));
107         datetick('x')
108         title('CMHP Moving Median Avg Velocity vs Marena Mesonet')
109         xlabel('Unix Time');
110         ylabel('Velocity (m/s)');
111         legend({'Cylndrical Multihole Probe','Marena Mesonet 5 Min Avg','
    Marena Mesonet Max'});
112         hold off
113         figIter = figIter+1;
114         Figs{figIter}=figure(figIter);
115         Figs{figIter}.Name = sprintf('CMHP Direction',MHPNumber);
116         % Prevent autoUpdate of axis bounds
117         set(Figs{figIter},'defaultLegendAutoUpdate','off');
118         clf
119         hold on
```

```
120         plt = scatter(MHP_tableExternal.datenumMHP,MHP_tableExternal.
    Direction,".")
121         plt = plot(Meso(:,1),Meso(:,3));
122         datetick('x')
123         ylabel('Direction (deg)');
124         xlabel('Unix Time');
125         legend({'Cylndrical Multihole Probe','Marena Mesonet 5 Min Avg'});
126         hold off
127      else
128
129         baseNameNoExt = "test1";
130         processTeensy(baseNameNoExt);
131         figIter = figIter+1;
132         Figs{figIter}=figure(figIter);
133         Figs{figIter}.Name = sprintf("CMHP ",MHPNumber);
134         % Prevent autoUpdate of axis bounds
135         set(Figs{figIter},'defaultLegendAutoUpdate','off');
136
137         %Plot recovered velocity values
138         plt = plot(MHP_tableExternal.datenumMHP,MHP_tableExternal.Velocity
    );
139         datetick('x')
140         title('CMHP Velocity ')
141         xlabel('Unix Time');
142         ylabel('Velocity (m/s)');
143         legend({'Cylndrical Multihole Probe'});
144
```

```
145         figIter = figIter+1;
146         Figs{figIter}=figure(figIter);
147         Figs{figIter}.Name = sprintf('CMHP Direction',MHPNumber);
148         % Prevent autoUpdate of axis bounds
149         set(Figs{figIter},'defaultLegendAutoUpdate','off');
150 
151         plt = plot(MHP_tableExternal.datenumMHP,MHP_tableExternal.
    Direction);
152         datetick('x')
153         ylabel('Direction (deg)');
154         xlabel('Unix Time');
155         legend({'Cylndrical Multihole Probe'})
156 
157 
158     end
159 
160 elseif datatype == "YOUNG"
161     figIter = figIter+1;
162     baseNameNoExt = "test2";
163     processTeensy4young(baseNameNoExt)
164     Figs{figIter}=figure(figIter);
165     Figs{figIter}.Name = sprintf("Young Ultrasonic",MHPNumber);
166     % Prevent autoUpdate of axis bounds
167     set(Figs{figIter},'defaultLegendAutoUpdate','off');
168 
169     %Plot recovered velocity values
170     plt = plot(Young_tableExternal.datenum,Young_tableExternal.Velocity)
```

```
171     datetick('x')
172     title('Young Ultrasonic Velocity')
173     xlabel('Unix Time');
174     ylabel('Velocity (m/s)');
175     legend({'Young Anemometer'});
176 
177     figIter = figIter+1;
178     Figs{figIter}=figure(figIter);
179     Figs{figIter}.Name = sprintf('Young Ultrasonic Direction',MHPNumber);
180     % Prevent autoUpdate of axis bounds
181     set(Figs{figIter},'defaultLegendAutoUpdate','off');
182 
183     plt = plot(Young_tableExternal.datenum,Young_tableExternal.Direction);
184     datetick('x')
185     ylabel('Direction (deg)');
186     xlabel('Unix Time');
187     legend({'Young 8100'});
188 
189 end
190 
191 %% FUNC: processTeensy - Read Teensy data, determine which datasets exist
192 % Determine which of MultiHoleProbe or TPH suite exists
193 % INPUT
194 % * baseNameNoExt - Pix base file name with no extension or filepath
      details
195 function processTeensy(baseNameNoExt)
196 global MHPNumber
```

```
197 global TPHNumber
198 global TeensyNumber
199 global parsedTeensy
200 global CMHPData
201
202 % Get the name of the file that the user wants to use.
203 [baseNameNoExtTeensy, ~, folderTeensy, fullInputMatFileNameTeensy]...
204     = file2open('*.csv','Select a Teensy Suite .CSV file');
205 % Read data in as matri=x
206 data = readmatrix(fullInputMatFileNameTeensy);
207 % Convert all NaN to -1
208 data(isnan(data)) = -1;
209
210 fullFile = textscan(fopen(fullInputMatFileNameTeensy),'%s');
211 rawData = fullFile{1};
212 iter = 0;
213 for i=15:4:length(rawData)
214     iter = iter + 1;
215     row = rawData{i};
216     rowData = split(row,',')';
217     posarray(iter,1) = rowData(1,end);
218 end
219 posarray = string(posarray);
220
221 % Check if MHP data exists
222 if(max(data(:,4))>0)
223     % If data exists, iterate MHP number (in case more than one is used)
```

```
224     MHPNumber = MHPNumber+1;
225     processMHP(data, folderTeensy, baseNameNoExt,posarray);
226 else
227     % If no data exists, output as empty
228     outputMHP = [];
229 end
230
231 end
232
233 %% FUNC: processTeensy4young - Read Teensy data, determine which datasets
        exist
234 % Determine which of MultiHoleProbe or TPH suite exists
235 % INPUT
236 % * baseNameNoExt - Pix base file name with no extension or filepath
        details
237 function processTeensy4young(baseNameNoExt)
238 global MHPNumber
239 global TPHNumber
240 global TeensyNumber
241 global parsedTeensy
242 global YoungData
243
244 % Get the name of the file that the user wants to use.
245 [baseNameNoExtTeensy, ~, folderTeensy, fullInputMatFileNameTeensy]...
246     = file2open('*.csv','Select a Teensy Suite .CSV file');
247 % Read data in as matri=x
248 data = readmatrix(fullInputMatFileNameTeensy);
```

```
249 % Convert all NaN to -1
250 data(isnan(data)) = -1;
251
252 fullFile = textscan(fopen(fullInputMatFileNameTeensy),'%s');
253 rawData = fullFile{1};
254 iter = 0;
255 for i=15:4:length(rawData)
256     iter = iter + 1;
257     row = rawData{i};
258     rowData = split(row,',')';
259     posarray(iter,1) = rowData(1,end);
260 end
261 posarray = string(posarray);
262
263 % Check if MHP data exists
264 if(max(data(:,4))>0)
265     % If data exists, iterate MHP number (in case more than one is used)
266     MHPNumber = MHPNumber+1;
267     processyoung(data, folderTeensy, baseNameNoExt);
268 else
269     % If no data exists, output as empty
270
271 end
272
273 end
274
275
```

```
%% FUNC: CMHPCalc

function [vel, dir] = CMHPCalc(MHPData,rho)
global directionmethod
vel = zeros(size(MHPData(:,4)));
dir = zeros(size(MHPData(:,4)));
for i = 1:length(MHPData(:,4))
    pressure = [MHPData(i,4),MHPData(i,5),MHPData(i,6),MHPData(i,7),
    MHPData(i,8),MHPData(i,9),MHPData(i,10),MHPData(i,11)];
    posP = abs(pressure);
    spacing =
    [0;22.5;45;67.5;90;112.5;135;157.5;180;202.5;225;247.5;270;292.5;315;337.5];

    [Pmax,MaxTR] = max(posP);
    vel(i,1) = sqrt((2*Pmax)/rho);
    if Pmax<5&&i>1
        dir(i,1)=dir(i-1,1);
    elseif directionmethod == "BASIC"

        if MaxTR == 1
            if MHPData(i,3+MaxTR)>0
                dir(i,1) = 0;
            else
                dir(i,1) = 90;
            end
        elseif MaxTR == 2
            if MHPData(i,3+MaxTR)>0
```

```
300                 dir(i,1) = 22.5;
301             else
302                 dir(i,1) = 112.5;
303             end
304         elseif MaxTR == 3
305             if MHPData(i,3+MaxTR)>0
306                 dir(i,1) = 45;
307             else
308                 dir(i,1) = 135;
309             end
310         elseif MaxTR == 4
311             if MHPData(i,3+MaxTR)>0
312                 dir(i,1) = 67.5;
313             else
314                 dir(i,1) = 157.5;
315             end
316         elseif MaxTR == 5
317             if MHPData(i,3+MaxTR)>0
318                 dir(i,1) = 180;
319             else
320                 dir(i,1) = 270;
321             end
322         elseif MaxTR == 6
323             if MHPData(i,3+MaxTR)>0
324                 dir(i,1) = 202.5;
325             else
326                 dir(i,1) = 292.5;
```

```
327             end
328         elseif MaxTR == 7
329             if MHPData(i,3+MaxTR)>0
330                 dir(i,1) = 225;
331             else
332                 dir(i,1) = 315;
333             end
334         elseif MaxTR == 8
335             if MHPData(i,3+MaxTR)>0
336                 dir(i,1) = 247.5;
337             else
338                 dir(i,1) = 337.5;
339             end
340         end
341
342      elseif directionmethod == "LINEAR"
343         pressure = abs([MHPData(i,4);MHPData(i,5);MHPData(i,6);MHPData(i
    ,7);MHPData(i,4);MHPData(i,5);MHPData(i,6);MHPData(i,7);MHPData(i,8);
    MHPData(i,9);MHPData(i,10);MHPData(i,11);MHPData(i,8);MHPData(i,9);
    MHPData(i,10);MHPData(i,11)]);
344         spacing =
    [0;22.5;45;67.5;90;112.5;135;157.5;180;202.5;225;247.5;270;292.5;315;337.5];
345
346             if MaxTR==1
347                 diff1 = abs(pressure(MaxTR)-pressure(8));
348                 difftot=abs(pressure(MaxTR+1)-pressure(8));
```

```
349                 if diff1>difftot
350                 dir(i,1)=spacing(MaxTR);
351                 else
352                 dir(i,1)=(diff1/difftot)*45+spacing(8);
353                 end
354             else
355                 diff1 = abs(pressure(MaxTR)-pressure(MaxTR-1));
356                 difftot=abs(pressure(MaxTR+1)-pressure(MaxTR-1));
357                 if diff1>difftot
358                 dir(i,1)=spacing(MaxTR);
359                 else
360                 dir(i,1)=(diff1/difftot)*45+spacing(MaxTR-1);
361                 end
362             end
363     elseif directionmethod == "CPMATCH"
364             refcp = getCP(vel(i,1),0.0508); %for a 2in probe
365             refdiff = refcp(46,2)-refcp(length(refcp),2);
366
367             if MaxTR > 1 && MaxTR < 8
368
369                 cp1 = pressure(MaxTR-1)/Pmax;
370                 cp2 = pressure(MaxTR+1)/Pmax;
371                 diff1 = abs(cp1-refdiff);
372                 diff2 = abs(cp2-refdiff);
373
374             elseif MaxTR == 1
375
```

```
376                 cp1 = pressure(8)/Pmax;
377                 cp2 = pressure(MaxTR+1)/Pmax;
378                 diff1 = abs(cp1-refdiff);
379                 diff2 = abs(cp2-refdiff);
380             elseif MaxTR == 8
381
382                 cp1 = pressure(MaxTR-1)/Pmax;
383                 cp2 = pressure(1)/Pmax;
384                 diff1 = abs(cp1-refdiff);
385                 diff2 = abs(cp2-refdiff);
386             end
387
388                 if diff1>diff2
389                     [c,n]=min(abs(cp1-refcp(:,2)));
390                     diradj = refcp(n,1);
391                     if diradj > 22.5
392                         diradj = 22.5;
393                     end
394                     if pressure(MaxTR) > 0
395                         if MaxTR <=4
396                             dir(i,1) = spacing(MaxTR)-diradj;
397                         else
398                             dir(i,1) = spacing(MaxTR+4)-diradj;
399                         end
400                     else
401                         if MaxTR <=4
402                             dir(i,1) = spacing(MaxTR+4)-diradj;
```

```
403                     else
404                         dir(i,1) = spacing(MaxTR+8)-diradj;
405                     end
406                 end
407
408
409             else
410                 [c,n]=min(abs(cp1-refcp(:,2)));
411                 diradj = refcp(n,1);
412                 if diradj > 22.5
413                     diradj = 22.5;
414                 end
415                 if pressure(MaxTR) > 0
416                     if MaxTR <=4
417                         dir(i,1) = spacing(MaxTR)+diradj;
418                     else
419                         dir(i,1) = spacing(MaxTR+4)+diradj;
420                     end
421                 else
422                     if MaxTR <=4
423                         dir(i,1) = spacing(MaxTR+4)+diradj;
424                     else
425                         dir(i,1) = spacing(MaxTR+8)+diradj;
426                     end
427                 end
428             end
429
```

```
430             if dir(i,1) > 360
431                 dir(i,1) = dir(i,1)-360;
432             elseif dir(i,1) < 0
433                 dir(i,1) = dir(i,1) + 360;
434             end
435
436     end
437 end
438 end
439
440 %% FUNC: getCP
441
442 function [cp] = getCP(vel,L)
443 global CpData
444
445 kv = 0.00001488; %kenimatic viscoity in m^2/s
446
447 % % Read Cp vs. Theta data in as matrix
448 % CpData = readmatrix('RE#_plot'); %this file must be in the same root
      folder
449 % % Convert all NaN to -1
450 % CpData(isnan(CpData)) = -1;
451
452 Re = (vel*L)/kv;
453
454 [val,idx]=min(abs(CpData(1,2:end)-Re));
455
```

```
456 theta = CpData(3:end,1).';
457 refcps = CpData (3:end,1+idx).';
458 % inttheta = 0:0.5:112.5;
459 % intcps = interp1(theta,refcps,abs(inttheta));
460
461 inttheta = 0:0.5:112.5;
462 intcps = interp1(theta,refcps,abs(inttheta));
463
464 %theta = [0;25;30;40;50;60;70;75;80;85;90;95;100;110;120;140;150;160];
465 %refcps =
      [1;0.8;0.6;-0.375;-0.95;-1.6;-1.8;-1.74;-1.625;-1.5;-1.45;-1.42;-1.43;-1.46;-1.48;-
466
467 %ploting options
468 %plot (theta,refcps) %uncomment to see just selected Cp curve
469
470 % hold on %uncomment until hold off to plot all Cp data
471 % i=0;
472 % leg = strings(1,length(CpData(1,2:end)));
473 % for i = 2:length(CpData(1,:))
474 % cplot = CpData(3:end,i).';
475 % plot (theta,cplot);
476 % leg(1,i-1) = append("Re = ",string(CpData(1,i)));
477 % end
478 % legend (leg)
479 % hold off
480
```

```
481 cp = [inttheta;intcps].'; %put a breakpoint here to see Cp graphs.
482 end
483 %% FUNC: processyoung - Parse and process young data
484 % Parse MHP data based on parseRangeUTC data from Pixhawk
485 % Output external table (DateUTC, TimeUTC) to user workspace
486 % Preseve internal table (DatenumUTC) for other uses
487 % INPUT
488 % * data - Teensy full dataset
489 % * folder - folder containing raw Teensy data
490 % * baseNameNoExt - Pix base file name with no extension or filepath
      details
491 % OUTPUT
492 % * outputMHP - Internal version of resulting table (DatenumUTC)
493 function outputyoung = processyoung(data, folder, baseNameNoExt)
494
495 global parsedVars
496 global parseRangeUTC
497 global MHPNumber
498 global figIter
499 global Figs
500 global CMHPData
501 global Young_tableExternal
502
503 % Get number of Rows and Columns of the data
504 nrows = length(data(:,1));
505 ncols = length(data(1,:));
506
```

```
507 % Air desnity, pre-set for now but will be calc'ed from TPH data
508 rho=1.197; % kg/m3
509
510 % Find location of IMU, CTUN, GPS, and NKF1 tables (all required)
511 for j=1:length(parsedVars)
512     if(strcmpi('IMU_table',char(parsedVars{j,1})))
513         locIMU=j;
514     end
515     if(strcmpi('CTUN_table',char(parsedVars{j,1})))
516         locCTUN=j;
517     end
518     if(strcmpi('GPS_table',char(parsedVars{j,1})))
519         locGPS=j;
520     end
521     if(strcmpi('NKF1_table',char(parsedVars{j,1})))
522         locNKF1=j;
523     end
524     if(strcmpi('XKF1_table',char(parsedVars{j,1})))
525         locNKF1=j;
526     end
527 end
528
529 %Convert the parsedVars tables to arrays
530 % IMU = table2array(parsedVars{locIMU,2});
531 % CTUN = table2array(parsedVars{locCTUN,2});
532 % GPS = table2array(parsedVars{locGPS,2});
533 % NKF1 = table2array(parsedVars{locNKF1,2});
```

```
534
535 % Convert rawData into arrays of relevant data
536 time = data(:,1);
537 youngvel = data(:,2);
538 youngdir = data(:,3);
539 youngalt = data(:,4);
540 youngsos = data(:,5);
541 youngtemp = data(:,6);
542 UnixT = data(:,7);
543 PixT = data(:,8);
544
545 % Backfill Pixhawk BOOT and UNIX times via linear interpolation
546 filledPixT(:,1)=timeInterpolation(time(:,1),PixT(:,1));
547 filledUnixT(:,1)=timeInterpolation(time(:,1),UnixT(:,1));
548
549 % Add all relevant data to MHPData matrix
550 YoungData(:,1)=time;        % Sensor board time
551 YoungData(:,2)=filledPixT;  % Pix board time
552 YoungData(:,3)=filledUnixT; % Pix Unix Time (GPS)
553 YoungData(:,4)=youngvel;
554 YoungData(:,5)=youngdir;
555 YoungData(:,6)=youngalt;
556 YoungData(:,7)=youngsos;
557 YoungData(:,8)=youngtemp;
558
559 %Average every two datapoints
560 YoungDataTemp = zeros(size(YoungData));
```

```
561
562 for c = 4:8
563     YoungDataTemp(1,c)= YoungData(1,c);
564     for i = 2:length(YoungData(:,c))-1
565         YoungDataTemp(i,c) = (YoungData(i+1,c)+YoungData(i,c)+YoungData(i
      -1,c))/3;
566     end
567     YoungDataTemp(i+1,c)= YoungData(i+1,c);
568 end
569
570 for c = 4:8
571     YoungData(:,c) = YoungDataTemp(:,c);
572 end
573
574
575 % Cut out trailing data that wasnt interpolated
576 YoungData(isnan(YoungData(:,3)),:)=[];
577
578 % Generate DateUTC and TimeUTC datasets for user view
579 Young_DateTime=datetime(YoungData(:,3),...
580     'ConvertFrom','posixTime','Format','MMM-dd-yyyy HH:mm:ss.S');
581 Young_Date=datestr(Young_DateTime,'mmm-dd-yyyy');
582 Young_Time=datestr(Young_DateTime,'HH:MM:SS.FFF');
583
584 % % Generate DatenumUTC for internal use
585 datenumYoung = datenum(Young_DateTime);
586
```

```
587 % Find MHP start time from Pixhawk parse
588
589 TO_Young = 1;
590
591
592 % Find MHP end time from Pixhawk parse
593
594 LND_Young = length(YoungData);
595
596
597 % Parse all MHP data based on Pixhawk parsing
598 Young_entry = YoungData(TO_Young:LND_Young,:);
599 Young_Date = Young_Date(TO_Young:LND_Young,:);
600 Young_Time = Young_Time(TO_Young:LND_Young,:);
601 Young_time_out = (Young_entry(:,1)-min(Young_entry(:,1)))/1000;
602 datenumYoung = datenumYoung(TO_Young:LND_Young,:);
603
604 % Create external table for user view
605 Young_tableExternal = table(Young_entry(:,1),Young_entry(:,2),
      Young_time_out,Young_Date,Young_Time,datenumYoung,...
606      Young_entry(:,4),Young_entry(:,5),Young_entry(:,6),Young_entry(:,7),
      Young_entry(:,8),...
607      'VariableNames', {'Board Time from PowerUp (msec)',...
608      'Pix Time from PowerUp (msec)','Pix time from parse','UTC Date','UTC
      Time','datenum'...
609      'Velocity','Direction','Wind Elevation (deg)','Speed of Sound (m/s)','
      Temp (C)'});
```

```
610 Young_tableExternal.Properties.Description = sprintf('MHP%d',MHPNumber);
611 % Output table to user workspace ----------------------------asignin
      --------
612 assignin('base',sprintf('Young_table%d',MHPNumber),Young_tableExternal);
613 % Save tabele for user review
614 table2saveCSV(baseNameNoExt, folder, Young_tableExternal)
615
616 YoungData = [Young_entry(:,1),Young_entry(:,2),Young_time_out,Young_Date,
      Young_Time,Young_entry(:,4),Young_entry(:,5),Young_entry(:,6),
      Young_entry(:,7),Young_entry(:,8)];
617
618 end
619
620 %% FUNC: timeInterpolation - Uses board time to interpolate other time
621 % externTime is assumed to be the same length as board time
622 % Back fills empty data in externTime by:
623 % * Using boardTime as the constant interpolated against
624 % * Using partial externTime as the interpolant, or truth data
625 % INPUT
626 % * boardTime - highspeed board time
627 % * externTime - same length as boardTime, but different timing variable
628 % OUTPUT
629 % * interpolatedArray - externTime with empties filled
630
631 function interpolatedArray = timeInterpolation(boardTime,externTime)
632
633 % Initialize the counter for output array
```

```
634 Interpcount=0;
635
636 % Interpolation BuildUp
637 for i=1:length(boardTime(:,1))
638     boardTimeInt = boardTime(i);   % Logger time (Teensy, Arduino, etc.)
639     externTimeInt = externTime(i); % External time (Pix, Unix, GPS, etc.)
640
641     % If valid timing value, add to interpolation array
642     if(externTimeInt~=-1)
643         Interpcount=Interpcount+1;
644         InterpData(Interpcount,1)=boardTimeInt;   % Sensor board time
645         InterpData(Interpcount,2)=externTimeInt;  % External Time
646     end
647 end
648
649 % Remove Duplicate External Time interpolation points
650 NewVals=unique(InterpData(:,2));
651 % Concatenate data based on unique datapoints only
652 for i=1:length(NewVals(:,1))
653     TempVal = find(InterpData(:,2)==NewVals(i,1),1,'first');
654     conCat(i,1) = InterpData(TempVal,1);
655     conCat(i,2) = NewVals(i,1);
656 end
657
658 % Backfill gaps in full dataset
659 for j = 1:length(boardTime(:,1))
660     if(externTime(j,1) == -1)
```

```
661         externTime(j,1) = interp1(conCat(:,1),conCat(:,2),boardTime(j,1),'
    linear');
662     end
663 end
664
665 % Output the interpolated External Time array
666 interpolatedArray = externTime;
667
668 end
669
670 %% FUNC: file2open - Open File Based on File Input Type
671 % Prompt user to open a file of type "TYPE" with a heading of "TEXT"
672 % Saves file data as output
673 % INPUT
674 % * Type - File type to isolate for user to choose. Can be an array.
675 % * * Types: .csv, .txt, .xlsx, etc
676 % * * User will only be open the types provided
677 % * Text - Title of the UI figure that opens
678 % OUTPUT
679 % * baseNameNoExt - base file name with no extension or filepath details
680 % * baseName - base file name with extension but no filepath
681 % * folder - filepath to the file, but without any file information
682 % * fullInputMatName - file name with filepath and extension added
683 % * * These are used as passthrough to other functions
684 function [baseNameNoExt, baseName, folder, fullInputMatName] = file2open(
    type,text)
685
```

```
686 global startingFolder
687
688 % Get the name of the file that the user wants to use.
689 defaultName = fullfile(startingFolder,type);
690 % Grab baseName and folder directly from the loading process
691 [baseName, folder] = uigetfile(defaultName, text);
692
693 % Redefine starting folder as current folder
694 % Allows next file open to start in same folder
695 startingFolder = folder;
696
697 if baseName == 0
698     % User clicked the Cancel button.
699     return;
700 end
701
702 % Remove extension from the baseName
703 [~, baseNameNoExt, ~] = fileparts(baseName);
704
705 % Recombine all parts to recreate the full name
706 fullInputMatName = fullfile(folder, baseName);
707
708 end
709 %% FUNC: file2save - Save File Based on file2open Details
710 % Take file components from file2open and save in same location
711 % INPUT
712 % * baseNameNoExt - base file name with no extension or filepath details
```

```
% * folder - filepath to the file with no file information
% * varToSave - variable with data to save externally
function file2save(baseNameNoExt, folder, varToSave)

% Get the name of the intput.mat file and save as input_parsed.mat
baseFileName = sprintf('%s_Parsed.mat', baseNameNoExt);
% Generate output file name using folder details and new name
fullParsedMatFileName = fullfile(folder, baseFileName);
% Save file with parsed data as the original filename plus the added
    portion
save(fullParsedMatFileName,'varToSave');
end
%% FUNC: table2saveCSV - Save Specific Tables from Workspace To .CSV File
% Must be table (no arrays or structures)
% Before loading, define 'yourTable.Properties.Description = yourVarName;'
% * yourVarName will be appended after DFL file name as new .CSV file
% * * Ex: DFL Name: NimbusFlight2_5_27_2021.bin
% * *     varName : GPS
% * *     output  : NimbusFlight2_5_27_2021_varName.csv
% INPUT
% * baseNameNoExt - base file name with no extension or filepath details
% * folder - filepath to the file with no file information
% * tableToSave - table with data to save externally
function table2saveCSV(baseNameNoExt, folder, tableToSave)

% If Description of table is empty, use preset name. Else, use the name
if(isempty(tableToSave.Properties.Description))
```

```
739     varName = 'undefinedVar';
740 else
741     varName = tableToSave.Properties.Description;
742 end
743
744 % Save file as "PixhawkDFLname_VARNAME.csv"
745 % VARNAME can be undefinedVar if description is not set
746 baseFileName = sprintf('%s_%s.csv', baseNameNoExt, varName);
747 fullOutputMatFileName = fullfile(folder, baseFileName);
748 % Write data to .csv file
749 writetable(tableToSave, fullOutputMatFileName);
750
751 end
752 %% FUNC: CMHPCalc
753
754 function [vel, dir] = CMHPCalc(MHPData,rho)
755 global directionmethod
756 vel = zeros(size(MHPData(:,4)));
757 dir = zeros(size(MHPData(:,4)));
758 for i = 1:length(MHPData(:,4))
759     pressure = [MHPData(i,4),MHPData(i,5),MHPData(i,6),MHPData(i,7),
      MHPData(i,8),MHPData(i,9),MHPData(i,10),MHPData(i,11)];
760     posP = abs(pressure);
761     spacing =
      [0;22.5;45;67.5;90;112.5;135;157.5;180;202.5;225;247.5;270;292.5;315;337.5];

762     [Pmax,MaxTR] = max(posP);
```

```
763     vel(i,1) = sqrt((2*Pmax)/rho);
764     if Pmax<5&&i>1
765         dir(i,1)=dir(i-1,1);
766     elseif directionmethod == "BASIC"
767
768         if MaxTR == 1
769             if MHPData(i,3+MaxTR)>0
770                 dir(i,1) = 0;
771             else
772                 dir(i,1) = 90;
773             end
774         elseif MaxTR == 2
775             if MHPData(i,3+MaxTR)>0
776                 dir(i,1) = 22.5;
777             else
778                 dir(i,1) = 112.5;
779             end
780         elseif MaxTR == 3
781             if MHPData(i,3+MaxTR)>0
782                 dir(i,1) = 45;
783             else
784                 dir(i,1) = 135;
785             end
786         elseif MaxTR == 4
787             if MHPData(i,3+MaxTR)>0
788                 dir(i,1) = 67.5;
789             else
```

```
790                 dir(i,1) = 157.5;
791             end
792         elseif MaxTR == 5
793             if MHPData(i,3+MaxTR)>0
794                 dir(i,1) = 180;
795             else
796                 dir(i,1) = 270;
797             end
798         elseif MaxTR == 6
799             if MHPData(i,3+MaxTR)>0
800                 dir(i,1) = 202.5;
801             else
802                 dir(i,1) = 292.5;
803             end
804         elseif MaxTR == 7
805             if MHPData(i,3+MaxTR)>0
806                 dir(i,1) = 225;
807             else
808                 dir(i,1) = 315;
809             end
810         elseif MaxTR == 8
811             if MHPData(i,3+MaxTR)>0
812                 dir(i,1) = 247.5;
813             else
814                 dir(i,1) = 337.5;
815             end
816         end
```

```
817
818     elseif directionmethod == "LINEAR"
819         pressure = abs([MHPData(i,4);MHPData(i,5);MHPData(i,6);MHPData(i
    ,7);MHPData(i,4);MHPData(i,5);MHPData(i,6);MHPData(i,7);MHPData(i,8);
    MHPData(i,9);MHPData(i,10);MHPData(i,11);MHPData(i,8);MHPData(i,9);
    MHPData(i,10);MHPData(i,11)]);
820         spacing =
    [0;22.5;45;67.5;90;112.5;135;157.5;180;202.5;225;247.5;270;292.5;315;337.5];
821
822         if MaxTR==1
823             diff1 = abs(pressure(MaxTR)-pressure(8));
824             difftot=abs(pressure(MaxTR+1)-pressure(8));
825             if diff1>difftot
826                 dir(i,1)=spacing(MaxTR);
827             else
828                 dir(i,1)=(diff1/difftot)*45+spacing(8);
829             end
830         else
831             diff1 = abs(pressure(MaxTR)-pressure(MaxTR-1));
832             difftot=abs(pressure(MaxTR+1)-pressure(MaxTR-1));
833             if diff1>difftot
834                 dir(i,1)=spacing(MaxTR);
835             else
836                 dir(i,1)=(diff1/difftot)*45+spacing(MaxTR-1);
837             end
838         end
```

```
839     elseif directionmethod == "CPMATCH"
840         refcp = getCP(vel(i,1),0.0508); %for a 2in probe
841         refdiff = refcp(46,2)-refcp(length(refcp),2);
842         refcp = sortrows(refcp,2);
843 
844         if MaxTR > 1 && MaxTR < 8
845 
846             cp1 = pressure(MaxTR-1)/Pmax;
847             cp2 = pressure(MaxTR+1)/Pmax;
848             diff1 = abs(cp1-refdiff);
849             diff2 = abs(cp2-refdiff);
850 
851         elseif MaxTR == 1
852 
853             cp1 = pressure(8)/Pmax;
854             cp2 = pressure(MaxTR+1)/Pmax;
855             diff1 = abs(cp1-refdiff);
856             diff2 = abs(cp2-refdiff);
857         elseif MaxTR == 8
858 
859             cp1 = pressure(MaxTR-1)/Pmax;
860             cp2 = pressure(1)/Pmax;
861             diff1 = abs(cp1-refdiff);
862             diff2 = abs(cp2-refdiff);
863         end
864 
865         if diff1>diff2
```

```
866             [c,n]=min(abs(cp1-refcp(:,2)));
867             diradj = refcp(n,1);
868             if diradj > 22.5
869                 diradj = 22.5;
870             end
871             dir(i,1) = spacing(MaxTR)-diradj;
872         else
873             [c,n]=min(abs(cp1-refcp(:,2)));
874             diradj = refcp(n,1);
875             if diradj > 22.5
876                 diradj = 22.5;
877             end
878             dir(i,1) = spacing(MaxTR)+diradj;
879         end
880         if dir(i,1) > 360
881             dir(i,1) = dir(i,1)-360;
882         elseif dir < 0
883             dir(i,1) = dir(i,1) + 360;
884         end
885
886     end
887 end
888 end
889
890 %% FUNC: getCP
891
892 function [cp] = getCP(vel,L)
```

```
893 kv = 0.00001488; %kenimatic viscoity in m^2/s
894
895 % Read Cp vs. Theta data in as matrix
896 data = readmatrix('RE#_plot'); %this file must be in the same root folder
897 % Convert all NaN to -1
898 data(isnan(data)) = -1;
899
900 Re = (vel*L)/kv;
901
902 [val,idx]=min(abs(data(1,2:end)-Re));
903
904 theta = data(3:end,1).';
905 refcps = data (3:end,1+idx).';
906 % inttheta = 0:0.5:112.5;
907 % intcps = interp1(theta,refcps,abs(inttheta));
908
909 inttheta = 0:0.5:112.5;
910 intcps = interp1(theta,refcps,abs(inttheta));
911
912 %theta = [0;25;30;40;50;60;70;75;80;85;90;95;100;110;120;140;150;160];
913 %refcps =
      [1;0.8;0.6;-0.375;-0.95;-1.6;-1.8;-1.74;-1.625;-1.5;-1.45;-1.42;-1.43;-1.46;-1.48;-
914
915 %ploting options
916 %plot (theta,refcps) %just selected
917
```

```
% hold on %all Cp data
% i=0;
% leg = strings(1,length(data(1,2:end)));
% for i = 2:length(data(1,:))
% cplot = data(3:end,i).';
% plot (theta,cplot);
% leg(1,i-1) = append("Re = ",string(data(1,i)));
% end
% legend (leg)
% hold off

cp = [inttheta;intcps].';
end

%% FUNC: ardupilogConvert - Loads in .BIN DFL, converts to Matlab data
% Asks user to find Pixhawk .BIN DataFlash Log files
% Converts the Binary to custom Matlab cell structure
% Stores data in cell structure for other functions to use
function [baseNameNoExt, baseName, folder, fullInputMatName] =
    ardupilogConvert()

global rawDFL

% Opens Pixhawk DFL file, saving the parts for use later (if we choose to)
[baseNameNoExt, baseName, folder,fullInputMatName] = file2open('*.bin',...
    'Select a .BIN Pixhawk DFL file');

```

```
944 % Convert the DFL (in binary) to custom Matlab cell structure
945 rawDFL = Ardupilog(fullInputMatName);
946
947 end
948 %% FUNC: startProcessing - Process Data Based on User Selection
949 % Take outputs from userSelection and begin processing
950 % INPUT
951 % * fig - Handle for the UI Figure
952 % * tabgp - Handle for the UITabGroup that runs each panel
953 % * iMetspn - Send number chosen from UI to iterative iMet read loop
954 % * spnTeensy - Send number chosen from UI to iterative Teensy read loop
955 function startProcessing()
956
957 global rawDFL
958 global rawVars
959 global parsedVars
960 global arduPilotType
961 global redactStructDFL
962
963 [baseNameNoExt, baseName, folder, fullInputMatName] = ardupilogConvert();
964
965 % Disable non-critical warning to ease user view
966 warning('off','MATLAB:structOnObject')
967
968 % Generate names from the DFL entires
969 fieldNames = fieldnames(rawDFL);
970 % Sort entries to match standard DFL output, remove static non-entries
```

```
971 fieldNames = sort(fieldNames(11:end,:));
972 % Determine length of DFL after static, unused data is removed
973 len = length(fieldNames);
974
975 for i=1:len
976
977     % If no data is in the LineNo column, skip to next dataset
978     if(isempty(rawDFL.(fieldNames{i}).LineNo))
979         % If the empty is CTUN, warn that ArduCopter doesnt use it at all
980         if(contains(rawDFL.(fieldNames{i}).name,'CTUN'))
981             warning('CTUN data not available for ArduCopter files; no
    airspeed data.');
982         end
983         continue
984     end
985
986     % Generate redacted structure of non-empty data
987     redactStructDFL.(fieldNames{i}) = struct(getfield(rawDFL,fieldNames{i
    }));
988
989 end
990
991 assignin('base','redactStructDFL',redactStructDFL);
992
993 % Initialize check for ArduPilot Type
994 Rover = 0;
995 Copter = 0;
```

```
996 QuadPlane = 0;
997 Plane = 0;
998
999 % Scan message log to determine what ArduPilot type it is
1000 for i=1:length(redactStructDFL.MSG.LineNo);
1001     if(contains(redactStructDFL.MSG.Message(i,1:length(redactStructDFL.MSG
        .Message(1,:))),'ArduCopter'))
1002         Copter = 1;
1003     elseif(contains(redactStructDFL.MSG.Message(i,1:length(redactStructDFL
        .MSG.Message(1,:))),'QuadPlane'))
1004         QuadPlane = 1;
1005     elseif(contains(redactStructDFL.MSG.Message(i,1:length(redactStructDFL
        .MSG.Message(1,:))),'ArduRover'))
1006         Rover = 1;
1007     elseif(contains(redactStructDFL.MSG.Message(i,1:length(redactStructDFL
        .MSG.Message(1,:))),'ArduPlane'))
1008         Plane = 1;
1009     end
1010 end
1011
1012 % Depending on what values were found, set ArduPilot type
1013 % This structure prevents false analysis as more than 1 can be present
1014 if(Rover == 1)
1015     arduPilotType = 'ArduRover';
1016 elseif(Copter == 1)
1017     arduPilotType = 'ArduCopter';
1018 elseif(QuadPlane == 1)
```

```
1019     arduPilotType = 'QuadPlane';
1020 elseif(Plane == 1)
1021     arduPilotType = 'FixedWing';
1022 end
1023
1024 fieldNamesIter = fieldnames(redactStructDFL);
1025 % Counter for variables copmpleted processing
1026 k=0;
1027 for i=1:length(fieldNamesIter)
1028     % If an un-selected checkbox, dont process the data
1029     if(strcmpi(fieldNamesIter{i},'GPS') | strcmpi(fieldNamesIter{i},'CTUN'
        )...
1030             | strcmpi(fieldNamesIter{i},'ATT') | strcmpi(fieldNamesIter{i
        },...
1031             'RCOU') | strcmpi(fieldNamesIter{i},'IMU') |...
1032             strcmpi(fieldNamesIter{i},'NKF1') |...
1033             strcmpi(fieldNamesIter{i},'BARO') | strcmpi(fieldNamesIter{i},
        'XKF1'))
1034
1035         % Keep track of valid entries
1036         k=k+1;
1037         rawVarLoop(fieldNamesIter{i},k);
1038     end
1039 end
1040
1041 % Turn warnings on that we turned off previously
1042 warning('on','MATLAB:structOnObject')
```

```
% Set target variables to find location in rawVars structure
target = {'GPS_table','RCOU_table','ATT_table','CTUN_table','IMU_table','
    BARO_table'};

% Find location in rawVars structure that targets reside in
% Store location data for later use
for i=1:length(target)
    for j=1:length(rawVars)
        res(j,1)= strcmpi(target(i),char(rawVars{j,1}));
        if(res(j,1) == 1)
            loc(i,1)=j;
            break
        end
    end
end

% If CTUN wasnt found, dont use
if(loc(4)==0)
    chooseSubset(rawVars{loc(1),2}, rawVars{loc(2),2},...
        rawVars{loc(3),2}, rawVars{loc(5),2}, rawVars{loc(6),2})
else
    chooseSubset(rawVars{loc(1),2}, rawVars{loc(2),2},...
        rawVars{loc(3),2}, rawVars{loc(5),2}, rawVars{loc(6),2}, rawVars{
    loc(4),2})
end

% Parse data using endpoints from chooseSubset across data chosen in UI
```

```
1068 for i=1:length(rawVars)
1069     % Parse data based on selection
1070     dataParsePix(i);
1071 end
1072
1073 % Save current parsedVars array as external Parsed file for user review
1074 file2save(baseNameNoExt, folder, parsedVars);
1075
1076
1077
1078 processTeensy(baseNameNoExt);
1079
1080
1081 end
1082 %% FUNC: chooseSubset - User-Chosen Start and End Parse Points
1083 % Generate 4 base plots, user selects start and end points
1084 % Start and End points saved to parseRangeUTC for parsing
1085 % INPUT
1086 % * GPS - raw GPS data, used for altitude and groundspeed data
1087 % * RCOU - raw RCOU data, used for throttle output data (converted to %)
1088 % * ATT - raw ATT data, used for state variables (degrees roll, pitch, etc
      )
1089 % * IMU - raw IMU data, used for highest resolution timing data
1090 % * BARO - raw BARO data, used for most accurate altitude data
1091 % * CTUN - raw CTUN data (in varargin, optional), used for airspeed data
1092 function chooseSubset(GPS, RCOU, ATT, IMU, BARO, varargin)
1093
```

```
1094 global figIter
1095 global Figs
1096 global parseRangeUTC
1097
1098 % Generate new figure window, iterate global to prevent overwrite
1099 figIter = figIter+1;
1100 Figs{figIter}=figure(figIter);
1101 Figs{figIter}.Name = 'Raw data from DFL. Click on graph for upper and
       lower bound for parsing.';
1102
1103 % If no CTUN data (ArduCopter), fill with array of zeros
1104 if(isempty(varargin))
1105     CTUN = [zeros(1000,2) linspace(IMU(1,3),IMU(end,4),1000)' zeros
       (1000,9)];
1106 else
1107     CTUN = varargin{1};
1108 end
1109
1110 % Initialize guess max and min throttle (accounts for max never achieved)
1111 thrMinPWM = 1100;
1112 thrMaxPWM = 1900;
1113 thrPercent(:,1) = (RCOU(:,7)-thrMinPWM)/(thrMaxPWM-thrMinPWM)*100;
1114
1115 % Groundspeed and Airspeed plot
1116 plt1 = subplot(4,1,1);
1117 plot(GPS(:,3),GPS(:,13),'b',CTUN(:,3),CTUN(:,12),'r')
1118 title('Groundspeed, Airspeed vs Time')
```

```
1119 ylabel({'Groundspeed (blue)';'Airspeed (red)';'(m/s)'})
1120
1121 % Throttle Output
1122 plt2 = subplot(4,1,2);
1123 plot(RCOU(:,3),thrPercent(:,1),'b')
1124 title('Throttle vs Time')
1125 ylabel({'Throttle';'(%)'})
1126 ylim([0 100])
1127
1128 % For the dotted line along x-axis zero point of pitch plot
1129 zeroPitch=int8(zeros(length(ATT(:,3)),1));
1130
1131 % Aircraft Pitch angle: Can change ylim to something more relevant.
1132 % TIV uses -20 to 50 to see high AoA landing
1133 plt3 = subplot(4,1,3);
1134 plot(ATT(:,3),ATT(:,8),'b',ATT(:,3),zeroPitch,'r:')
1135 title('Aircraft Pitch Angle vs Time')
1136 ylabel({'Aircraft Pitch';'Angle (  )'})
1137 ylim([-10 40])
1138
1139 % Altitude plot (GPS, left side)
1140 plt4 = subplot(4,1,4);
1141 yyaxis left
1142 plot(GPS(:,3),GPS(:,12),'b');
1143 ylim([min(GPS(:,12))-25 max(GPS(:,12))+25])
1144 ylabel({'GPS Altitude (blue)';'m MSL'})
1145 title('Altitude vs Time')
```

```
1146
1147 % Altitude plot (BARO, right side)
1148 yyaxis right
1149 plot(BARO(:,3),BARO(:,5),'r')
1150 ylim([min(BARO(:,5))-25 max(BARO(:,5))+25])
1151 ylabel({'BARO Altitude (red)';'m AGL'})
1152 xlabel('Time (seconds)')
1153 linkaxes([plt1 plt2 plt3 plt4],'x')
1154 xlim([min(GPS(:,3)) max(GPS(:,3))])
1155
1156 % Initialize parsing counter, max out at two (one start, one end)
1157 m=0;
1158 % Loop allowing user to select start and end points
1159 while true
1160     % Grab horiz and vert information from plot, use as output for parsing
1161     % button tracks mouse clicks
1162     [horiz, vert, button] = ginput(1);
1163     % If user closes window with no data or only one data point, exit
1164     if isempty(horiz) || button(1) == 3; break; end
1165     % User clicked a valid entry, so iterate and continue
1166     m = m+1;
1167     % Save x value of data point clicked, use to find respective Y points
1168     x_m(m) = horiz(1);
1169     % Prevent plot updates until all locations are found
1170     hold on
1171     y_va(m)=CTUN(find(CTUN(:,3)>=x_m(m),1,'first'),12);      % Airspeed
1172     y_vg(m)=GPS(find(GPS(:,3)>=x_m(m),1,'first'),13);       % Groundspeed
```

```
1173  y_thr(m)=RCOU(find(RCOU(:,3)>=x_m(m),1,'first'),7);      % Throttle
      Percent
1174  y_pitch(m)=ATT(find(ATT(:,3)>=x_m(m),1,'first'),8); % Aircraft Pitch
1175  y_GPSalt(m)=GPS(find(GPS(:,3)>=x_m(m),1,'first'),12);     % GPS
      Altitude
1176  y_BAROalt(m)=BARO(find(BARO(:,3)>=x_m(m),1,'first'),5);     % BARO
      Altitude
1177
1178  % Replot same base graphs, but update with X markers at chosen
      location
1179  % Groundspeed plot
1180  subplot(4,1,1)
1181  plot(GPS(:,3),GPS(:,13),'b',CTUN(:,3),CTUN(:,12),'r',x_m,y_vg,'kx',x_m
      ,y_va,'kx')
1182  title('Groundspeed, Airspeed vs Time')
1183  ylabel({'Groundspeed (blue)';'Airspeed (red)';'(m/s)'})
1184
1185  % Throttle Output
1186  subplot(4,1,2)
1187  plot(RCOU(:,3),thrPercent(:,1),'b',x_m, ((y_thr-thrMinPWM)/(thrMaxPWM-
      thrMinPWM)*100),'kx')
1188  title('Throttle vs Time')
1189  ylabel({'Throttle';'(%)'})
1190  ylim([0 100])
1191
1192  % Aircraft Pitch angle: Can change ylim to something more relevant.
1193  % TIV uses -20 to 50 to see high AoA landing
```

```
1194    subplot(4,1,3)
1195    plot(ATT(:,3),ATT(:,8),'b',ATT(:,3),zeroPitch,'r:',x_m,y_pitch,'kx')
1196    title('Aircraft Pitch Angle vs Time')
1197    ylabel({'Aircraft Pitch';'Angle (  )'})
1198    ylim([-10 40])
1199
1200    % Altitude plot (GPS, left side)
1201    plt4 = subplot(4,1,4);
1202    yyaxis left
1203    plot(GPS(:,3),GPS(:,12),'b',x_m,y_GPSalt,'kx');
1204    ylim([min(GPS(:,12))-25 max(GPS(:,12))+25])
1205    ylabel({'GPS Altitude (blue)';'m MSL'})
1206    title('Altitude vs Time')
1207
1208    % Altitude plot (BARO, right side)
1209    yyaxis right
1210    plot(BARO(:,3),BARO(:,5),'r',x_m,y_BAROalt,'kx')
1211    ylim([min(BARO(:,5))-25 max(BARO(:,5))+25])
1212    ylabel({'BARO Altitude (red)';'m AGL'})
1213    xlabel('Time (seconds)')
1214    linkaxes([plt1 plt2 plt3 plt4],'x')
1215    xlim([min(GPS(:,3)) max(GPS(:,3))])
1216
1217    % Update plots now that all locations are found
1218    drawnow
1219
1220    % If both start and end are chosen, exit loop
```

```
1221      if(m>=2)
1222          break;
1223      end
1224
1225  end
1226
1227  % Create new figure to show only parsed data
1228  figIter = figIter+1;
1229  Figs{figIter}=figure(figIter);
1230  Figs{figIter}.Name = 'Preview of user-parsed DFL data.';
1231
1232  % Recreate all plots using only parsed data
1233  % Groundspeed plot
1234  plt1 = subplot(4,1,1);
1235  plot(GPS(:,3)-x_m(1),GPS(:,13),'b',CTUN(:,3)-x_m(1),CTUN(:,12),'r')
1236  title('Groundspeed, Airspeed vs Time')
1237  ylabel({'Groundspeed (blue)';'Airspeed (red)';'(m/s)'})
1238
1239  % Throttle Output
1240  plt2 = subplot(4,1,2);
1241  plot(RCOU(:,3)-x_m(1),thrPercent(:,1),'b')
1242  title('Throttle vs Time')
1243  ylabel({'Throttle';'(%)'})
1244  ylim([0 100])
1245
1246  % Aircraft Pitch angle: Can change ylim to something more relevant.
1247  % TIV uses -20 to 50 to see high AoA landing
```

```
plt3 = subplot(4,1,3);
plot(ATT(:,3)-x_m(1),ATT(:,8),'b',ATT(:,3)-x_m(1),zeroPitch,'r:')
title('Aircraft Pitch Angle vs Time')
ylabel({'Aircraft Pitch';'Angle (  )'})
ylim([-10 40])

% Altitude plot (GPS, left side)
plt4 = subplot(4,1,4);
yyaxis left
plot(GPS(:,3)-x_m(1),GPS(:,12),'b');
ylim([min(GPS(:,12))-25 max(GPS(:,12))+25])
ylabel({'GPS Altitude (blue)';'m MSL'})
title('Altitude vs Time')

% Altitude plot (BARO, right side)
yyaxis right
plot(BARO(:,3)-x_m(1),BARO(:,5),'r')
ylim([min(BARO(:,5))-25 max(BARO(:,5))+25])
ylabel({'BARO Altitude (red)';'m AGL'})
xlabel('Time (seconds)')
linkaxes([plt1 plt2 plt3 plt4],'x')
xlim([0 x_m(2)-x_m(1)])

% If starting data point is glitched, find first non-glitched point
checkLoc1 = find(IMU(:,3)>x_m(1),1,'first')-1;
while(IMU(checkLoc1+1,3)-IMU(checkLoc1,3)>=1)
    checkLoc1 = find(IMU(checkLoc1+5:end,3)>x_m(1),1,'first')-1;
```

```
1275 end
1276
1277 % Save IMU-based DatenumUTC, TimeS, LineNo data (highest resolution)
1278 parseRangeUTC(1,1)=IMU(checkLoc1,4); % Highest resolution DatenumUTC start
1279 parseRangeUTC(1,3)=IMU(checkLoc1,3); % Highest resolution TimeS start
1280 parseRangeUTC(1,5)=IMU(checkLoc1,1); % Highest resolution LineNo start
1281
1282 % If ending data point is glitched, find last non-glitched point
1283 checkLoc2 = find(IMU(:,3)>x_m(2),1,'first')-1;
1284 while(IMU(checkLoc2+1,3)-IMU(checkLoc2,3)>=1)
1285     checkLoc2 = find(IMU(checkLoc2+5:end,3)>x_m(2),1,'first')-1;
1286 end
1287
1288 % Save IMU-based DatenumUTC, TimeS, LineNo data (highest resolution)
1289 parseRangeUTC(1,2)=IMU(checkLoc2,4); % Highest resolution DatenumUTC end
1290 parseRangeUTC(1,4)=IMU(checkLoc2,3); % Highest resolution TimeS end
1291 parseRangeUTC(1,6)=IMU(checkLoc2,1); % Highest resolution LineNo end
1292
1293 end
1294 %% FUNC: dataParsePix - Redact Variable Set by End Points
1295 % Parse Pixhawk data based on parseRangeUTC start and end points
1296 % INPUT
1297 % * i - iteration in loop acted on, gives location in rawVars to grab data
1298 function dataParsePix(i)
1299
1300 global parseRangeUTC
1301 global parsedVars
```

```
1302 global rawVars
1303
1304 % Grab and convert rawVar data into useful data for parsing
1305 varName = rawVars{i,1};
1306 varData = rawVars{i,2};
1307 varFields(:,1) = rawVars{i,3};
1308
1309 % Finf effective start (TO) and end (LND) times based on time comparisons
1310 TO = find(varData(:,1)>parseRangeUTC(5),1,'first');
1311 LND = find(varData(:,1)>parseRangeUTC(6),1,'first');
1312
1313 % Parse all columns in data set by the row numbers generated above
1314 varData = varData(TO:LND,:);
1315
1316 % Convert col3 (TimeS) into TimeSinceParse (seconds)
1317 % Very useful metric for data plots
1318 varData(:,3) = varData(:,3)-parseRangeUTC(1,3);
1319 % Generate internally-referenced table (no Date or Time, just DatenumUTC)
1320 varDataInternal = array2table(varData,'VariableNames',varFields);
1321 parsedVars{i,1} = varName;
1322 parsedVars{i,2} = varDataInternal;
1323 parsedVars{i,3} = varFields;
1324 varDataInternal.Properties.Description = strrep(varName,'_table','');
1325
1326 % For external table, generate discrete DateUTC and TimeUTC values
1327 var_DateTime(:,1) = datetime(varData(:,4),'ConvertFrom','datenum');
1328 TimeUTC(:,1) = datetime(var_DateTime,'Format','HH:mm:ss.SSS');
```

```
1329 DateUTC(:,1) = datetime(var_DateTime,'Format','MMM-dd-yyy');
1330
1331 % Save table with discrete DateUTC and TimeUTC for user view
1332 varDataExternal = removevars(varDataInternal,{'DatenumUTC'});
1333 varDataExternal = addvars(varDataExternal,DateUTC,TimeUTC,'After','TimeS')
        ;
1334
1335 % Send external version to user workspace
1336 assignin('base',varName,varDataExternal);
1337
1338 end
1339 %% FUNC: rawVarLoop - Function to parse specific data from redactStructDFL
1340 % Parse over NKF1, GPS, IMU, CTUN, ATT, BARO, RCOU
1341 % INPUT
1342 % * name - Fieldname that is housed in redactStructDFL
1343 % * k - iterator from main loop of which valid entry is active
1344 function rawVarLoop(name,k)
1345
1346 global rawVars
1347 global redactStructDFL
1348
1349 % Use as varName for parsing purposes
1350 varName = name;
1351
1352 % Get varName data from structDFL
1353 % Use it to get the fieldUnits array or variable names
1354 % Convert the array to a structure (in case it isnt already)
```

```
1355 % Use "fieldnames" to export the data as a cell array
1356 posArray = fieldnames(struct(getfield(getfield(redactStructDFL,varName),'
       fieldUnits')));
1357 % Add in extra Variable Names to finish out the array setup
1358 posArray = ['LineNo'; posArray(1,1); 'TimeS'; 'DatenumUTC'; posArray(2:end
       ,1)];
1359
1360 % Get varName from structDFL and convert it to a structure
1361 tempData = struct(getfield(redactStructDFL,varName));
1362 % Convert structure to array in an order specified by posArray
1363 for j=1:length(posArray)
1364     newData(:,j) = tempData.(char(posArray(j,1)));
1365 end
1366
1367 % Check for random data spikes
1368 % Remove if present (hardward error/software glitch)
1369 for jj=length(newData):-1:2
1370     if(newData(jj,3)-newData(jj-1,3)<0)
1371         newData(jj-1,:)=[];
1372     end
1373 end
1374
1375 % Create new table using the data and variables created above
1376 newDataTable = array2table(newData,'VariableNames',posArray);
1377 % Add '_table' at the end of the name
1378 varName = strcat(varName,'_table');
1379 % Store data in rawVars structure for internal use
```

```
1380 rawVars{k,1} = varName;
1381 rawVars{k,2} = newData;
1382 rawVars{k,3} = posArray;
1383 % Add fetchable table name to Description
1384 newDataTable.Properties.Description = varName;
1385 % Send table to base workspace for user view
1386 assignin('base',varName,newDataTable);
1387 end
```

VITA

Andrew Cole

Candidate for the Degree of

Master of Science

Thesis: DEVELOPMENT AND EVALUATION OF A 360° DIFFERENTIAL PRESSURE GUST SENSOR FOR EXTREME WEATHER ENVIRONMENTS

Major Field: Mechanical and Aerospace Engineering

Biographical:

Education:

Completed the requirements for the Master of Science in Mechanical and Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in July, 2022.

Completed the requirements for the Bachelor of Science in Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in 2020.

Completed the requirements for the Bachelor of Science in Mechanical Engineering at Oklahoma State University, Stillwater, Oklahoma in 2020.