THIS BLOWS?: EVALUATION OF ADDITIVE MANUFACTURED FIVE HOLE PROBES FOR UNMANNED SYSTEM BASED WIND MEASUREMENTS

By

KYLE HICKMAN

Bachelor of Science in Mechanical Engineering
Oklahoma State University
Stillwater, Oklahoma
2019

Bachelor of Science in Aerospace Engineering
Oklahoma State University
Stillwater, Oklahoma
2019

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, 2021

THIS BLOWS?: EVALUATION OF ADDITIVE MANUFACTURED FIVE HOLE PROBES FOR UNMANNED SYSTEM BASED WIND MEASUREMENTS

Thesis Approved:

Dr. Jamey D. Jacob
____
Thesis Advisor

Dr. He Bai
____

Dr. Imraan Faruque
____

ACKNOWLEDGMENTS

There are many people to acknowledge, I'm sure I will miss a few. To start I want to thank my friends and family for always being supportive, not sure how I lucked into having such quality people around me. My advisor Dr. Jacob for his time and guidance. I appreciated the freedom to explore and make mistakes. Not that I always enjoyed the mistakes part. Literally crashing my thesis into the ground was not so fun. I also want to thank the rest of my committee Drs. Faruque and Bai. I have been able to learn much from both. All my professors at OSU have taught me much. The lessons I have taken from them have been as valuable for my thesis as I'm sure they will be down the road. I also would like to thank the entire staff at USRI. They have always been eager to help answer questions and teach what they know. The opportunities and knowledge I have gained from my experience there have been tremendous. I especially want to thank Levi Ross. His knowledge and willingness to help saved me an untold number of headaches and hours. From MATLAB to sensor DAQs I have learned much from him. Finally, my main research partner for the past two years James Brenner. He has done much on this project from designing and printing hardware to helping running calibrations on the probes. Being able to work with him has been invaluable and made the hours spent in a the wind tunnel a bit better.

This work is supported in part by the National Science Foundation under Grant No. 1539070, *Collaboration Leading Operational UAS Development for Meteorology and Atmospheric Physics*

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

(CLOUD-MAP), and Grant No. 1925147, *NRI: Safe Wind-Aware Navigation for Collaborative Autonomous Aircraft in Low Altitude Airspace*, and by NASA under the University Leadership Initiative. Additional support provided by the OSU Unmanned Systems Research Institute. I appreciate the assistance of many USRI staff and students.

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

Name: KYLE HICKMAN

Date of Degree: JULY, 2021

Title of Study: THIS BLOWS?: EVALUATION OF ADDITIVE MANUFACTURED FIVE HOLE PROBES FOR UNMANNED SYSTEM BASED WIND MEASUREMENTS

Major Field: MECHANICAL AND AEROSPACE ENGINEERING

Abstract: A deep understanding of the wind in the atmosphere is becoming valuable as unmanned aerial systems (UAS) are integrated into the National Airspace System. Unmanned vehicles are also becoming a key tool in weather observation and research. One of the most effective ways to measure wind from a fixed wing aircraft is a multi-hole probe (MHP). Commercial probes can be prohibitively expensive for wide scale use. Custom probes machined from metal can require extensive manufacture time and expensive tools for machining. Additive manufactured probes have been proven to be accurate in a wind tunnel and offer a good option for a cheap and easy to manufacture MHP. Limited flight testing and evaluation of the probes in flight has been done. This thesis investigates the use of a fully additive manufactured probe for UAS based wind measurements. A variety of different evaluation techniques from literature are used and compared. The paper covers the design, calibration, and preliminary validation of the probes. The probe is demonstrated for use in a meteorological application where the probes are used with a full meteorological sensor suite to produce atmospheric boundary layer(ABL) observations.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Figure		Page

Figure Page

CHAPTER I

INTRODUCTION

1.1 Motivation

The use of unmanned aerial systems as meteorological tools has been developing rapidly over the years. Unmanned systems have demonstrated the ability to support meteorological observation and research efforts. These efforts have promise to be able to supplement other atmospheric observation methods such as balloons and mesonets.

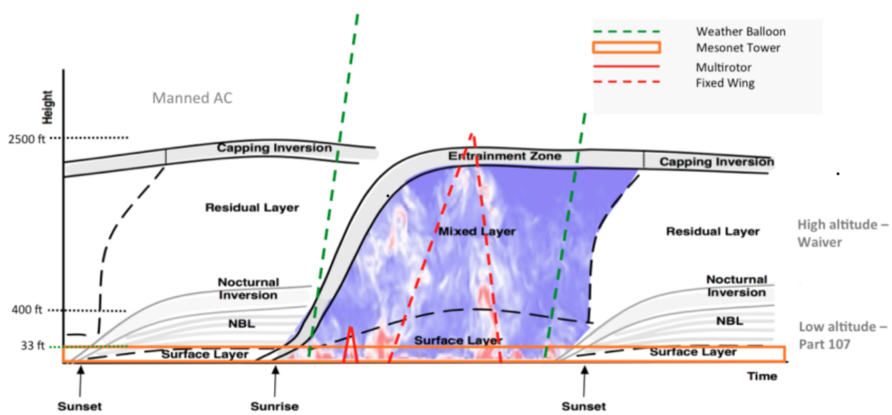


Figure 1: Diurnal Cycle of the ABL with Proposed UAS Operations[23]

Figure 1 shows the Atmospheric Boundary Layer (ABL) over the diurnal cycle. The diurnal cycle is how the boundary layer transition in a 24 hour night/day cycle. Unmanned systems have enabled the study of this cycle with increased detail[15]. This would lead to better data for meteorologist enabling better forecasting and more accurate data reporting.

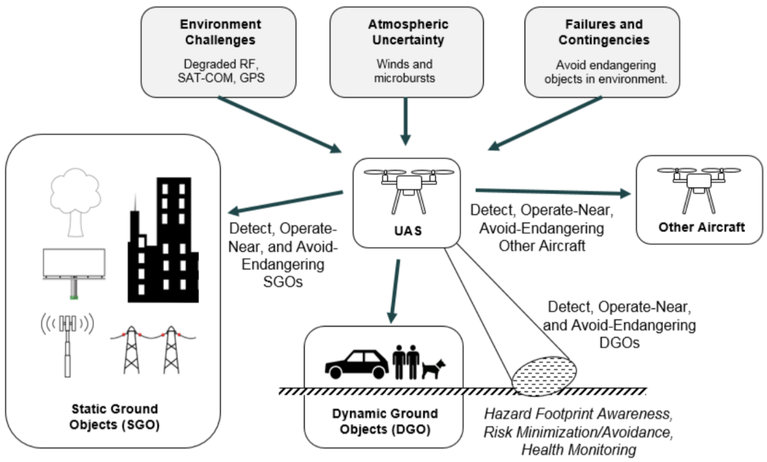


Figure 2: NASA Safe Autonomous Flight Environment SAFE50 Navigational Hazards in Urban Environments [2]

Beyond meteorological research for the integration of UAS into our cities and urban areas wind research is needed[2]. Figure 2 are challenges NASA has identified as barriers to UAS integration into urban areas. One of these areas is near real time robust wind information. Multi-hole probes are not the full solution to this issue. However, development of a cost effective, mass producible probe could enable the data needed to support simulation and modeling that will help solve the problem.

1.2 Background

Multi-hole probes have been used for aerial based atmospheric measurements since the late 1960s[13].They have been used for everything from measurement of wind on a ship, characterization of flows through turbo machinery, and measurement of wind from aircraft. The use of MHPs for atmospheric research from fixed wings has been explored for a long time. The probes used on many of these existing systems are incredibly expensive or difficult to manufacture. A cheap easy to manufacture probe

1.3 Goals and Objectives

The over arching goal of this thesis will be to continue the design, evaluation, and integration of an additive manufactured multi hole probe. With a final objective of demonstrating the use of the probe in a meteorological application. The specific objective are as follows:

- Design 5 hole probe for UAS integration
- Calibrate 5 hole probe evaluate baseline errors
- Integrate a five hole probe to unmanned aircraft
- Implement algorithm to address biases and isolate wind vector from probe measurements
- Calibrate and validate 5 hole probe system by comparison with weather towers and other UAS wind systems
- Compare accuracy and errors to other other wind sensing UAS systems
- Demonstrate the use of the 5 hole probe system in a holistic meteorological application

The design of the probe builds upon previous work done on 3D manufactured probes[14].This worked demonstrated the accuracy and use of additive manufactured in a wind tunnel setting. With the objective of using the probe in a meteorological application meteorological the data quality desired for meteorological observation should be discussed.

Table 1 are the desired accuracies as reported through the CLOUD-MAP project. These are not realistic objectives for this project. This will become apparent as the accuracies of other systems are explored in the next chapter.

Table 1: Desired Sensor Specifications for Meteorological Observation [23]

Meteorological Variables and Accuracy		**Sensor Response Time**	
Temperature	±0.2 °C	Time	<5s (<1s Preferred)
Relative Humidity	±5.0%	**Operational Environmental Conditions**	
Pressure	±1.0hPa	Temperature	-30 to 40 °C
Wind Speed	±0.5m/s	Relative Humidity	0 to 100%
Wind Direction	±5 Degrees Azimuth	Wind Speed	0 to 45 m/s

1.4 Thesis Outline

The discussion of the probe will begin with a review of previous work on multi-hole probes and other systems and sensors that are used for wind measurements. The wind tunnel calibration of the probe will then be discussed in Chapter 3. All ground based instruments will be included here as well. Chapter 4 will involve the basics of wind vector realization from a multi-hole probe. Different flight methods and data processing methods are described. An evaluation of an anemometer equipped quad used for calibration is also found here. The results of comparison flights are found in Chapter 6. The final section will include the conclusion and discussion of results leaving off with future work.

CHAPTER II

LITERATURE REVIEW

2.1 Wind Measurement Instruments

Modern meteorologist have a range of instruments to measure wind speed from the flow. These systems have a wide range of size, accuracy, cost, etc. While many novel wind sensors exist this section of the literature review will be systems that are often mounted to unmanned systems or are relevant validation of wind sensors in presented in this thesis.

2.1.1 Cup Anemometer

Figure 3: Cup Anemometer

Cup anemometers, as pictured above, are one of the most simple and widely used anemometers for basic wind speed. Cup anemometers are widely used because of three main benefits. First, as long as the cups are horizontal the anemometer is omni-directional. Second, modern cup anemometers have linear calibration for wind speed and direction. Cup anemometers are also very robust and easy to operate. The cup anemometer does have a variety of disadvantages that make reduce its effectiveness for some applications. Cup anemometers suffer from a slow step response since it takes time for the wind to impart momentum on the cups. This is more sever for decreases than increases in wind speed. This leads to "Over speeding" or the anemometer measuring something higher than the actual averaged wind speed [25].

2.1.2 Propeller Anemometer

Figure 4: Propeller Anemometer

Propeller anemometers are typically mounted on a wind vane for atmospheric measurements. Although slightly more complex mechanically than the cup anemometer it is still a simple system. The propeller vane anemometer is able to measure the wind angle more simply than a cup anemometer. Due to the interaction of the vane and the propeller the system has a second order response and calibration. The propeller vane does have a few systematic errors like the cup anemometer. It has a similar issue with a slow step response to wind speed, high response systems can minimize this. A direction lag is inherent due to the vane always lagging the wind. The rotation of the vane in the absence of wind can also cause false propeller speed readings[26].

2.1.3 Hot Wire Anemometer

Figure 5: Hot Wire Anemometer

Hot wire anemometers have two different methods of measuring flow speed. Constant temperature hot wire anemometer and constant current hot wire anemometer. Both types of hot wire work by relating the heat transfer from a wire to the velocity of a flow field. In the case of a constant temperature the relation is the current required to pass through a wire to maintain temperature for a flow velocity. The constant current hot wire does the inverse, keeping the current constant and relating the temperature change to flow velocity. Hot wires if calibrated properly are able to measure difficult to observe phenomena such as turbulence. This is because hot wires are very sensitive to flow changes. This high sensitivity makes measurements from hot wires very susceptible to error from mounting and alignment with the flow[18].

2.1.4 Ultrasonic Anemometers

Figure 6: Young 81000 3D Ultrasonic Anemometer

Ultrasonic anemometers come in both 2D and 3D versions. The anemometer operate using an inverse time transit difference[12]. This is done by generating high frequency acoustic pulses. These pulses are then measured upstream and downstream of the source. The baseline equation for this is

$$\frac{1}{t_{12}} - \frac{1}{t_{21}} = \frac{(c_0 + v\cos\theta)}{L} - \frac{c_0 - v\cos\theta}{L} \tag{2.1.1}$$

where c_0 is the speed of sound in air, $t_1 2$ and $t_2 1$ are the times it take to traverse between sensors, v is the wind speed, L is the distance between sensors. Ultrasonic anemometers are beneficial because they have no moving parts and can be compact in form. However, they are one of the more expensive anemometer types.

2.1.5 Pitot Tube

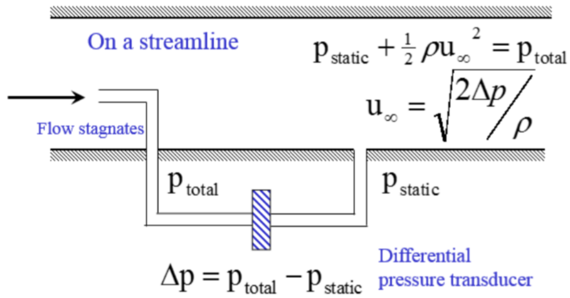


Figure 7: Pitot Probe Layout

Pitot tube anemometers calculate airspeed based on the difference between stagnation pressure (P_{total}) and static pressures (P_{static}). Figure 7 is an illustration of how a pitot probe works. The relationship between these pressures can be equated to airspeed using Bernoulli's equation:

$$p_{static} = p_{total} + \frac{\rho u_\infty^2}{2} + \rho g h \tag{2.1.2}$$

Equation 2.1.2 is the simplified version of the Bernoulli equation. The $\rho g h$ term in the Bernoulli is a potential pressure term, where ρ is density, g is gravity, and h is a vertical height. Since the pitot probe does not have a large vertical distance between measurements this term can be ignored. u_∞ is the free stream airflow speed. Rearranging Equation 2.1.2 to isolate the free stream velocity and removing the potential term results in the following:

$$u_\infty = \sqrt{\frac{2\Delta p}{\rho}} \tag{2.1.3}$$

Equation 2.1.3 is used to calculate velocity from a pitot probe. The Δp term is the difference between the stagnation pressure and the total pressure ($p_{static} - p_{total}$). These calculation carry over for multi-hole probes where air speed calculations follow Bernoulli equation.

2.1.6 5 Hole Probes

The 5 hole probe (5HP) was developed by Admiral Taylor in 1915 for the measurement of 3D wind from ships[37]. Multi-hole probes are similar to pitot probes in that they use a static and total pressure to calculate the speed of the flow. Multi-hole probe are able to combine those measurements with extra ports in perpendicular planes to the static port. For a 5 hole probe the layout of ports is shown in Figure 8.

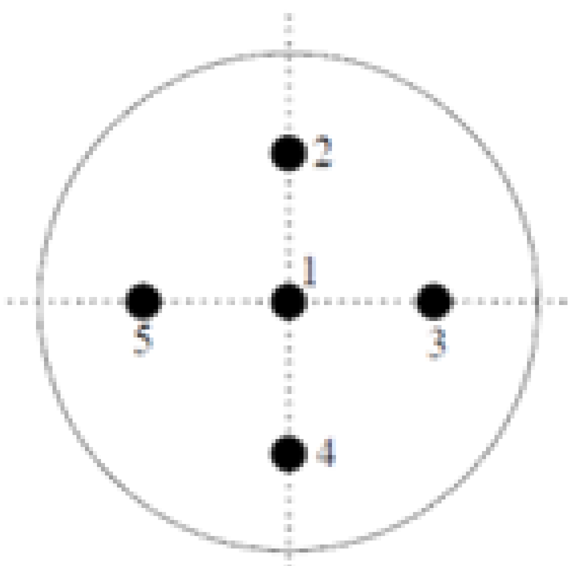


Figure 8: Pitot Probe Layout

The tip geometries of the probe largely determines its operating characteristics. The most common tip geometries used are hemispherical, conical, and pyramid, as shown in Figure 2 from [19]. Numerous studies have demonstrated that differences in conical and pyramid tips is primarily based on performance of how flow separation behaves around the probe. One study shows that with smooth surfaces (see hemispherical or conical) separation occurs more slowly [36]. This is generally a good trait to have for MHPs, but this also comes at a cost of being sensitive to changes in Reynolds Number due to abrupt changes in the free-stream-velocity. From [35] most MHPs usually have an angular operating range of -25 degrees to 25 degrees. From previous literature studies, for the flow regions that a small UAV will be experiencing a hemispherical tip head has been shown to be the best geometry [39].

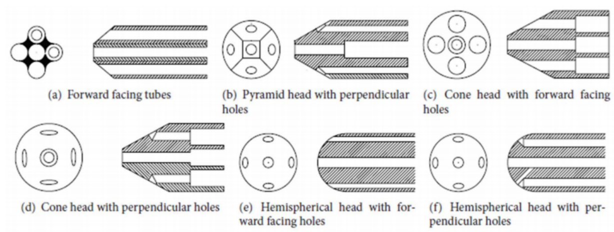


Figure 9: 5HP Tip Geometry Types[19]

Due to imperfections in manufacturing multi-hole probe must be extensively calibrated. There are two methods used to calibrate multi-hole probes nulling and non-nulling mode[37]. In a nulling calibration method, a probe is inserted into a flow and rotated until the error signal across two opposing ports (e.g alpha low and alpha high) is nulled[31]. Then, by taking the inclination angle at that position, it is related to the flow direction, and by measuring the pressure at that position, the velocity is obtained. This method is highly accurate, but it is a very involved process that requires large amounts of space for traversing and very long data acquisition sessions. The non-nulling method calibration, while tedious, is easier to perform in a standard wind-tunnel test section. This method involves having the probe fixed in the center of the wind-tunnel and slowly rotated from one of its angular operating limits to the other, while each pressure reading of the probe is measured and correlated to the angular position of the probe. This data is then compared to a Pitot-static probe upstream in the wind-tunnel, which measures the free-stream tunnel velocity. This method was first introduced by Treastor and Yocum[37].

2.2 Systems for Atmospheric Observation

The following section will cover different systems used for atmospheric observations. Rather than just wind these systems take a multitude of atmospheric measurements including wind.

2.2.1 Weather Towers

Weather towers are a popular method for continuous weather monitoring.The Oklahoma Mesonet is a system of 121 weather towers scattered across Oklahoma, at least one for each county. These towers are outfitted a wide range of sensors.

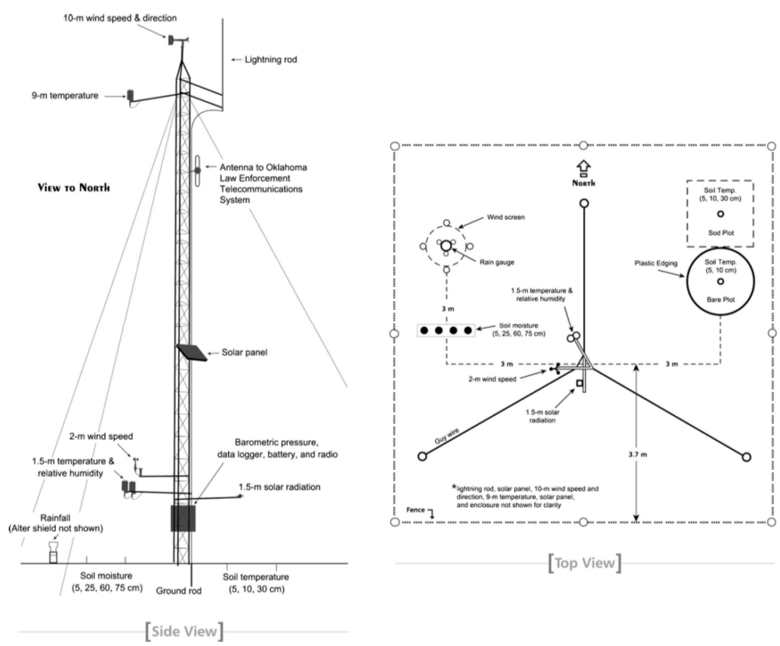


Figure 10: Oklahoma Mesonet Tower Schematic[29]

The schematic above shows the general layout of an Oklahoma Mesonet station. The Oklahoma Mesonet reports data every five minutes. For the 10m wind speed a R.M. Young 5103 anemometer is equipped at 2m a R.M. Young 3101 is equipped. Along with wind speed and direction the Mesonet sites take temperature, pressure, humidity, rainfall, solar radiation, and a multitude of soil parameters[29].

2.2.2 Weather Balloons

Figure 11: Weather Balloon Launch[28]

Weather balloons have been used since the 1930s for weather observations[38]. Weather balloons are filled with hydrogen and can take data up to 115,000ft. Radiosondes are attached to the weather balloons for pressure, temperature, and humidity readings. Rawinsondes are GPS enabled or tracked by radar, this allows for the wind speed to be estimated from the track of the balloons[8]. One issue with weather balloons is cost since many radiosondes are not recovered. Balloons are used to create soundings.

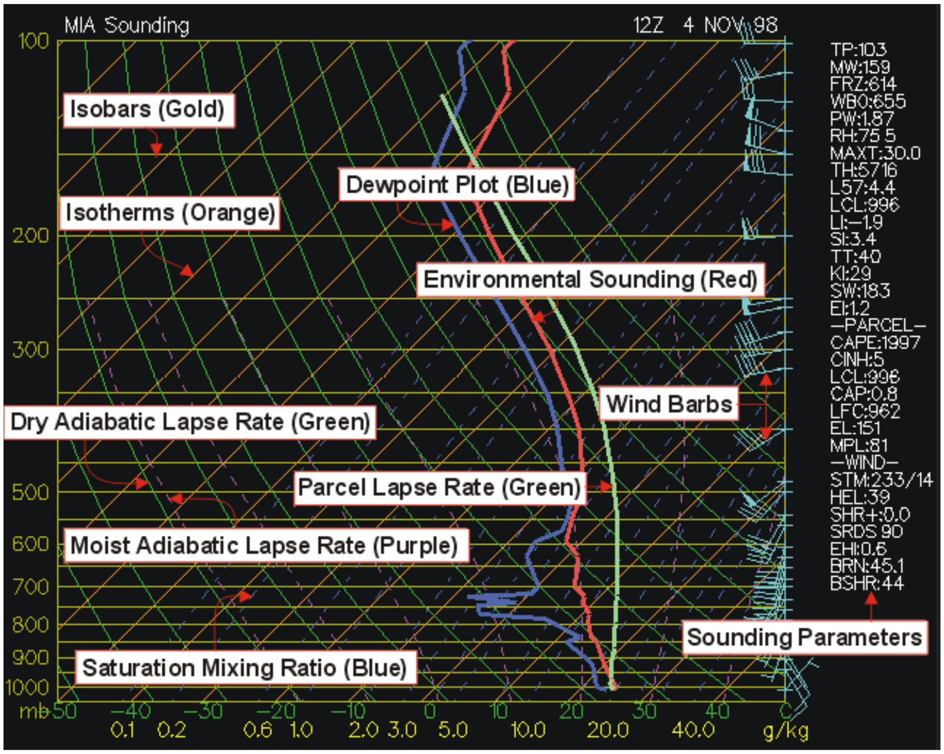


Figure 12: Skew-T Weather Sounding [10]

Figure 12 is an example of how the data from a weather balloon is presented. The figure is referred to as an atmospheric sounding. From the sounding important atmospheric measures can be derived.

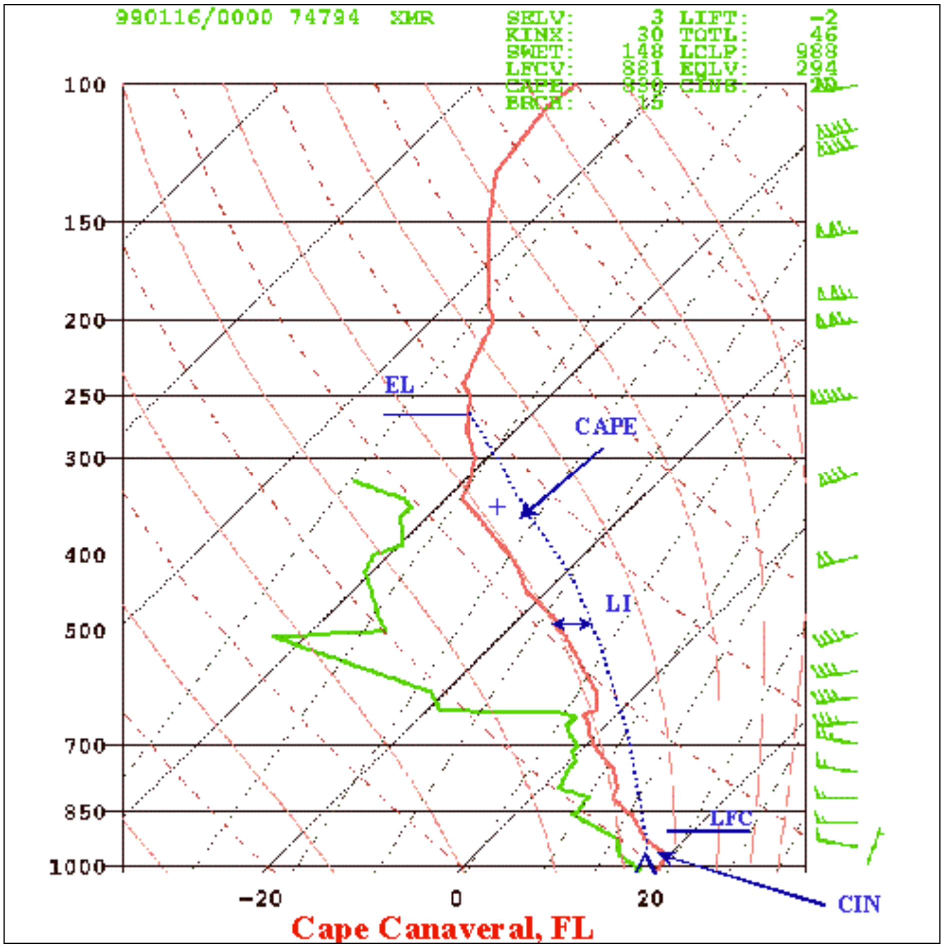


Figure 13: Skew-T Weather Sounding with Different Derived Parameters [10]

Figure 13 shows important parameters that can be derived from the Skew-T sounding. Theses parameters are used in forecasting. Going from bottom to top on the figure with CIN or the Convective Inhibition shows how stable the atmosphere is. The Level of Free Convection (LFC) this is a boundary where above particles will begin to accelerate upwards. The Lifted Index the temperature difference between the parcel and environmental temperature at 500mb. The Convective Available Potential Energy (CAPE) the positive area between the red line and the dotted blue line (parcel and environmental temperature) this indicates the instability in the atmosphere, correlating to likelihood and severity of storms. The Equilibrium Level (EL) is the pressure value at the top of the CAPE. The Lifted Condensation Level (LCL) is near where the CIN is labeled on this figure, it signifies where clouds will form (the peak of the blue triangle on the x axis at 20).

2.2.3 Manned Aircraft

The implementation of Inertial navigation systems (INS) on aircraft allowed for accounting for the motion of aircraft in atmospheric measurements. This implementation began being taken advantage of in the late 1960s. One of the first major implementation was sounding flight flown with a Canberra PR 3 aircraft done by Axford in 1968[13]. The aircraft was outfit with a multi-hole probe and gimbal based INS system. The INS system is the key to deriving wind component from aircraft based multi hole probe measurements. The INS system allows for the removal of aircraft dynamics from the measurement as will be described in the Multi-hole probe algorithm section. While the flight campaigns were successful the lack of a data acquisition system made the processing of the data prohibitively time consuming.

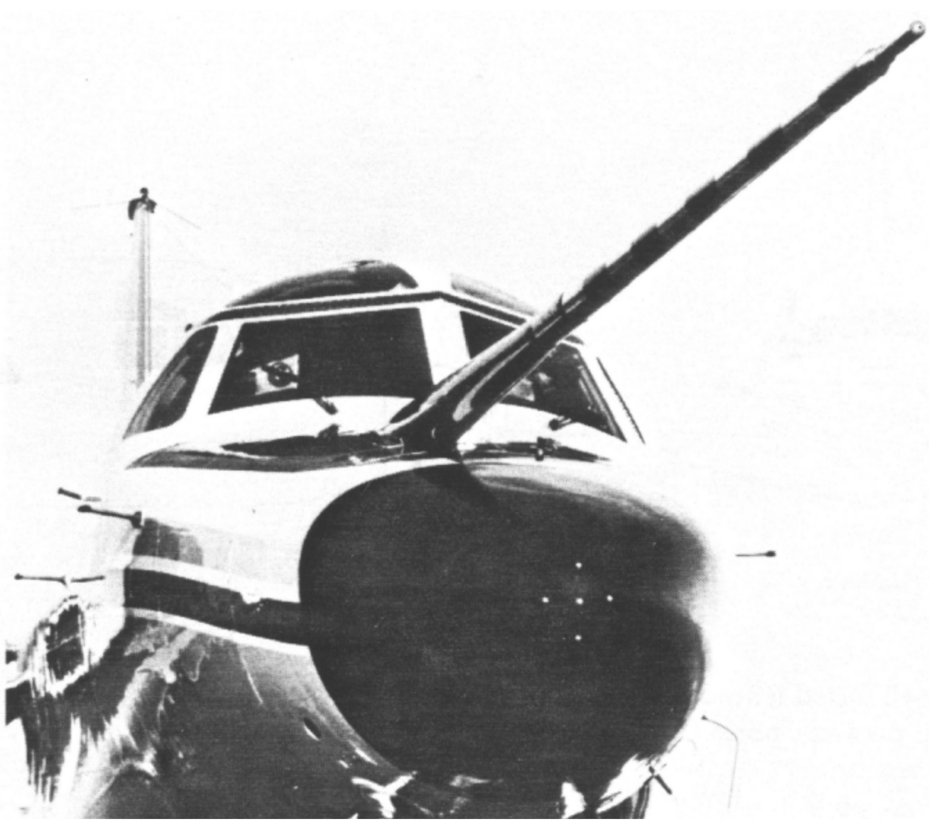


Figure 14: NCAR C130Q Hercules Equipped for Horizontal Wind Measurements[27]

A flight campaign performed by NCAR has been able to use the methods developed by Axford[13] with modern INS and data acquisition. The program uses C130Q Hercules with a 5 hole probe system as pictured in Figure 14. The studies done with this aircraft establish calibration and error investigation procedures. The limitations found of the system are a frequency response of 10Hz and a short term wind accuracy of ± 1m/s[27]

2.2.4 Unmanned Aircraft Weather and Wind Measurements

The use of Unmanned aircraft for weather observation was first explored in 1970 by Konrad et al [24]. The flight tests conducted demonstrated the feasibility of using unmanned systems for atmospheric research. By today's standards the system used was rudimentary having no autopilot and limited endurance.

(a) Atmospheric RC Aircraft in Flight

(b) Aircraft Spotting System

Figure 15: Early RC Atmospheric Flights [24]

The aircraft used in the Konrad study had sensors to measure temperature, pressure, and, humidity. The aircraft was also equipped with a pitot probe and thermistor mass flow sensor to track the aircraft's airspeed. Wind speed was not an objective of the flight campaign.

2.3 Algorithms For Fixed Wing Wind Estimation

One of the major challenges of getting accurate wind measurements from unmanned aircraft is removing the aircraft state from the raw measurements. For discussion of these algorithms a definition of aircraft frames is required.

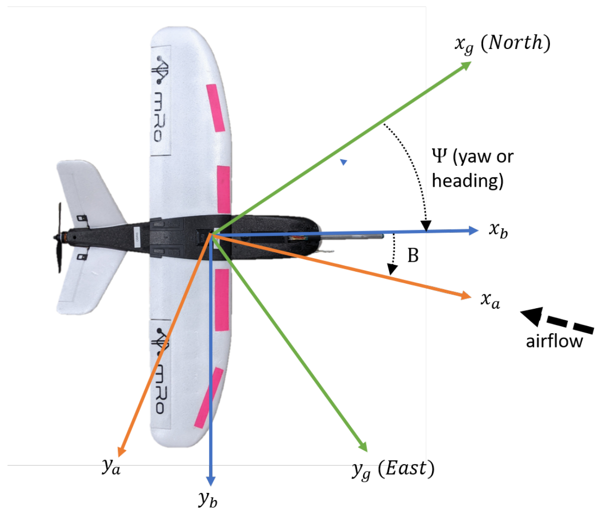


Figure 16: Aircraft Frames Visualization

Figure 16 is a visualization of the different frames of reference that will be needed for discussion in this paper. In green, x_g and y_g are ground or Earth based coordinates. These relate to North and East on a map respectively and are independent of the aircraft. In blue, x_b and y_b are aircraft or body based coordinates. For these coordinates x_b relates to the heading of the aircraft and is positive out of the nose, y_b is positive out of the starboard(right) wing these rotate with the aircraft as it maneuvers. ψ is the angle that x_b is offset from North. In orange, x_a and y_a are air or wind frame coordinates. x_a points in the direction that the airflow over the aircraft is coming from. This airflow is a function of the wind and the aircraft ground speed. β is the angle offset of the airflow from x_b. For all discussion of coordinate frames in this paper the subscript g will refer to earth frame, subscript b will refer to body frame, and subscript a will refer to the wind frame.

2.3.1 Instrument Free Wind Estimation

One method of instrument free wind estimation is developed by Mayer et al.[28]. The algorithm uses the assumption that an aircraft with a constant throttle and pitch should maintain a constant airspeed.

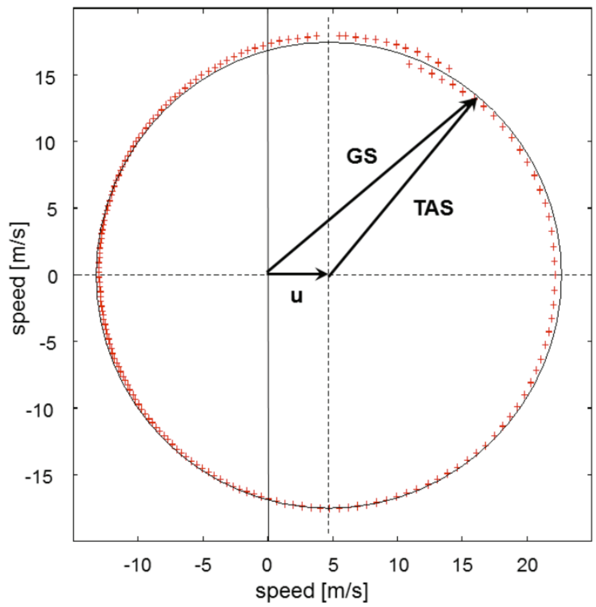


Figure 17: Visualization of algorithm wind with derived wind triangle[28]

Figure 17 illustrates the wind triangle the algorithm is based around. In the wind triangle GS is the ground speed of the aircraft, TAS is the true airspeed of the aircraft, and u is a wind vector. For an aircraft without an airspeed sensor only ground speed is know from GPS data. Based off the wind triangle:

$$TAS + u = GS \tag{2.3.1}$$

True air speed(TAS) is calculated as

$$\overline{TAS} = \frac{1}{N}\sum_{i}^{N} \|GS_i - u\| \tag{2.3.2}$$

With TAS and GS variance is calculated

$$\sigma_{TAS}^2 = \frac{1}{N}\sum_{i}^{N}(\|GS_i - u\| - \overline{TAS})^2 \tag{2.3.3}$$

Using this algorithm the variance is minimized to find the wind speed(u).

This method does have significant drawbacks. One of the major issues is that the algorithm breaks if the aircraft does not maintain the assumed constant throttle and pitch[28]. While the algorithm can be used for any constant flight profile (orbits, racetracks, etc.) it does require two full cycles to average over to be accurate. The method also contains large uncertainties[33].

2.3.2 Multi-Hole Probe Based wind Estimation

Multi hole probes allow for algorithms to be simplified and reduced to mostly frame rotations. This is because the β and α angles can be known from the 5HP data reducing the need for extra calculations to estimate β or α. A few different multi hole probe algorithms have been used. The following is a reduced version of the base algorithm used in a variety of multi hole probe wind studies[13][27][33]. With many other study using a augmented versions of the algorithm[30][11]. Referring back to the reference frames established in Figure 16 the wind vector in ground frame can be described as

$$\overrightarrow{w}_g = \overrightarrow{v}_g + \mathbf{T}_{gb}\mathbf{T}_{ba}\overrightarrow{u}_a \tag{2.3.4}$$

In Equation 2.3.4 $\overrightarrow{v}_g$ is the ground speed of the aircraft. $\mathbf{T}_{gb}$ and $\mathbf{T}_{ba}$ are rotation matrices operating on $\overrightarrow{u}_a$ the total airspeed of the vehicle. For this equation $\overrightarrow{u}_a$ is being rotated from wind to body ($\mathbf{T}_{ba}$) then from body to earth ($\mathbf{T}_{gb}$). This allows the merger with the aircraft ground speed and results in a wind speed from the ground reference which is desired for meteorological wind analysis.

The algorithm calculates true airspeed $\overrightarrow{u}_a$ from the norm, $|\overrightarrow{u}_a|$. Where,

$$|\overrightarrow{u}_a|^2 = 2c_pT_{tot}[1 - (\frac{p}{p+q})^{\kappa}] \tag{2.3.5}$$

κ, the poisons ratio, is calculated as $\kappa = Rc_p^{-1}$ where for standard dry air $R = 287Jkg^{-1}K^{-1}$ and $c_p = 1004Jkg^{-1}K^{-1}$. From this point the u_a is rotate from wind frame to body frame using $\mathbf{T}_{ba}$ based on *alpha* and *beta* as follows.

$$\overrightarrow{u}_b = -\frac{|\overrightarrow{u}_a|}{\sqrt{1 + tan^2\alpha + tan^2\beta}}\begin{pmatrix} 1 \\ tan\beta \\ tan\alpha \end{pmatrix} \tag{2.3.6}$$

Once rotated into body frame next is earth frame. $\mathbf{T}_{gb}$ is done with three turns about each of the aircraft's axis $T_1(\phi)$, $\mathbf{T}_2(\theta)$ and $\mathbf{T}_3(\psi)$ roll pitch and yaw respectively. So,

$$\mathbf{T}_{gb} = \mathbf{T}_1(\psi)\mathbf{T}_2(\theta)\mathbf{T}_3(\phi)$$
$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{pmatrix} \begin{pmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{pmatrix} \begin{pmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (2.3.7)$$

The equations can then be combined to form the following where $D = \sqrt{1 + tan^2\beta + tan^2\alpha}$. For the sake of space in the following equation c is cosine, s is sine and , t is tangent.

$$\vec{w}_g = \vec{v}_g - \frac{|\vec{u}_a|}{D}$$
$$\begin{pmatrix} c\psi c\theta + t\alpha(s\phi s\psi + c\phi c\psi s\theta) + t\beta(c\psi s\phi s\theta - c\phi c\psi) \\ c\theta s\psi + t\alpha(c\phi s\psi s\theta - c\psi s\phi) + t\beta(c\phi c\psi + s\phi s\psi s\theta) \\ -s\theta + c\phi c\theta t\alpha + c\theta s\psi t\beta \end{pmatrix} \quad (2.3.8)$$

Equation 2.3.8 could be considered the baseline 5 hole probe algorithm. The complete version was originally developed for large manned aircraft by Axford[13]. The major difference between the presentation of the equation by Rautenberg for UAS and the full form is the removal of the angular velocities term. The addition of the angular rates term to Equation 2.3.4

$$\overrightarrow{w}_g = \overrightarrow{v}_g + \mathbf{T}_{gb}([\mathbf{T}_{ba}\overrightarrow{u}_a] + [\dot{\Omega} \times \overrightarrow{r}]_b) \tag{2.3.9}$$

In Equation 2.3.9 $\dot{\Omega}$ is the angular rates of the aircraft ($\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$). The term $\overrightarrow{r}$ relates to the distance of the probe from the IMU, which should be near the center of gravity of the aircraft. The angular rates are multiplied by the probe arm to get angular velocities. For UAS much discussion is had over whether this term is necessary since the distance from the center of gravity to the probe tip is small in UAS [33][30]. Nichols et al. also notes a v_{flex} term that accounts for unaccounted flexing from the aircraft. While noted this term is not able to be measured so this term is left out of calculations[30].

2.3.3 Unmanned Aircraft Atmospheric Measurements

Wind measurement techniques have been tested for both fixed wing platform and multi-rotors. Flight campaigns such as the Lower Atmospheric Process Studies at Elevation - a Remotely piloted Aircraft Team Experiment (LAPSE-RATE)[15] and the Collaboration Leading Operational UAS Development for Meteorology and Atmospheric Physics (CLOUDMAP)[23] have demonstrated the potential for UAS based meteorological efforts.

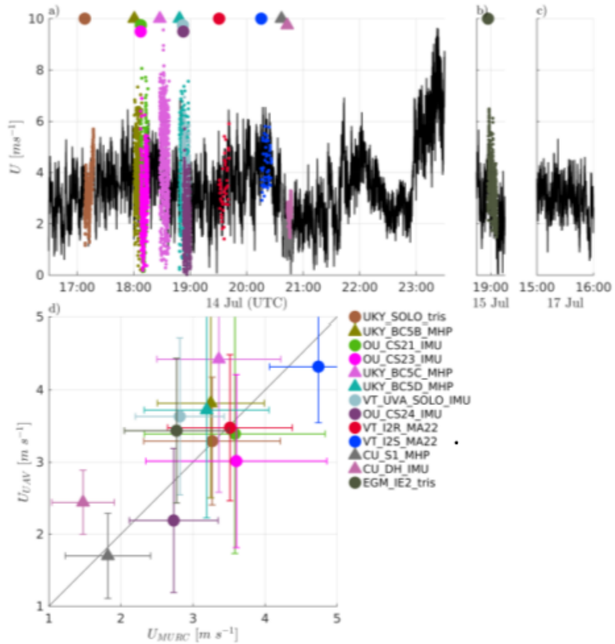


Figure 18: Comparison of LAPSE-RATE Wind Measurements[15]

The LAPSE-RATE flight campaign was a multi-university co-operative flight campaign that took place in July of 2018. A total of 8 universities and 2 private companies. 13 different aircraft were equipped to measure wind speed. Figure 18 is a comparison between the wind speed measurements of all 13 of these vehicles and a ground sensor station the Mobile UAS Research Collaboratory (MURC). Figure 18 shows a comparison of all the wind measurements of the aircraft flown. Overall all the mean value differences between all the aircraft wind speeds was $.22m/s \pm .59$[15]. One of the major issues noted with comparing these systems is managing different data operations between the teams. For the comparisons no control was placed on the post processing of the data.

The post processing method implement by the University of Kentucky is a good example of how much post processing can change the accuracy of the probe[11]. To account for biases in probe measurement a optimization process is implemented on an objective function(δ). The process is applied to a section of the flight where the aircraft does not have significant acceleration or deceleration. Orbits after take off are recommended. The objective function is as follows

$$\delta_U = \langle U \rangle_{U_{ac}>0} - \langle U \rangle_{U_{ac}<0} \tag{2.3.10}$$

$$\delta_V = \langle V \rangle_{V_{ac}>0} - \langle V \rangle_{V_{ac}<0} \tag{2.3.11}$$

$$\delta = \delta_U^2 + \delta_v^2 \tag{2.3.12}$$

In equation 2.3.10 and 2.3.11 $\langle\rangle$ signifies an average over the total data set. The equations are taking the the difference in velocity over positive and negative aircraft inertial speeds, U_{ac}. Then equation 2.3.12 is minimized for each source of bias. The sources of bias that are assumed are as follows

$$\theta(t) = \theta_m(t) + \Delta\theta \tag{2.3.13}$$

$$\phi(t) = \phi_m(t) + \Delta\phi \tag{2.3.14}$$

$$\psi(t) = \psi_m(t) + \Delta\psi \tag{2.3.15}$$

$$Q(t) = \zeta Q(t) \tag{2.3.16}$$

$$U_m(t) = U_m(t_m + \Delta t) \tag{2.3.17}$$

These biases account for heading errors: pitch(θ), roll(ϕ), and yaw(ψ). Dynamic pressure error Q. The final error is a time shift error. The source for this is a time shift between the sensor reading and the INS readings.

Table 2: University of Kentucky LAPSE-RATE Correction Factors [11]

sUAS	Flight	$\Delta\theta$	$\Delta\phi$	$\Delta\gamma$	ζ	Δt
1	08:30 MDT	−4.9°	0.17°	2.6°	1.1	0.98 s
1	09:30 MDT	−5.8°	2.7°	2.1°	1.1	0.01 s
2	09:10 MDT	−3.9°	0.85°	1.4°	1.15	2.9 s
2	10:10 MDT	−3.9°	0.78°	1.3°	1.15	2.6 s

Applying these calculations led to the correction factors shown in Table 2. The corrections were most significant in the pitch axis. The corrections on heading and dynamic pressure are large enough to be significant in accuracy of the the probe.

Table 3: UAS MHP System Accuracies

Vehicle	**Probe Type**	**Speed Accuracy (m/s)**	**Accuracy Method**
M2AV[1]	5HP	0.5	1σ
Blackswift S0[9]	5HP	0.48	N/A
ALADINA[16]	5HP	0.5	Gaussian error
wind RPA [17]	5HP	1.1	Gaussian error
Manta[34]	9HP	0.045	1σ
Scan Eagle [34]	5HP	0.045	1σ

Table 3 is a comparison of a few different MHP equipped UAS aircraft. Some aircraft have different uncertainties for horizontal and vertical wind, the table lists only horizontal uncertainties. The Manta and the Scan Eagle both have significantly better accuracies due to the higher quality autopilots that are used to resolve the aircraft state. The Manta is designed by BAE systems and the Scan Eagle is designed by BOEING. Both autopilots have extremely low uncertainties. For example the Blackswift S0 maintains a 0.02m/s accelerometer uncertainty and a 1° attitude uncertainty[9]. The Manta and Scan Eagle both have accelerometer accuracies 0.01 m/s and attitude uncertainties ranging from 0.005 to 0.011 degrees[34].

CHAPTER III

FIVE HOLE PROBE DESIGN AND GROUND CALIBRATION AND TESTING

3.1 Five Hole Probe Development

Multi hole probes make an ideal instrument for wind speed measurements from fixed wing aircraft. Multi hole probes can provide an accurate 3D wind measurement given proper calibration and integration. The details and methodology of the probe design, development, and wind tunnel calibration is presented in chapter three. Details on probe integration to the vehicle and validation of vehicle system measurement is discussed in chapter 4.

Development - 3D Printed Probe

As with regular manufactured MHPs, with 3D manufactured probes there are three main components of consideration for designing the probe: size, shape, and length. Taking the knowledge from studies over 5HP's, the design for the 3D-printed 5HP has a maximum outer 0.5inch diameter and hemispherical head[14]. The maximum diameter was chosen because the diameter provided enough room for all five of the internal hole lines to be printed with the available resolution of the SLA FormLabs printer while still small enough to fit inside a small UAV and provide the performance needed based off of previous studies [14]. The inner hole diameter for the 5HP was chosen to be 0.06 inches. This was verified to be the smallest hole diameter the SLA printer could reliably print while maintaining the integrity of the hole and keeping clear of resin debris. This diameter was also the same diameter as the pressure transducer ports on the sensor board. Each of the five holes were evenly spaced apart from each other across the 0.36 in diameters of the probe body.The static line was placed at 1.5 inches away from the tip of the probe.The static ports themselves consist of a static ring of 0.2 inches in diameter. This ring is connected by 4 perpendicular holes that extend to the outside of the probe. Then a 0.06 inch hole is connected to the static ring and runs down to the end of the probe at 2 degree offset to provide additional room at the end probe for mounting. The total length of the probe is seven inches ending in a chamfered end. The length of the probe was chosen such that it provided enough room to be safely mounted to a fixed-wing UAV or wind tunnel for testing. As the intention for this probe was to be fully 3D-printed, the pressure lines had to be able to connect to the probe itself. At the end of the probe six ports extends 0.15 inches out from the chamfered end. The end was chamfered to put each of the connecting ports at different distance to allow for easier accessibility when connecting the Tygon pressure lines to pressure ports. The probe's isometric dimensions are shown in Figure 3.

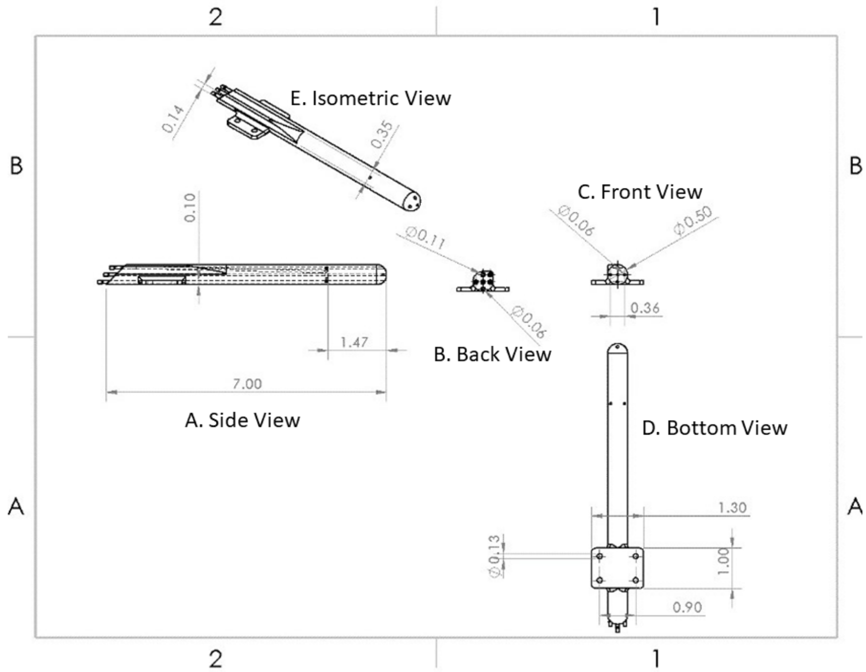


Figure 19: Probe Dimensions: A. Side View, B. Back View, C. Front View, D. Bottom View, E. Isometric View

3D printing Process

The 3D prints of the probes were prone to failure. This was typically caused by clogged ports. Through much trial and error a semi reliable printing process was found. One of the major trade-offs found was between print time and print quality.

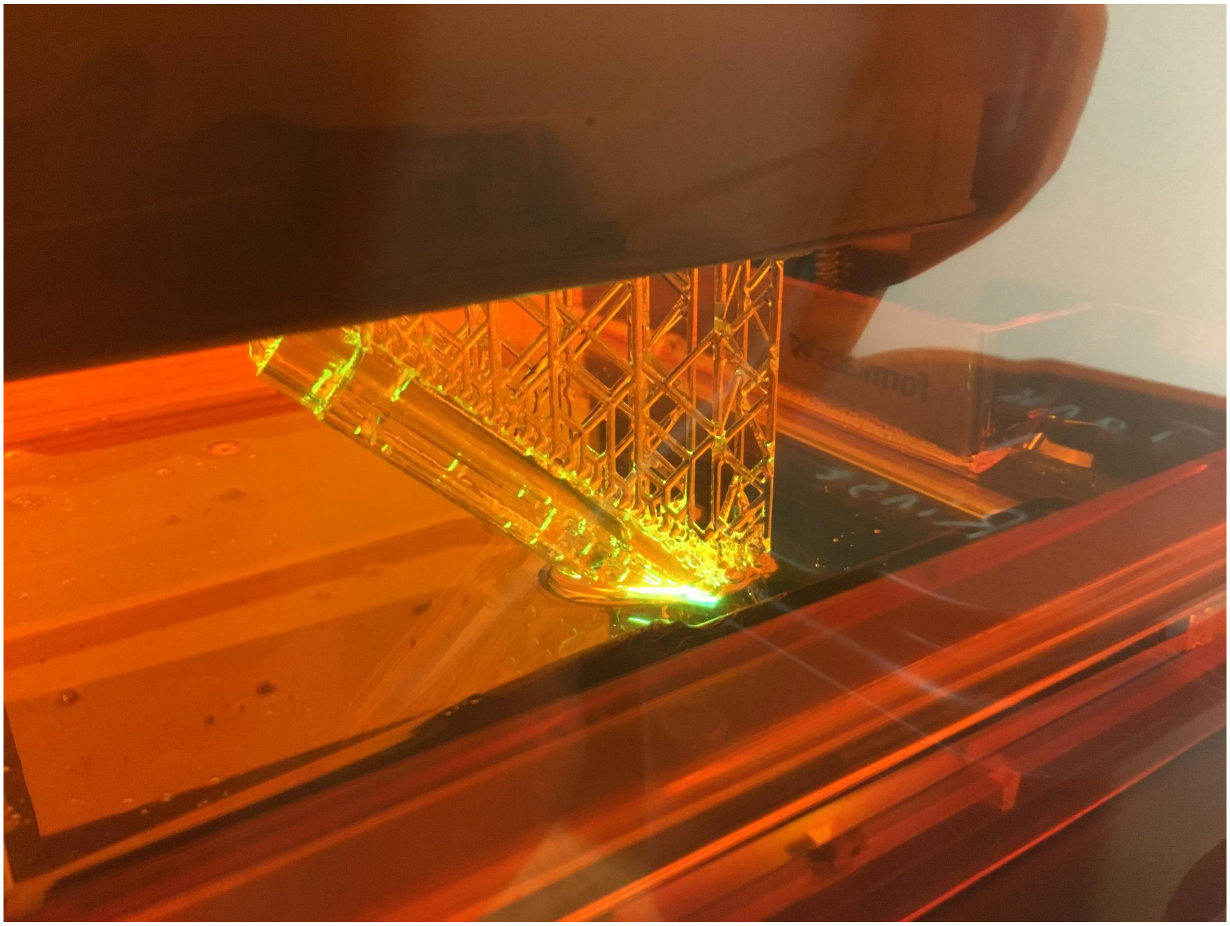


Figure 20: Probe in Mid Print on Formlabs Printer

Figure 20 is a photo of on one of the probes nearing the end of the print. This probe is at around a 45 degree angle to the bed. This was found to be a good angle to print the probe at. Any lesser angle, although it printed faster, resulted in a clogged probe. It was found that for best print results the probe needs to be cleared with alcohol, preferably within an hour of print finish. This prevents the resin from air drying.

Figure 21: Probe Tubes Being Cleared with Alcohol

Figure 21 shows how the probe is cleaned post print. A small bottle of isopropol alcohol is squirted into each of the holes. This is cycled with blowing with an air hose until the alcohol goes through and there is no resin clogging the probe. After this process the probe is sent through the Form Wash and Cure. This process submerges the probe in an isopropol alcohol solution that is then agitated to clean the probe further. After that the probe is inserted into a cure chamber where the temperature is raised to 80C and exposed to LED light. This cures the and hardens the resign improving strength and quality.

3.1.1 Development - Hybrid Probe

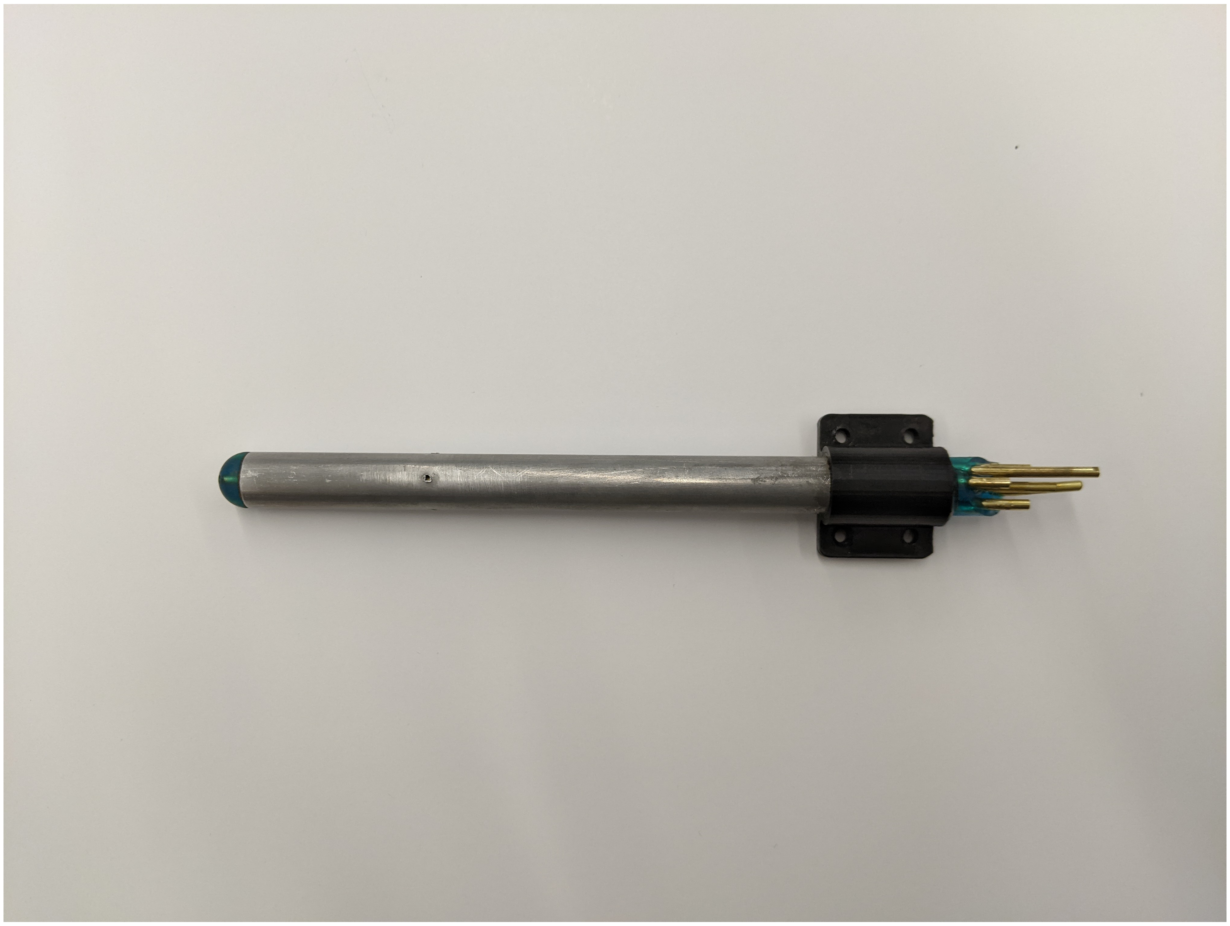


Figure 22: Metal Probe

While the work of Solmoz proved the feasibility of a 3D printed probe[14]. As previously discussed, when the 3D printed probe was initially lengthened finding proper print procedures keep the holes clear enough for the probe to function was a challenge. While a printing process was eventually established that led to successful probes, two different "hybrid" probes were made. The idea was developed from the RPP probe designed by RMIT University. This probe was designed with only a 3D printed tip, and end cap this was to be able to change the tip if the probe took damage during flight operations[32]. With this as a driver a probe was made using the same dimensions as the fully 3D printed, however only the tip was 3D printed. The body of the probe was a quarter inch aluminum tube with a .36 inch inner diameter. 4 holes were drilled at right angles in the tube for static ports. The tip has 5 .10 inch holes for .06 inch inner diameter brass tubes. These act as the ports for the 5HP.

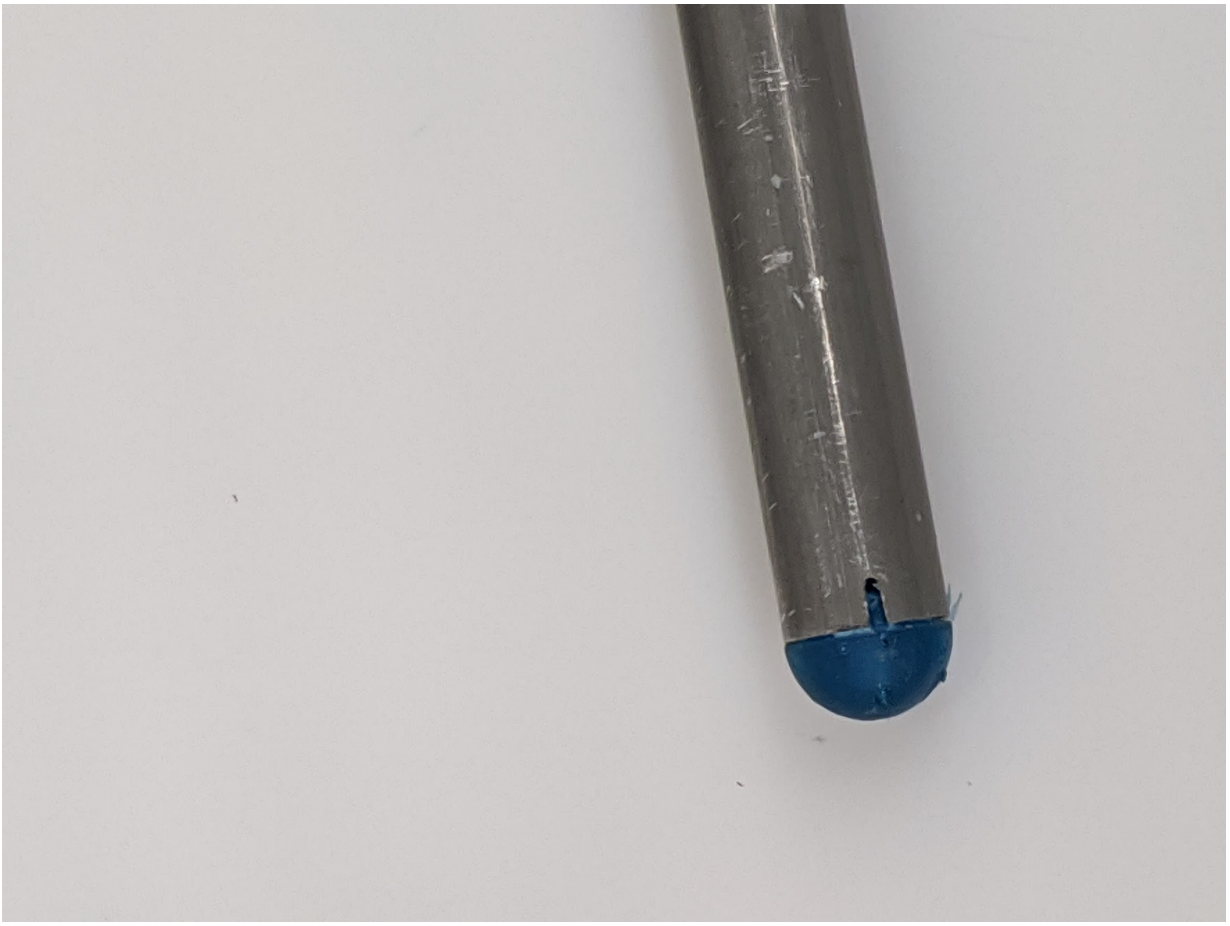


Figure 23: Metal Probe Tip

Seen in Figure 23 a notch was placed on the 3D printed tip and the aluminum tube. This prevented the tip from rotating. On the back end of the tube there are 6 tube pass through. One for each of the 5 holes and one for the static pressure. These probes have similar performance as the 3D manufactured probes, operationally these probes were found to be more reliable as will be discussed in the flight testing section.

Calibration

Because of the inherent flaws in manufacture of MHPs. To measure 3-dimensional flow, each probe requires its own calibration[22]. This is compounded by the fact that these probes are 3d-printed and will have slightly different flow characteristics even when printed on the same 3d-printer. The goal is to determine experimentally the pressure data sets that show probe's behavior in a known flow field. Using the total and static pressures, pitch and yaw angles, and the velocity magnitudes, the data can be expressed as non-dimensional pressure coefficient values. This known flow field is taken over a range of known Reynolds numbers, Mach numbers, and velocities as the probes are traversed across a range of angles incrementally.

Due to the set up of the lab experiment available, the non-nulling method was used to calibrate all of the 3d-printed probes for this research. The wind-tunnel used for non-nulling calibration of the 3D manufactured probes in this experiment is a Flotech 1440 produced by GDJ Inc. with a 12X12X36 inch test section as shown in Figure 4. The wind-tunnel is equipped with a 2hp motor and capable of producing flow speeds from 0 – 25 m/s. The test is best suited for low Reynolds number testing and is controlled by an analog wheel that changes the RPM of the motor. The upstream Pitot-static probe that is connected to an Omega Analog sensor that outputs the pressure and velocity data through a Labview DAQ. The stepper motor used to sweep the probe back forth is a DMX-UMD-23 stepper motor produced by ARCUS Technology. The sensor that is being used to record the pressure from the MHP is a custom made sensor board that is using a Teensy 3.6 and three Honeywell SSCDRRN001PD2A5 digital pressure transducers seen in Figure 5. This sensor package allows for a data capture of 200 hertz. Both pitot-static probe and the MHP were connecting to their respective sensors via Tygon tubing. The sensor system will be discussed further in Chapter 3 where the DAQ can be discussed on an aircraft level.

Figure 24: Wind Tunnel Arrangement

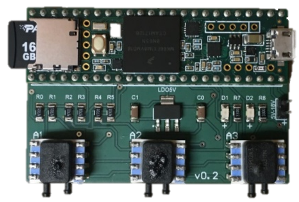


Figure 25: Digital Sensor Package

The procedure for the experiment started by mounting the MHP to the stepper mount via a steel rod and placing it in the center of the wind tunnel at 0 degrees as shown Figure 6. To keep the tip of the probe away from the boundary layer effects from the edges of the test chamber, the probe will pivot around its tip instead of its end. This was done by 3D-printing an extension mount that connects the probe to the steel rod.

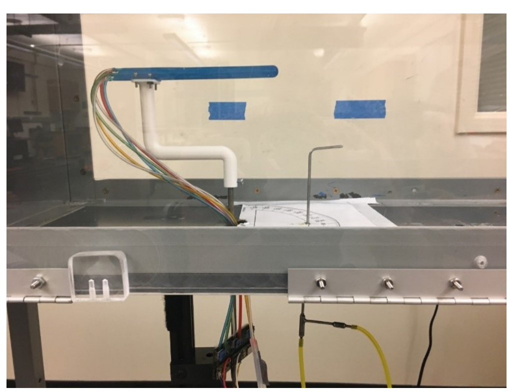


Figure 26: MHP Mounted in Wind Tunnel

Because of the nature of the non-nulling calibration method (yawing sweeps) and 5 hole probes having pitch and yaw angular observations, only the one set of pressure holes will receive pressures differential in a given sweep. Therefore the probes will needed to be mounted and swept in both an alpha and a beta orientation independently to properly calibrate them, as seen in Figure 7.

(a) Alpha Orientation (b) Beta Orientation

Figure 27: Calibration Sweep Orientations

After the probe is mounted securely to the wind tunnel, the Tygon tubes are run through the bottom of the wind tunnel and connected to the digital sensor board. The board is then turned on by connecting the micro-USB port to a 5V power supply. Then, the wind-tunnel is turned on to a fixed speed and allowed to run continuously to reach steady-state (30 seconds). Once this is done, the stepper motor program is started and the probe moves from -45 degrees to +45 degrees in 5 degree increments. The range was chosen because the operating range for similar MHPs has been proven to be around -30 to +30 degrees. Thus the entire operating range of the probe is captured. The probe is held at each step for 30 seconds, allowing the tip flow to reach steady state and gather enough data to acquire an accurate average at that point. It is worth mentioning that the response time for the probe is much shorter than than 30 seconds ($\sim$10ms), but this allows for a more accurate calibration. The sweep is then repeated for each test cycle, and all three sweeps are averaged together for that specific speed. There are three total speeds the probes were tested at: 10, 15, and 20 m/s for both alpha orientation and beta orientation, totaling nine test per orientation. Once the experiment is performed, the data is offloaded via micro SD card, parsed in Matlab, and the graphs are generated.

Calibration of the multi-hole probes in known flow-field and the curves associated derived from the calibrations is critical for accurate in flight measurements. The key to calibrating MHPs are the coefficient of pressures. The pressure coefficient is a non-dimensional feature that is defined by the pressure difference over the dynamic pressure. The dynamic pressure is defined as,

$$q = p_a - p_\infty = \frac{1}{2} * \rho_\infty * U_\infty^2 \tag{3.1.1}$$

For the pitch angle, this is

$$C_{p\theta} = \frac{P_2 - P_4}{P_1 - P_a} \tag{3.1.2}$$

and for the yaw angle, the equation is

$$C_{p\phi} = \frac{P_3 - P_5}{P_1 - P_a} \tag{3.1.3}$$

and for pitot coefficient of pressure

$$C_{pPitot} = \frac{U_{pitot}}{U_\infty} \tag{3.1.4}$$

As shown in the Figure 8 the axis being tested $C_{p\theta}$ has a slope of some gradient, whereas, $C_{p\phi}$ is nulled and has zero or nearly zero value. Another important parameter for MHP's is the angular operating range. As Figure 9. depicts, the further along the curve within the linear range gives the angular range. As mentioned before, different MHP's tip geometries affect the MHP's operating angle range. Outside of the nonlinear curve of the probe the angle sensitive is less, therefor, maintaining values of C_p that correspond to the associated angle is harder to correlate.

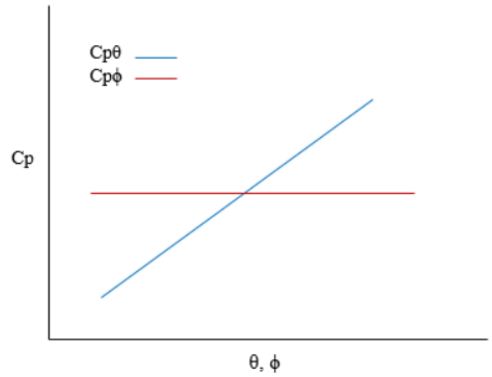


Figure 28: Notional pressure coefficient curves for pitch and yaw depicting a 5HP traversing on the pitching axis while the yaw axis nulled

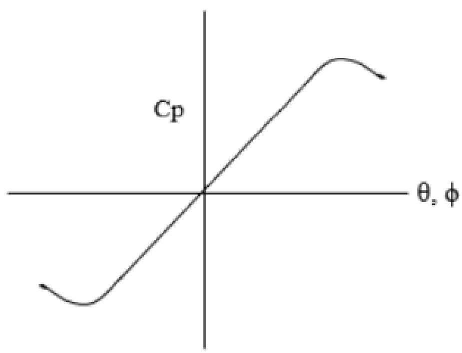


Figure 29: Illustration depicting pressure coefficient linear range

Figure 10. shows the decomposition of velocity, U, where the rotational axis for pitch and yaw θ and ϕ are determined by from pressure differences on their respective axis using Equations 2 and 3.

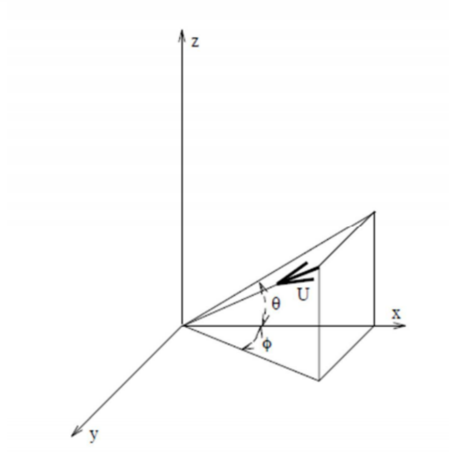


Figure 30: Orientation of Vector pitch and yaw angle

Using the Bernoulli's equation the magnitude of the velocity vector can be resolved.

$$U = \sqrt{\frac{2}{\rho_\infty} q} = \sqrt{\frac{2}{\rho_\infty}(p_o - p_\infty)} \quad (3.1.5)$$

Fig. 11 shows a ideal velocity magnitude curve with symmetry about about θ and ϕ. As the probe is symmetric, the curve should ideally be symmetric about θ and ϕ.

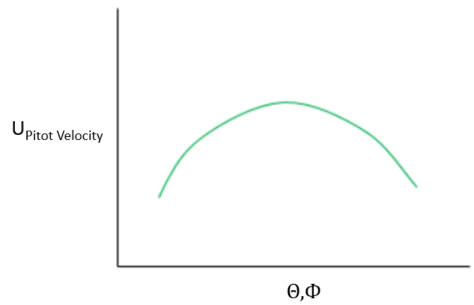


Figure 31: Ideal Velocity Magnitude Curve

After calculating the magnitude and obtaining the direction of the velocity, the three velocity components can be be calculated by using equations 3.1.5, 3.1.8, and 3.1.7.

$$Re = \rho * u * L/u \quad (3.1.6)$$

$$w = Usin(\phi) \quad (3.1.7)$$

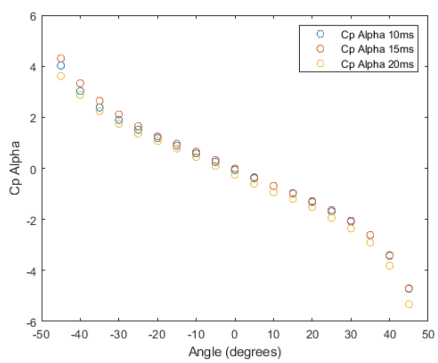
$$v = Usin(\theta)cos(\phi) \quad (3.1.8)$$

The main measurement gain from using MHP is velocity, the effect of Reynolds number should be considered for calibration [20]. Increasing the know flow-field velocity of the wind tunnel will increase the Reynolds regime as well to its relationship to velocity.

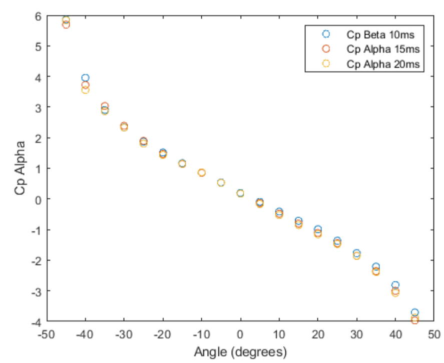
$$Re = \frac{\rho * U_{\infty} * D}{u} \tag{3.1.9}$$

During the calibrations a range of Reynolds number should be set. The range of the Reynolds number is dependent on the expected operating range of the MHP. For these MHP's that operating range is the flight regime of the UAVs it being is mounted to (10 - 25 m/s). With that being stated multiple speeds of the flight regime should be tested to ensure the consistent of the MHP calibration. For these probes speeds of 10, 15, and 20 m/s were chosen.

Calibration results for the 3D-manufactured probes are presented below for both the pressure coefficient and velocity magnitude curves.

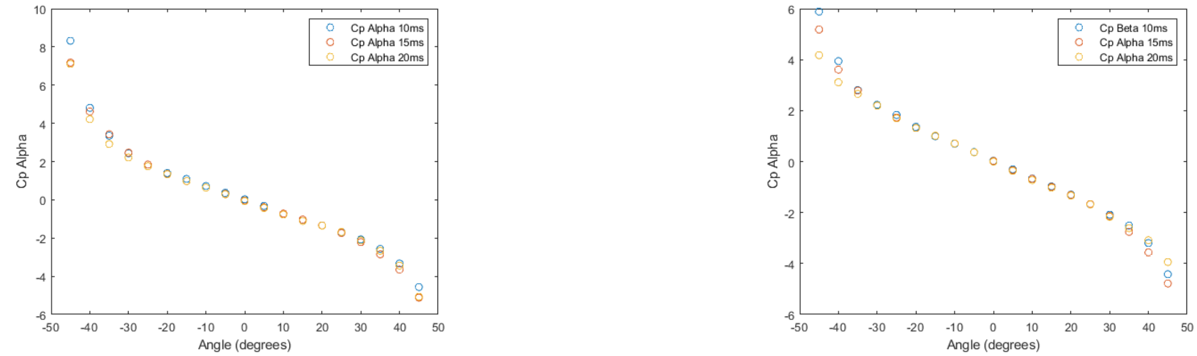


(a) $C_{p\alpha}$ Vs Angle



(b) $C_{p\mathrm{B}}$ Vs Angle

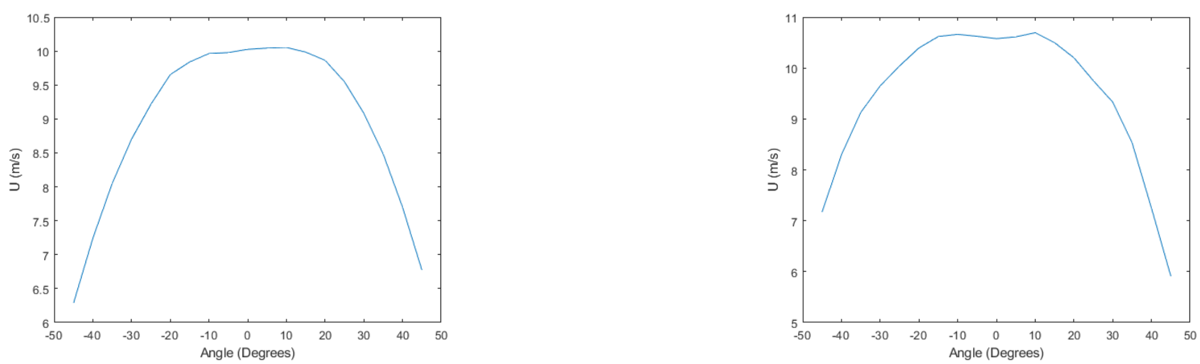
Figure 32: Probe2 (a) C_p Alpha Vs. Angle and (b) C_p Beta Vs Angle



(a) C_p Alpha Vs Angle (b) C_p Beta Vs Angle

Figure 33: Probe3 (a) C_p Alpha Vs. Angle and (b) C_p Beta Vs Angle

Note the linear range of the Probe2 falls offs significantly at the outer ranges of the MHP (45 and -45). This confirms past research in that it shows the probes performance is not adequate at these angle ranges and therefor should only be operated in angles within its linear range of (30 to -30) degrees. Figure 12 also shows the similarity of C_p values across all three speeds tested. The linear relationship for both probes has a R^2 value of 0.95 (almost 1) indicating there is a good overall data fit. The Probe3 results are shown in Figure 13. The comparison between the two probes demonstrates the repeat-ability of the probe design.



(a) Probe2 U_{pitot} Vs Angle (b) Probe3 U_{pitot} Vs Angle

Figure 34: Magnitude Pitot Velocity Vs Angle

Figure 14 show the magnitude velocity curve for the 3D-manufactured probes. The figures show symmetry of the maximum velocities throughout the traverse from -45 to +45°

demonstrating the repeat-ability of the 3D-manufactured probes

Validation

After proper calibration of the probes the calibration data can be tabled and used to interpolate velocities and angles from unknown measurements. Development of the 3D-manufactured probe is done with the intended use on UAS fixed winged vehicles in mind. The intended vehicles mostly stay under 40 m/s as the flight speed. This allows for the interpolation process for probe to be simplified since the probe will stay in a laminar flow for the duration of measurements. This is significant because the three major quantities used for interpolation, $C_{p\alpha}$, $C_{p\beta}$, and C_{ppitot} will not have significant changes over the expected air speeds.

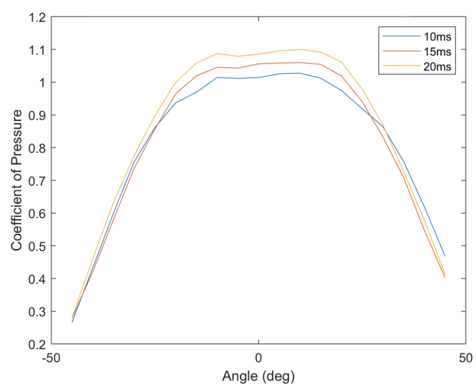


Figure 35: $C_{p_{pitot}}$ Change with Velocity and Angle

Figure 15 illustrates how C_{ppitot} changes from 10 to 20 m/s. The C_{ppitot} trends for each or the three speeds are nearly identical. For interpolation purposes the assumption that no significant change between speed was made. This same assumption was made for $C_{p\alpha}$ and $C_{p\beta}$ these trends are graphed in Figure 32b. These assumptions allow for the interpolation process to be broken down into five major steps.

1. Calculate $C_{p\beta}$ and $C_{p\alpha}$ from the pressure measurements.

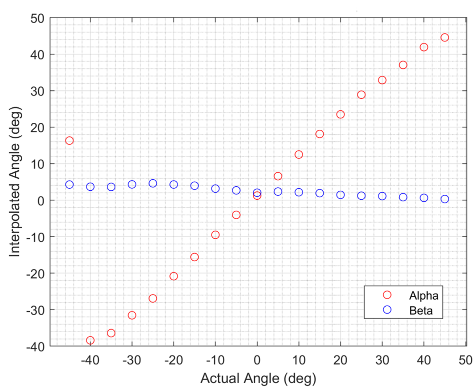
2. Interpolate for Alpha and Beta from the coefficient of pressures.

3. Use Alpha and Beta to interpolate for C_{ppitot}

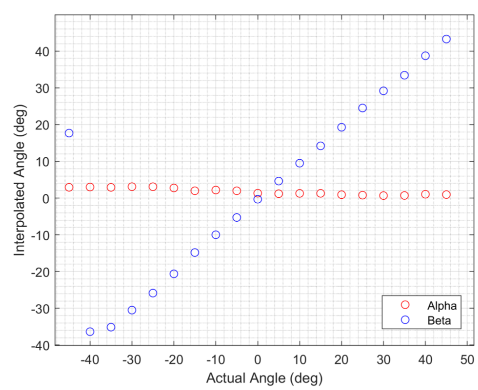
4. Use C_{ppitot} to Calculate U using Equation 3.1.4

5. Using Equations 3.1.5, 3.1.8, and 3.1.7 with Alpha as ψ and Beta as θ resolve the velocity vectors.

As a step to verify that the interpolation method works the code was used to resolve angles and velocities from calibration data. The calibration data provides a good known quantity for to test the interpolations against. The values that are compared are the the values of Alpha and Beta, and the velocities(total and components).



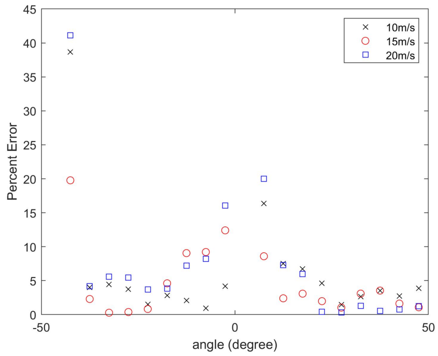
(a) Alpha vs Actual Angle



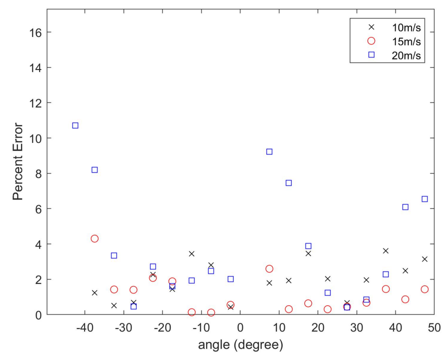
(b) Beta vs Actual Angle

Figure 36: Probe3 Interpolated Comparison to Actual Angle

Figure 36 is a comparison between the known Alpha and Beta angles and the interpolated values. One major outlier is present. The -45°angle has a significant error. This is constant across all three probes and all test. The cause for this error is currently unknown. More test will be done to attempt to isolate the cause of the error. However, the error should be outside the conditions encounter by a fixed wing UAS and should not affect normal data operations.



(a) Percent Error Alpha Angle



(b) Percent Error Beta

Figure 37: Percent Error of Interpolated Angles

Figure 37 shows the percent error for the different angle interpolations across the three different speeds. Note the for the graphs, the -45°percent error was near 100 percent. For scaling purposes it was removed from the graph. There is a peak around the 0°. One of the causes of this could be asymmetry in the probe tip that is unexpected causing pressure differentials. The second interpolation point to verify is the velocities.

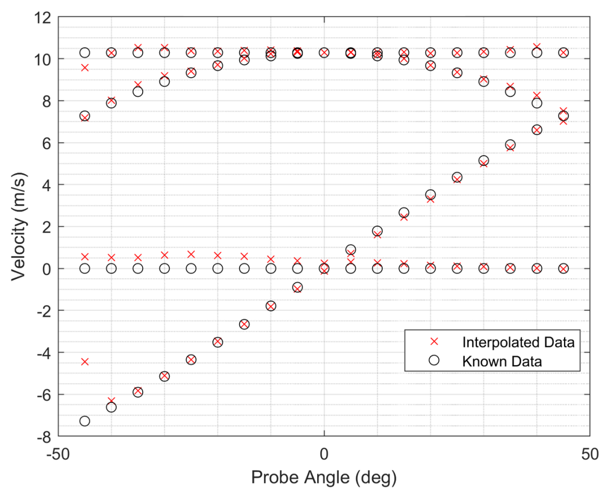
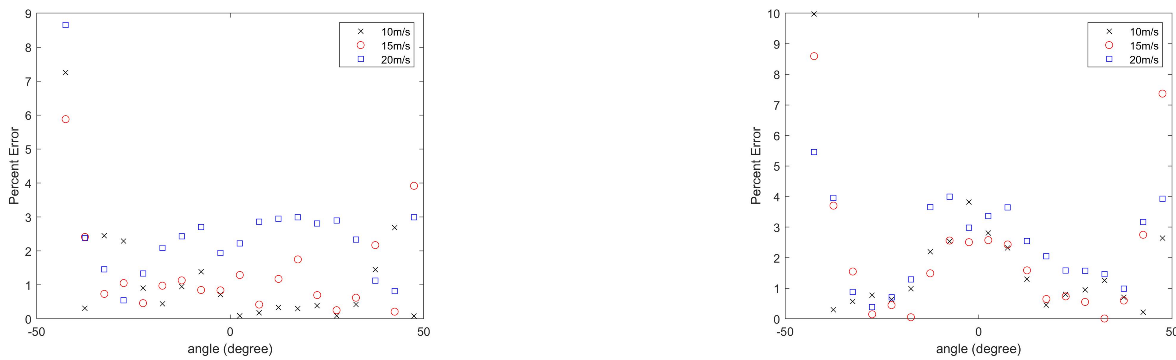


Figure 38: Velocity Comparison for 10m/s Alpha Sweep

Figure 38 shows a comparison between the known velocities and the interpolated velocities for a 10 m/s alpha sweep. As previously noted there is still an anomaly at the -45°step. Other than that issue the interpolated data is relatively accurate.



(a) Percent Error Total Velocity Alpha Angle

(b) Percent Error Total Velocity Beta

Figure 39: Percent Error of Interpolated Angles

Figure 39 is the percent error of the Total velocity interpolations. The errors stay relatively low except for at the 45°extremes. The percent error also increases with velocity. The error increase near zero degrees in the Beta orientation exist the same as it did with the angles. Overall the Beta error has a is less linear. This could be due to print support defects left on the probe. Due to the print method small bumps of support material are left along the side of the probe. While these are sanded they may still cause pressure disturbances along the probe.

3.1.2 Quantization Error

Since the MHP is operating using digital pressure transducers a quantization error is present. The equation for quantization error is as follows:

$$\epsilon_r = \frac{V_{fso}}{2^n} \tag{3.1.10}$$

The Honeywell SSCDRRN001PD2A5 is a 14 bit digital sensor that operates on a ± 5V range. In equation 3.1.10 V_{fso} is the voltage range of the sensor in this case 10V. n is the the bit count of of the senor, 14. This lead to a quantization error of the senor of .0006V. This is the smallest voltage change detectable by the sensor. This voltage change converts to a .00012psi resolution. Using equation 3.1.5 at sea level standard atmosphere with a density of 1.225kg/m^3.The quantization error of .00012psi will be used as the $(p_0 - p_\infty)$. This leads to a velocity precision of $\epsilon_r = 0.014m/s$.

Probe Response

A good understanding of probe response is of interest to begin to understand how the probe will react to gusts. This will also give a good idea of the probes maximum response time and begin to set a standard for frequency response.

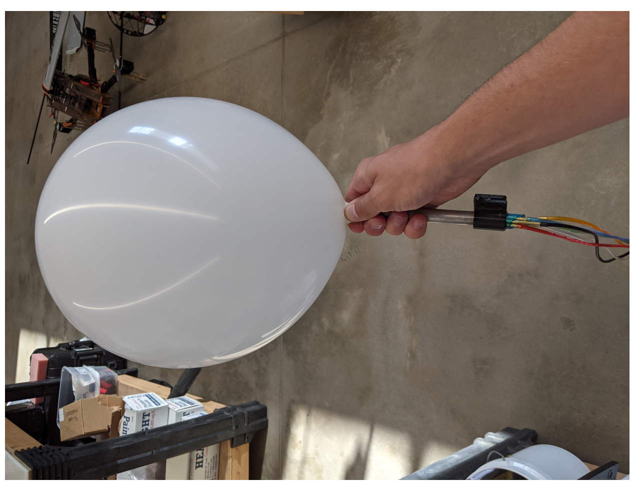


Figure 40: Pitot Step Response

Figure 40 is a picture of the experimental setup. A balloon was placed around the tip of the probe and filled to capacity. Care was taken to leave the static ports uncovered to create a pressure difference between the static and total pressure ports. The balloon was then popped which led to the total port returning to the same pressure as the static port. This should be nearly and instantaneous change.

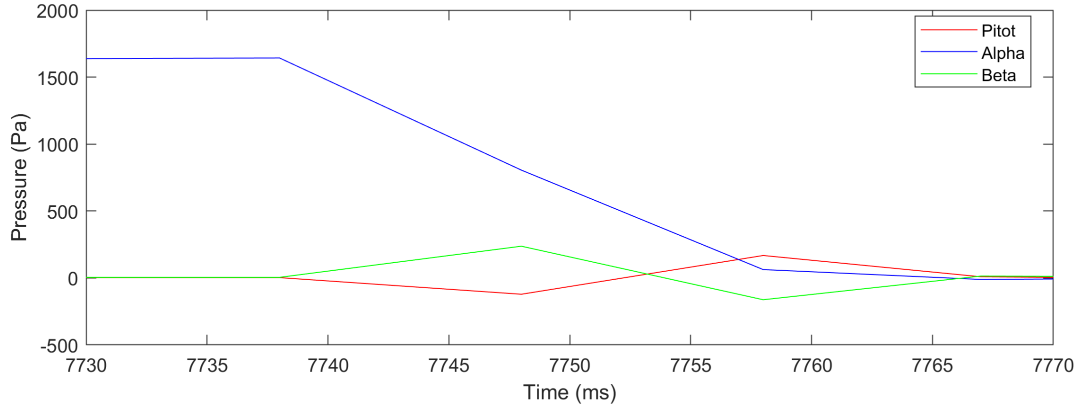


Figure 41: Alpha Step Response

Figure 41 is the response of the alpha pressure transducer to the step response. The responses of the Beta and Pitot sensors can be found in the appendix in Figure **??** and **??**. The three sensors all had a settling of around 20 ms or .02 seconds. Using this response time the probes should have a maximum response rate of 50Hz.

3.2 Ultrasonic Anemometers

Ultrasonic anemometers offer a good solution for wind measurements from multi-rotor UAS. Multi-rotors outfitted for wind measurements have been found to be more accurate than fixed wing methods[15]. Ultrasonic anemometers have large range of size, weight, and accuracy. Many of the more accurate anemometers are prohibitive in weight or size for easy mounting on a multi-rotor. Below is a comparison of the anemometers used to assist calibration and validation of the 5 hole probe.

Table 4: Anemometer Table Part 1

Anemometer	**Use**	**Type**	**Range (m/s)**	**Resolution (m/s)**
R. M. Young 92000[5]	Ground Station	2D Anemometer	0-70	0.01
R.M. Young 5103[6]	Mesonet Tower	2D Anemometer	0-100	N/A
Trisonica Mini[3]	Quad Mounted	2D Anemometer	0-50	0.01

Table 5: Anemommeter Table Part 2

Anemometer	**Wind Speed Accuracy**	**Wind Direction Accuracy**
R. M. Young 92000	0 - 30 m/s ±2% or 0.3 m/s, 30 - 70 m/s ±3%	±2°
R.M. Young 5103	± 0.3 m/s (0.6 mph) or 1% of reading	N/A
Trisonica Mini	0 - 10 m/s ±0.1 m/s, 11 - 30 m/s ±1%, 31-50 m/s ±2%	±1°

All of the wind sensors are assumed to have the manufacture defined accuracy and resolution. The two anemometers operated by Oklahoma State use a TEENSY based data logger. Both of those sensors communicate via a RS-232 protocol, RS-232 uses ±13 volts for communication. The TEENSY is equipped for TTL communication which operates via a ±3.3 volt signal. To make these compatible a SparkFun RS232 Shifter is used to convert the ±13 volt signal to a ±3.3 volt signal. This allows for communication between the sensors and the TEENSY without damaging the TEENSY.

Figure 42: Anemomet Trisonica Mini

The TriSonica Mini(Figure 42) is small light weight ultrasonic anemometer. The anemometers size and weight make it a good choice for use on a multi-rotors. The sensor weights 50 grams(2oz) and measures 9.1x9.1x5.2cm. The manufacturers posted range, resolution, and accuracy are shown in Table 4. The sensor outputs wind speed and direction at 4Hz using a RS-242 communication protocol. The Trisonica operates at a Baud rate of 9600.

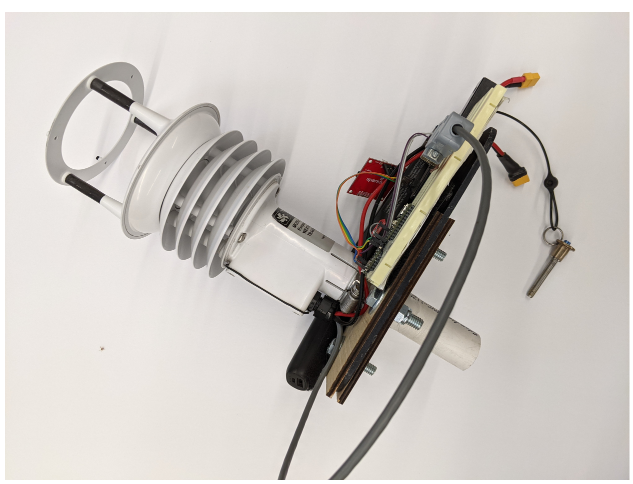


Figure 43: R.M. Young 92000 on Mount for Tower Operations

The R.M. Young 92000 is used as a reference anemometer for calibration flights. The R.M. Young 92000 is a 2D ultrasonic anemometer. The range resolution and accuracy can be found in Table 4. The anemometer is design to be mounted to a post for continuous wind measurement. The sensor outputs 2D wind and direction at 4Hz using a RS-242 communication protocol. The Young 5103 operates on at a Baud rate of 38400.

Figure 44: R.M. Young 5103

The R.M. Young 5103 is relevant for flights against around Oklahoma Mesonet site. The Oklahoma mesonet reports averaged wind speed and direction from the Young every 5 minutes. The Young 5103 output an anolog signal at 90Hz. A 1.0m/s wind is required for the sensor to register an accurate speed. A 1.4m/s wind is require for direction measurements. As discussed in the Chapter 2, since the Young 5103 is a propeller vane anemometer it has inherit lag for wind speed and direction. For the wind speed the 5103 has a distance constant of 2.7 meters with a 63% recovery. The direction has a 1.3m distance constant with a 50% recovery.

CHAPTER IV

INSTRUMENT INTEGRATION AND FLIGHT TESTING AND CALIBRATION

4.0.1 Data Acquisition System Architecture

The data acquisition system used for 5-hole Probe and anemometer sensors is built around a Arduino Teensy micro controller. The system functions for Arduino Teensy 3.6 or 4.1. The Teensy is programmable through the Arduino IDE with the Teensyduino software package. The Teensy has a good balance between function and size. The Teensy 4.1 has the dimensions of 2.4in x .7in. The Teensy has pins for 8 serial connections, 3 SPI connections, and 3 I2C.

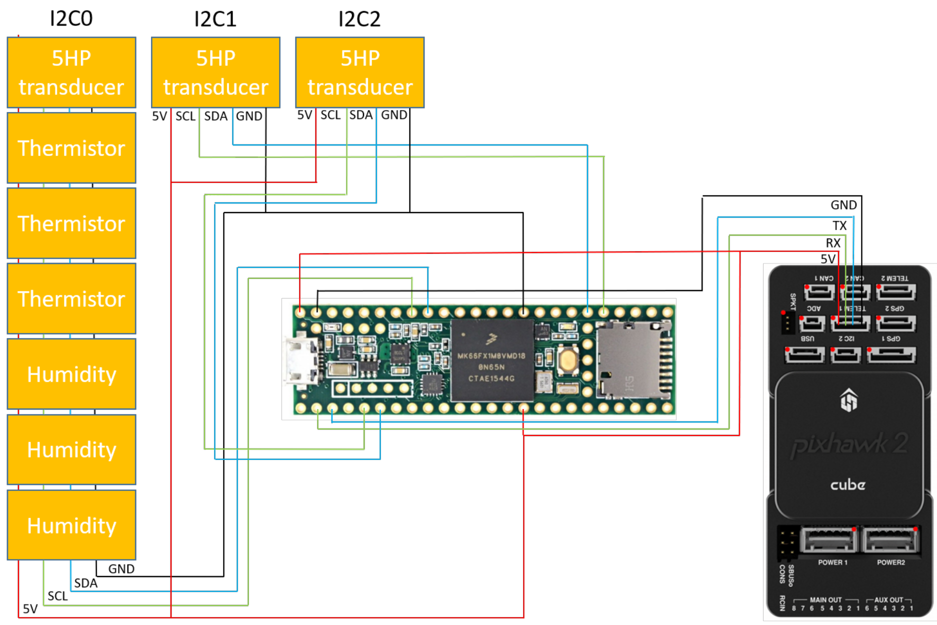


Figure 45: Wire Diagram of Logger System

Figure 45 is the wiring diagram for how the Teensy interfaces with the Pixhawk and the sensors. The temperature and humidity sensors on the left side of the schematic are optional and are not installed on the Nano Talon. The temperature and humidity segment of the DAQ system are also still in early stages of development, they are not be calibrated or validated. Because of this when temperature and humidity are needed I-met sensors are used. The Nano Talon does have slightly different layout due to the different autopilot. Instead of the Pixhawk Cube with a carrier board the Nano Talon uses the mRo AUAV 2.1.

One of the major issues of MHP systems is time syncing the data of all of the sources[11]. The DAQ system is designed to minimize this time issue. This time error minimization is done by sharing the Pixhawk board time to the Teensy. This is done over the RX TX UART lines between the Pixhawk and Teensy using the MavLINK protocol. Both devices keep time via there clock cycles. Since this is shared between the and logged on the Teensy, the hypothetical time lag between data sets would be just the time it takes for the two systems to communicate. Although there is also an inherit lag in the physical interaction of the MHP with the surrounding air. This time comparison is the foundation of the data processing method.

4.0.2 Aircraft Description

The multi hole probes are integrated with 2 different vehicles the VTOL Nimbus and the Nano Talon. The two vehicles are pictured below.

(a) VTOL Nimbus

(b) Nano Talon

Figure 46: Integrated Vehicles

The Nano Talon and the VTOL Nimbus were selected for weather operations due to the ability for the aircraft to be quickly deployed. The Nimbus is able to be taken off and landed in a VTOL configuration. This allows quick and simple deployment that does not require an special launch methods or a runway. The Nano Talon is a small hand launched foam aircraft allowing for quick and easy deployment from nearly anywhere. The aircraft is belly landed this is possible on a variety of surfaces, although grass is preferred.

Both aircraft are controlled with Pixhawk flight controllers. The Pixhawk operates using Ardupilot control algorithms. The Ardupilot built in Kalman Filters are used to remove the aircraft states from the probe measurements. As a result the accuracy and errors in the wind estimation of the system are significantly influenced by the quality of the Pixhawk sensors and Kalman Filter.

4.0.3 Sensor Mounting

The process to integrate and use a multi-hole probe on a UAS can be broken down into 4 different steps.

1. Wind Tunnel calibration of probe
2. Physical Integration on Aircraft
3. System Error and Offset Calculation
4. System Validation

Aligning the sensor with the aircraft axis is important to avoid bias errors. The mounting methods of the Nimbus and Nano Talon are both designed to keep the sensor central. However, one challenge with MHP measurements from small UAS is consistent mounting. Much of this bias has been able to be addressed in post processing[11].

Nano Talon

The Nano Talon mounting presented a challenge due to the small payload volume of the aircraft. The mounting method for the Nano Talon went through three different iterations. For all of the mount configurations the data acquisition board was left in a payload area in the belly of the Nano Talon.

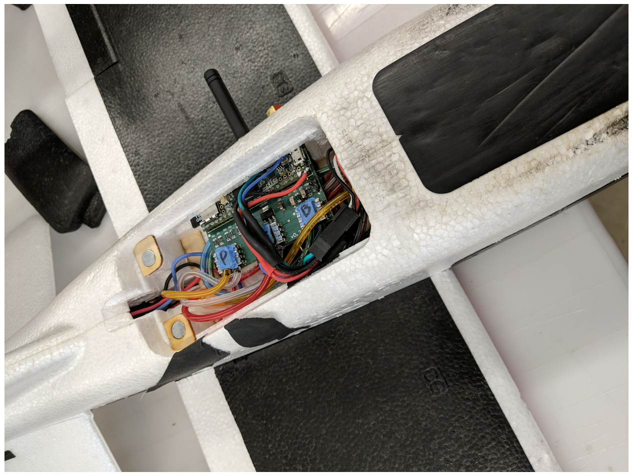


Figure 47: DAQ in Nano Talon

Figure 47 shows how the DAQ is integrated into the Nano Talon. The board sits on top of a RFD 900 Telemetry unit. The RFD 900 is used for telemetry link to a ground station. Attached to the pressure sensors on the DAQ is 6 15.5 in lengths of pitot tubing. This tubing runs up front to the probe at the nose of the Nano Talon.

Figure 48: Nano Talon Probe Mount

Figure 48 shows the location of the final probe mount. This location allowed the pitot tubing to be passed through a hole in the nose back to the DAQ. It also avoided conflict with the battery since the probe comes off with the hatch. This location is centrally located on the aircraft and the probe is place carefully to minimize any issues with offsets from the aircraft axis.

Nimbus

The Nimbus has a large payload area in the nose in front of the battery this is where the the DAQ is stored. The Nimbus has 21 inches of pitot tubing between the DAQ and the probe. The probe is mounted to a wood plate integrated into the nose.

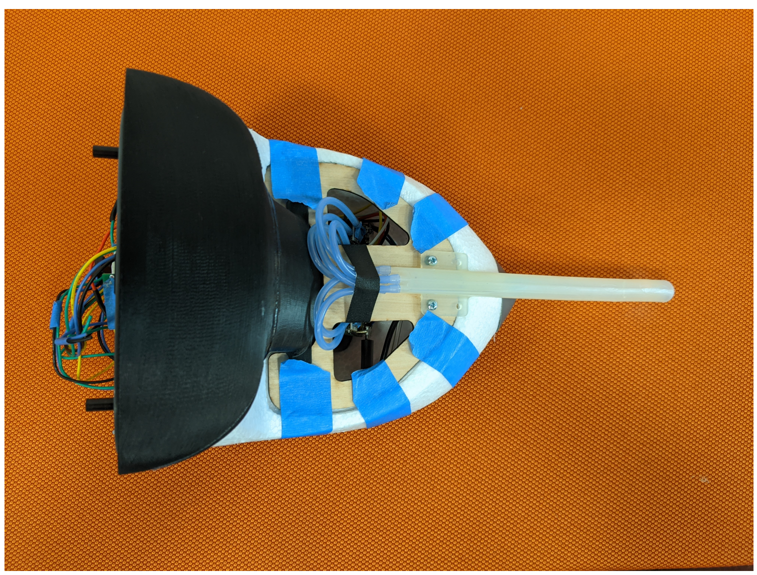
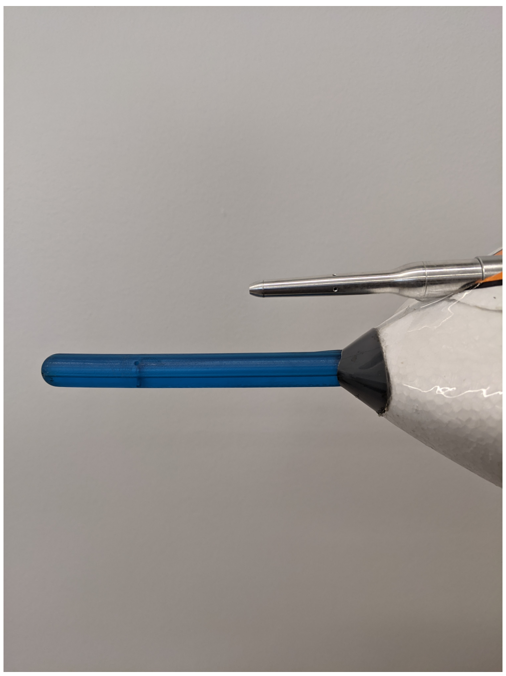


Figure 49: Nimbus Probe Mount

The probe is then bolted to the wooden plate. Figure 49 shows a top view of the nose section. The black section is a airflow duct for the integration of temperature, pressure, and humidity sensors.

4.0.4 3D Print Durability Issues

Fully additive manufactured multi-hole probes have been proven to be accurate in a wind tunnel setting[14]. Significant problems with the probes became apparent with repeated field use. The probes printed with the Formlabs Form 2 printer. Two different resin types were tested for in the field. The first resin used is Formlabs Tough resin, unfortunately Formlabs does not have material properties posted for this material. The other used is Formlabs Durable resin. The technical data can be found on Formlabs website[7].



(a) Probe Bending From Normal Use

(b) Damaged Probe Tip

Figure 50: Probe Wear

Figure 50 shows damage to the probe that occurred from regular use. Figure 50a show warping that occurred to the probe this was observed even when the probe was in storage, although transportation increased the chance of the warping. This warping got severe enough to be an issue and make the calibrations invalid. The significant scrapes on the probe tip shown in Figure 50b presents a similar issue.

Figure 51: Broken Probe Pitot Attachment

The most significant issue with the fully 3D printed probe was the nubs the pitot tubing attach to breaking off. These nubs are very brittle and multiple flight tests were cut short due to them breaking. Due to the small nature of both aircraft the probe was always in an area where flight operations such as replacing batteries risked breakage. These issues with the 3D print material led to a switch to the hybrid probes for flight testing. These probes proved very durable and reliable.

4.0.5 Wind Vector Realization

This section will walk through the process of taking raw 5 hole probe and aircraft data and deriving wind vectors from it. The flight data used is from flight tests on 6/2/2021. Two different flight were flown, a flight in a wind box pattern and a flight doing orbits. The weather for the flights was very low winds, between 0-4 m/s with the Stillwater airport recording a max wind speed of 8mph or 3.5m/s at the times of the flights. The wind was blowing SSW shifting to NNW. The temperature for the flights was 74°F with a relative humidity of 60%. We will start the wind vector realization process on the orbit flight first. This flight was flown between 1:50 and 2:05 CST.

Orbit Processing

Figure 52: Orbit Flight Path From 6/2/2021

Figure 52 shows the flight path flown for the orbit flights. These orbits were at constant altitude with minimal changes in airspeed.

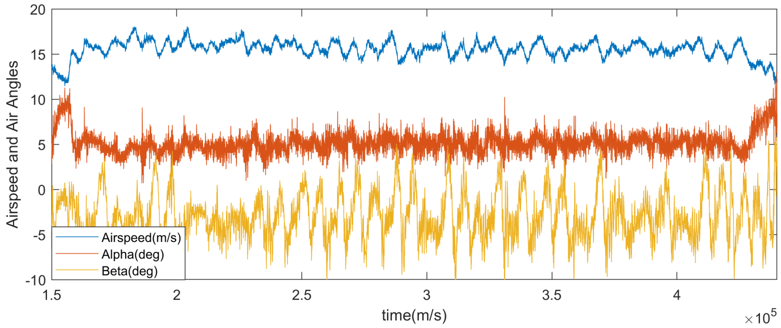


Figure 53: Raw Velocity MHP Data From 6/2/2021 (Orbit)

To acquire a wind vector the aircraft states must be removed from the 5 hole probe measurements. To remove the states from the probe the algorithm discussed in chapter 2 was implemented. Before the algorithm can be implemented the probe data and the aircraft state must be fused. The 5HP runs at 60 to 100Hz. Speeds have been maintained at 180Hz, however not reliably. The Pixhawk Kalman filter runs at 25Hz. To match these data sets the MHP was averaged down to 25Hz for all state removal.

Recalling equation 2.3.8 the aircraft heading (ϕ, θ, ψ) come from aircraft Attitude data. This data comes mostly from compass and IMU calculations. The ground velocity $\overrightarrow{v}_g$ (VN,VE,VD) is derived from the GPS, Barometer, Pitot Probe, and IMU. Alpha, Beta, and, $\overrightarrow{u}_a$ come from the MHP. The wind frame states are the values displayed in Figure 53.

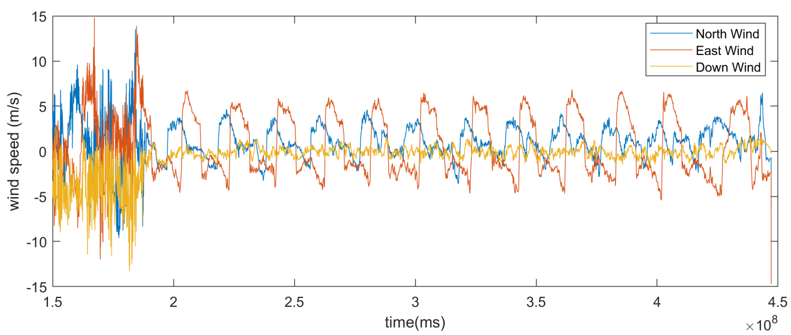


Figure 54: NED MHP Data From 6/2/2021

Figure 54 is the resulting wind vector from removing the aircraft states (other than angular velocities) from the MHP data. The data appears to be very noisy at the beginning this is due to the aircraft being in takeoff conditions not level flight. This kind of data is discarded for wind measurements. For comparison equation 2.3.9 is used to remover the angular rates.

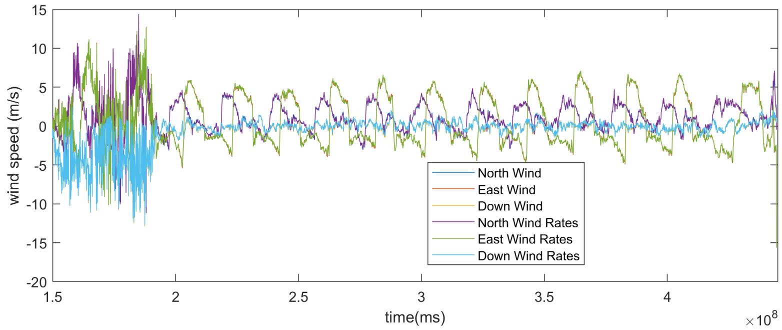


Figure 55: NED MHP Data From 6/2/2021 with Angular Rates Removed

Removing the angular rates let to insignificant differences. The overall wind speed estimation saw a percent difference of .04%. This is because the small lever arm of the aircraft combined with a minimal rotation rate leads to insignificant changes in probe measurements. For example the maximum pitch rate for the aircraft on this flight is 0.0341rad/s with a lever are of .4m that lead to an observed air speed on the probe of .013m/s. For general wind data this is an minimal change. The rates for the rest of the calculations will be assumed to be irrelevant.

Beyond that there is significant oscillations in the wind vectors.This is one of the issues noted by Rautenberg about taking data with orbit patterns[33]. However, the data can be averaged over the orbit period to get an accurate wind vector from an orbit. Since the orbit period is difficult to measure in the field a single-sided amplitude spectral analysis is used. To begin this process first the sample frequency and sample length must be calculated. For this analysis a Fast Fourier Transform (FFT) can be run on the raw direction or airspeed data. For the FFT on this data Matlab's fft function is used[4].

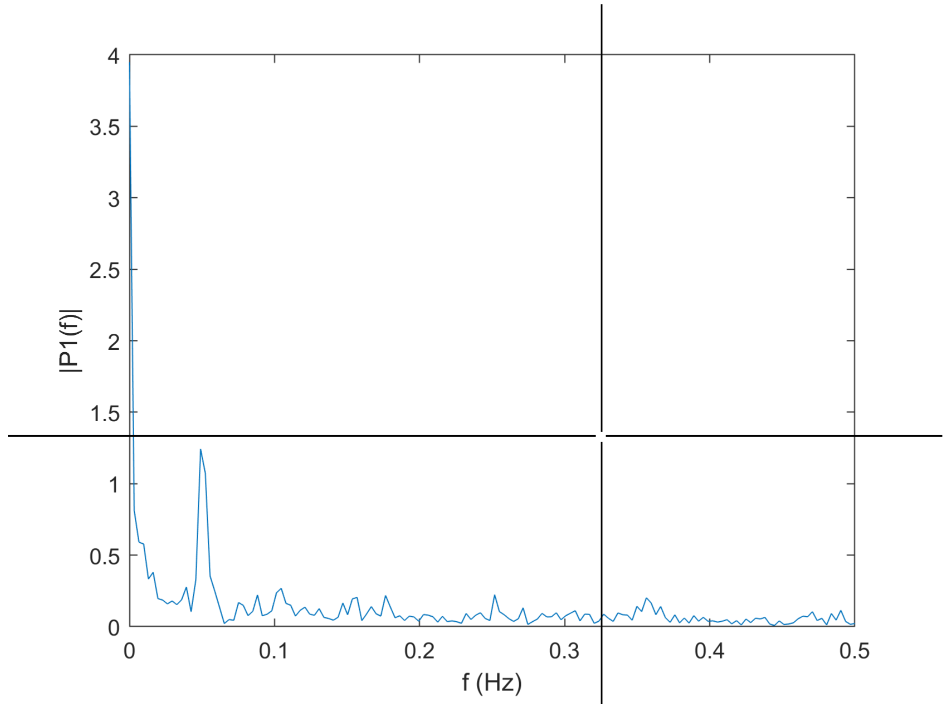


Figure 56: Single Sided Amplitude Spectral Analysis Data From 6/2/2021 (Orbits)

Figure 56 is the result of running the data through an FFT. There is a significant and easily observable spike around a frequency of .05Hz. This relates to a 20 second period which aligns with the typical orbit period for a Nano Talon. The peak is then selected in the Matlab program.

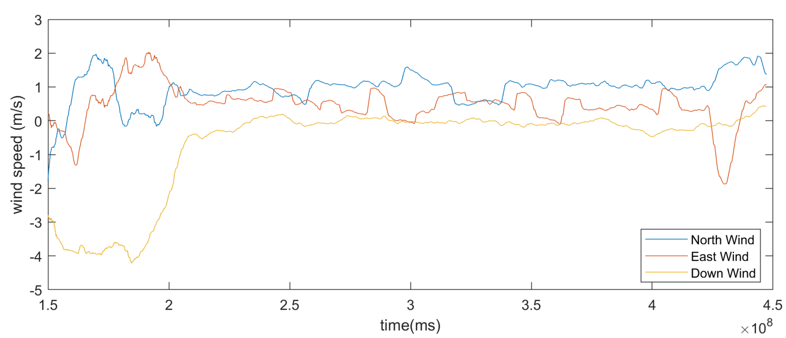


Figure 57: Averaged NED MHP Data From 6/2/2021

Figure 57 is the averaged data. For this average the data has had a moving average with the length of one orbit period applied. The requirement for this average is one thing that makes orbit patterns undesirable. Much of the wind resolution is lost having to average out the orbit period. For wind speed and direction data, which is what is desired for meteorological purposes, the orbits remain useful. For data that does not have to be averaged different flight patterns can be flown. The most popular of these is a wind box.

Wind Square Processing

Figure 58: Wind Square Flight Path 6/2/2021

Figure 80 shows the flight path flown for the wind square flights. These squares are at constant altitude with minimal changes in airspeed.

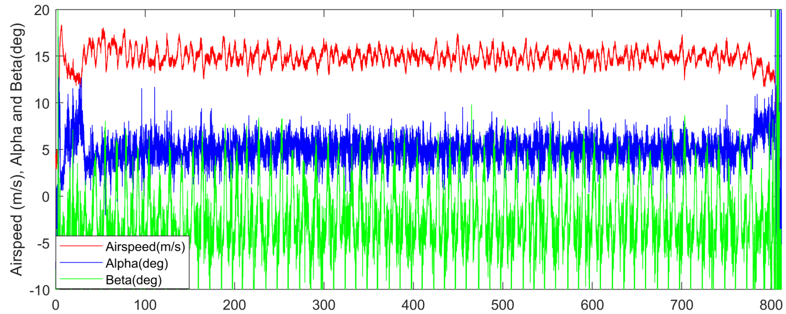


Figure 59: Raw Velocity MHP Data From 6/2/2021 (Squares)

Figure 59 is the raw wind frame data from the MHP from the wind square flights on 6/2/2021. Compared to figure 53 the Beta angle is far less oscillatory. The data has more noise than the orbit data.

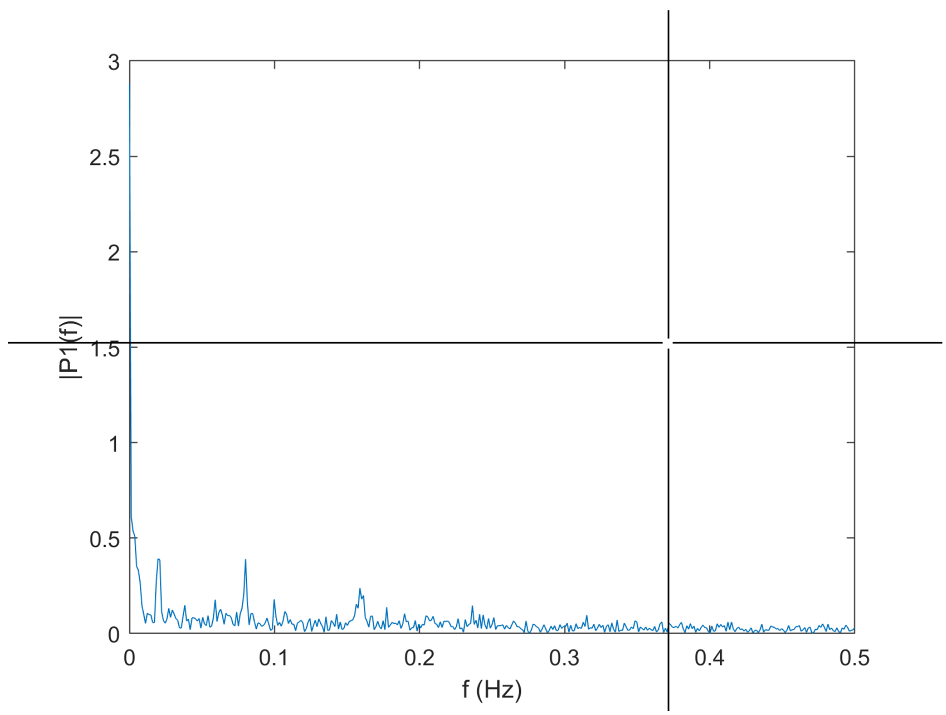


Figure 60: Single Sided Amplitude Spectral Analysis Data From 6/2/2021 (Squares)

Figure 60 is the single sided amplitude spectral analysis data from the wind square flights. While there are some low frequency spikes, a significant one such as the one observed in the orbit data (Figure 56) is not seen. This is because the wind square path does not contaminate the data with an maneuver as badly as the orbits do. This reduces the need to average the data after the aircraft states are removed.

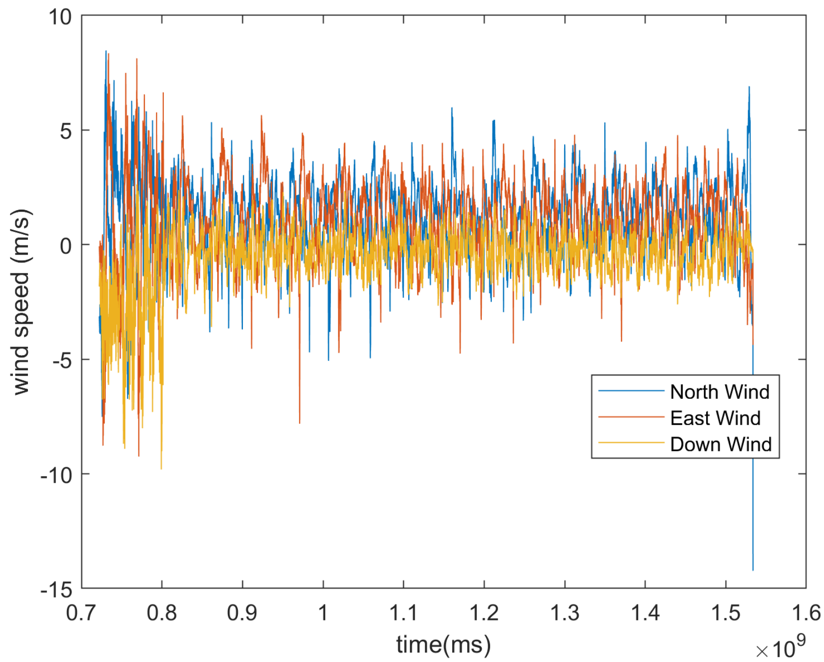


Figure 61: NED MHP Data From 6/2/2021(Square)

Figure 61 is the wind squared data with the aircraft states removed. No a significant oscillations are seen in the data. For baseline meteorological purposes a moving average over the duration of the square is still valuable. This square took around 1 minute for each complete rotation. Applying a 1 minute moving average.

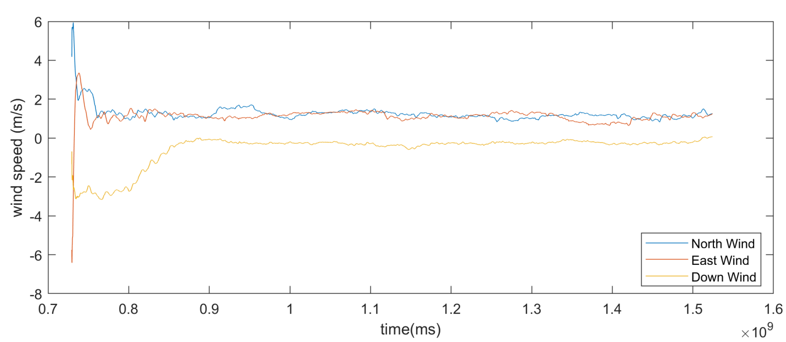
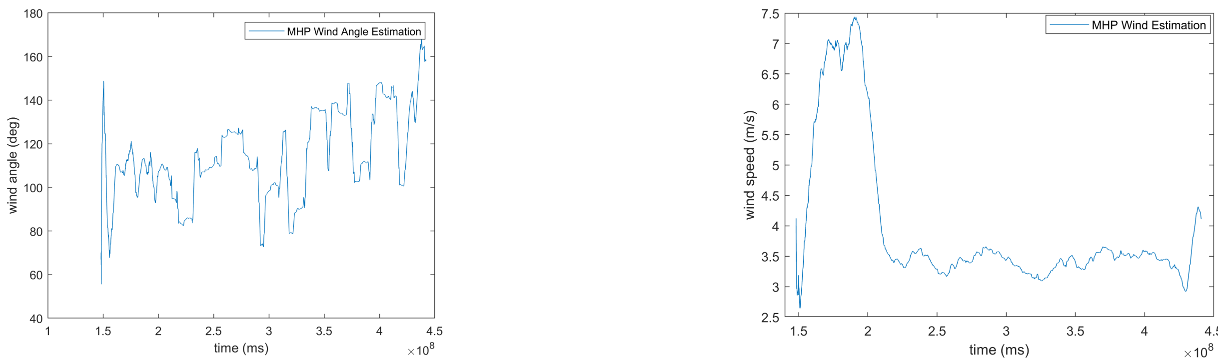


Figure 62: Averaged NED MHP Data From 6/2/2021(Square)

Figure 62 is the wind square data with a moving average of a minute applied. While higher frequency events will be lost in this average it will be valuable to compare to average wind speeds from other sensors.

Wind Vector Conversions



(a) Wind Direction (b) Wind Speed

Figure 63: Wind speed and Direction Graphs

While the Earth based NED is the preferred method of display for meteorology converting the vector to a wind angle and a wind speed can make comparisons between data sources easier. Figure 63 shows the wind ploted as a speed and direction. To plot this the 3D wind vector is converted to a 2D horizontal speed and angle. This is helpful for many of the data comparisons. This is done with the following two equations.

$$Wind_{speed} = \sqrt{(VWN)^2 + (VWE)^2} \tag{4.0.1}$$

$$wind_{direction} = \arctan(\frac{WVN}{WVE}) \tag{4.0.2}$$

In equation 4.0.1 and 4.0.2 WVN and WVE are the North wind vector and East wind vector respectively.

4.1 SKB 1000 - Trisonica Calibration

Figure 64: SKB-1000 with Trisonica Mount

For in flight validation of the probe comparison to a Trisonica mounted to the SKB-1000 is done. The SKB-1000 is a H-quad designed by USRI. The quad has a 6 lb payload capacity with a 20 minute endurance. The Trisonica is mounted to a gimbal system to reduce the effect of the quads movements. The calibration of the quad was done by James Brenner and is fully documented in his thesis. A brief overview will be provided here.

The quad system was calibrated and assessed in a series of flights next to a tower with a Young 92000. It was found that the most accurate location for the Trisonica mount was at the center of the quad. The Trisonica is mounted to a 24in carbon fiber tube. The data logger system for the Trisonica is the same as the one designed for the MHP.

Table 6: Middle 24in Calm Day

Statistics	Result
Avg. Vel. Young (m/s)	1.75
Avg. Vel. Trisonica (m/s)	2.14
Avg. Difference	0.39
RMSE (m/s)	0.57
Bias (m/s)	0.402
Correlation Coeff.	0.71
Variance Young	0.15
Variance Trisonica	0.33
Skewness Young	0.56
Skewness Trisonica	0.34
Kurtosis Young	0.01
Kurtosis Trisonica	0.18

Table 6 is the results of the calibration flights. The root mean square error of the system was found to be around 0.6 m/s. It was noted that higher winds seemed to increase the error of the system.

CHAPTER V

RESULTS

5.1 Validation and Error Analysis

In flight validation of MHPs tends to be a challenge. Literature takes a few different approaches to MHP validation. The three prevailing methods are comparisons to ground stations, comparisons to other aircraft, and comparison to simulations/atmospheric theory. Many research projects have done a mixture or all three. The validation for this probe will involve flying against more validated sensors and an Gaussian error propagation.

5.1.1 Noise Analysis

Before moving onto the validation of the probe the noise in the wind signal will be addressed. The noise is quantifiable through a signal to noise ratio(SNR). SNR can be calculated by taking the mean of a measurement(signal) dividing by the standard deviation(noise) as follows.

$$SNR = \frac{Mean}{\sigma} \tag{5.1.1}$$

Equation 5.1.1 is used to calculate the Signal to Noise ratio. The mean and standard deviation (σ) are calculated from a steady level flight section. In this section the mean was around 18m/s with the standard deviation of 0.25m/s. This resulted in a SNR of 70. Any value greater than one shows a signal that is stronger than the noise. The SNR can be improved by the implementation of a filter.

5.1.2 Gaussian Error Propagation

The first method of analysis of the probe is a Gaussian error analysis. This analysis was done following the methods laid out in Garman et al.[21]. The Gaussian error propagation is done by calculating the error of all of the major quantifiable factors that will contribute to the wind estimation error. These values can then be placed into a Gaussian propagation equation that incorporates these errors.

The probe error is a combination of all the sensor and parameters required for a wind vector as discussed in Chapter 4. These error sources for the wind vector come from two sources the MHP DAQ system and the aircraft state estimator. The sources of error for the MHP system are from two different sources the probe and the pressure transducers. For horizontal wind this will be the angle accuracy of the probe found via the calibration of the probe(α, β). The other measurement from the probe is airspeed (U_a), the precision of this is driven by the pressure transducers. The state estimations of the aircraft aircraft are what is removed from the probe measurement to get the wind vector. This makes the state precision very influential to the error of the system. The roll pitch and yaw precision are used to remove the aircraft angle from the probe measurements. The precision of the attitude states are listed as $\sigma_\phi, \sigma_\theta$, and σ_ψ. The aircraft ground speed are the other state taken from the aircraft estimator. These are removed from the probe speed measurement for the calculation of the wind speed. The precision of the these are listed as σ_{VN}, σ_{VE}, and σ_{VD}. These relate to the

North, East, and down velocities respectively. For integration into the equation the angles (σ_α, σ_β, σ_ϕ, σ_θ, σ_ψ) must be converted to a velocity(m/s). This is done by converting to radians and multiplying by the airspeed of the aircraft. This returns the velocity change that the probe will encounter from the errors. Finally the terms are combined to develope the following Gaussian estimation equation.

$$\sigma_{wind} = \sqrt{(\sigma_\beta U_a)^2 + (\sigma_{U_a})^2 + (\sigma_{VN})^2 + (\sigma_{VE})^2 + (\sigma_\psi U_a)^2 + (\sigma_\theta U_a)^2} \tag{5.1.2}$$

. The alpha and beta uncertainties are taken from the probe calibrations and wind tunnel validations shown in chapter 3. This a 4% precision was found assuming a maximum angle change in flight of ± 10 degrees, from this a precision of .4 degrees can be calculated. The precision of the parameters from the aircraft state estimator are taken from steady level flight data on calm wind days. A 1σ precision is used by taking the standard deviation of the logged states over the steady level flight section.

Table 7: Nano Talon Error Propagation

MRO 2.1x (Nano Talon)		
Variable	Symbol	Precision
Roll Angle	σ_ϕ	1.4 deg
Pitch Angle	σ_θ	1.1 deg
Heading Angle	σ_ψ	1.8 deg
Velocity East	σ_{VE}	0.25m/s
Velocity North	σ_{VN}	0.25m/s
Vertical Velocity	σ_{VD}	0.25m/s
True Air Speed	U_a	0.26m/s
Angle of Attack	α	0.4 deg
Sideslip Angle	b	0.4 deg
σ_{wind}		.85m/s

Table 8: Nimbus Error Propagation

Pixhawk Orange Cube (Nimbus)		
Variable	Symbol	Precision
Roll Angle	σ_ϕ	1.5 deg
Pitch Angle	σ_θ	1.2 deg
Heading Angle	σ_θ	2.3 deg
Velocity East	σ_{VE}	0.41m/s
Velocity North	σ_{VN}	0.41m/s
Vertical Velocity	σ_{VD}	0.41m/s
True Air Speed	U_a	0.41m/s
Angle of Attack	α	0.4 deg
Sideslip Angle	β	0.4 deg
σ_{wind}		1.26m/s

Table 7 is the propagation for the Nano Talon. It has a final uncertainty estimation of 0.85 m/s. The Nimbus is shown in table 8 it has a larger uncertainty estimation because of its higher flights speed which is apparent in the angle precision's affect on the propagation. Overall the Nimbus has higher errors in the compass and GPS. This may be caused by the large motors around the autopilot unit. Further study should be done to minimize this noise.

5.1.3 Kolmogorov's -5/3 Law

One of the initial methods for checking the quality of a MHPs velocity measurements is to check the adherence to Kolmogorov's -5/3 power law. The law is based around atmospheric turbulence having a frequency decay of -5/3.

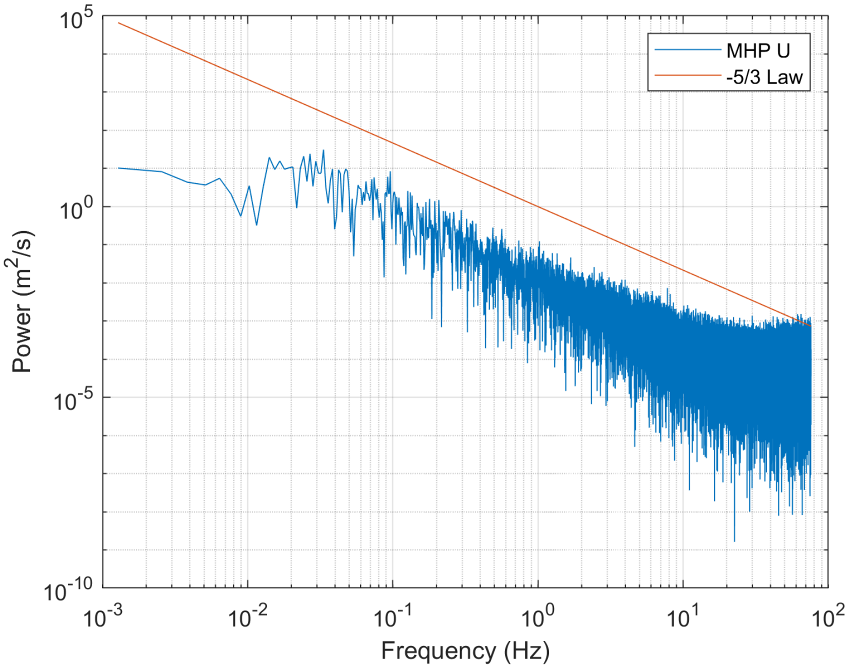


Figure 65: Spectral Power Density with -5/3 Law (Flight Data)

Figure 65 is the power spectral density of a wind square flight flown on 5/28/2021. The blue is the the power spectral density of the raw MHP velocity data. The frequency decay of the data does follow the -5/3 law.

For validation that the -5/3 decay is due to atmospheric measurements and not probe noise a power spectral density was ran on 5HP data when the probe was covered. This will remove any atmospheric measurements from the probe and isolate the data to just probe noise.

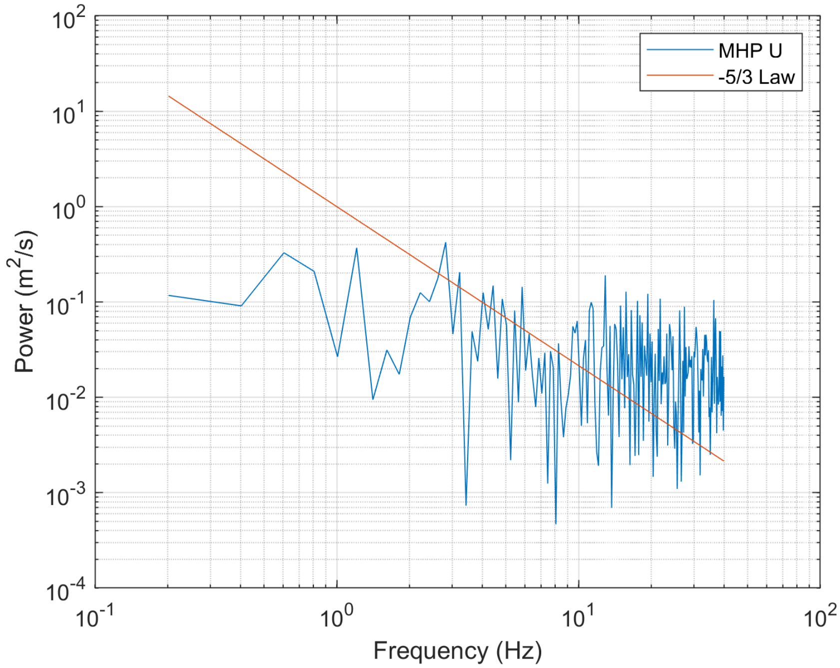


Figure 66: Spectral Power Density with -5/3 Law (Covered Probe)

Figure 66 is the result of running the covered probe through a power spectral density. The slope of the data is mostly flat near zero. This enforces that Figure 65 is representative of atmospheric turbulence.

5.1.4 Mesonet Station

On 4/2/21 the Nano Talon was flown at the Marshall Mesonet Station. This flight campaign was the reason for the switch to a hybrid probe due to probes breaking in the field. However, the flight data shown here is taken from a full 3D manufactured probe. This is the only flight presented in this paper that is done with a fully 3D manufactured probe. The performance was not significantly different between the probe though. The winds for the test were very high around 10m/s the flight speed of the aircraft. This allowed for a unique flight where the aircraft was able to maintain position and point into the wind.

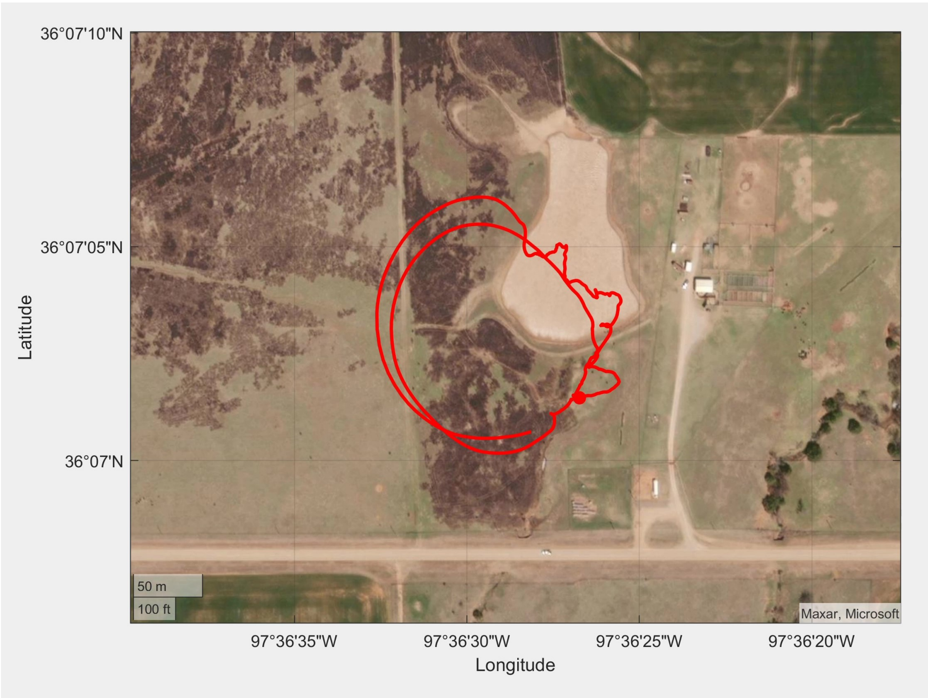


Figure 67: Wind Speed Comparison to Marshall Mesonet Site 4/2/2021

Figure 67 is the first orbit of the Nano Talon. The random lines on the right side are from the Nano Talon sitting in place and moving side to side. This allowed for a good comparison to the Mesonet station.

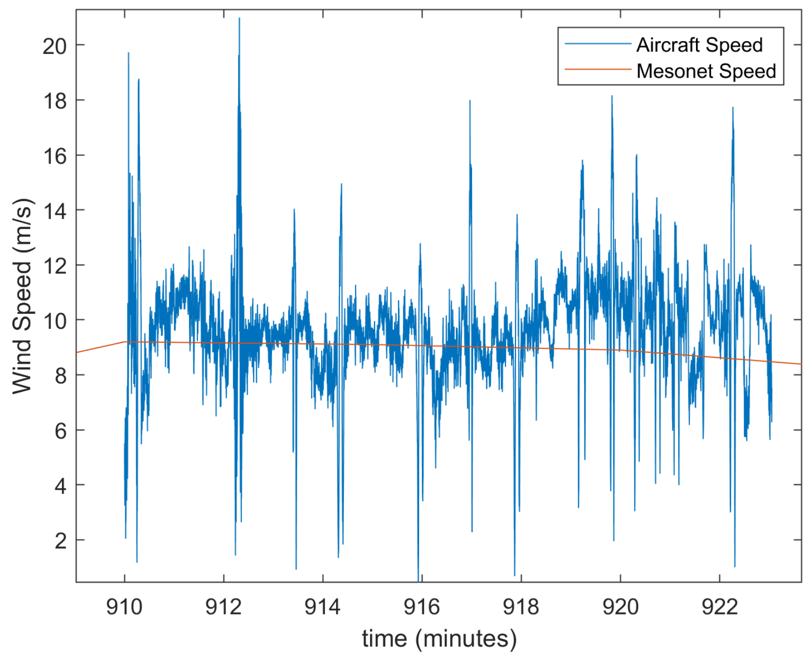


Figure 68: Wind Speed Comparison to Marshall Mesonet Site 4/2/2021

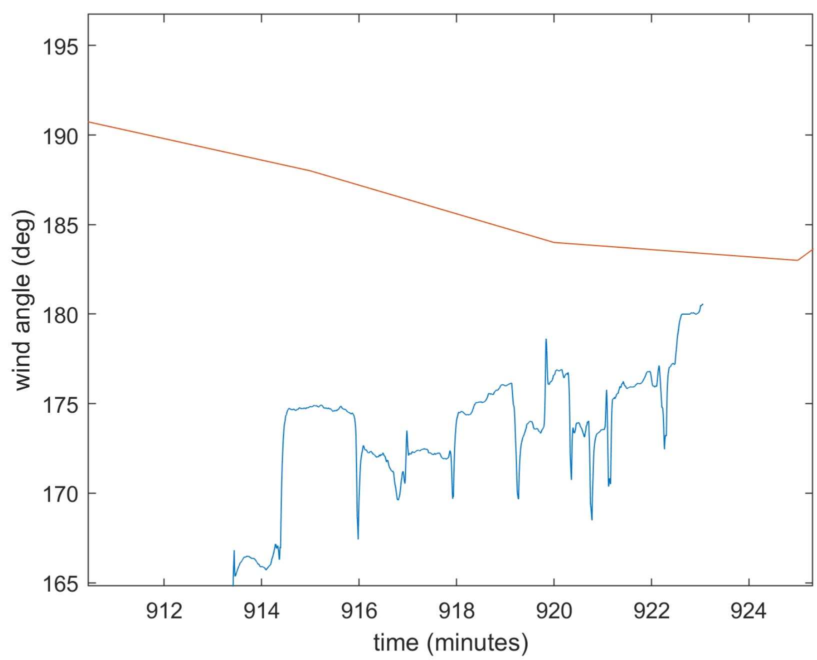


Figure 69: Wind Direction Comparison to Marshall Mesonet Site 4/2/2021

Figure 68 and 69 are the airspeed and direction graphs respectively. The direction graph starts very far from the Mesonet tower. As the flight moves on the value gets closer. This may be a factor of the estimator struggling while stationary. Near the end of the flight the aircraft begins maneuvering more which would help the estimator become more accurate.

Table 9: Mesonet Probe Data Compare

	Mesonet	Probe
Wind Avg (m/s)	8.775	9.64
Angle Avg (deg)	186.5	112.08

The speed comparison is close. The Nano Talon showing a slightly higher speed could be due to the slight difference in altitude. The Nano Talon was flown at 50m the Mesonet tower only has readings up to 10m.

5.1.5 Comparison Flights

The following sections will cover flights where the MHP was flown against other sensors. The sensors flown against are the Young 92000, the SKB-1000 mounted Trisonica, and Young 5103. The details on these instruments can be found in table **??**. The analysis of the Trisonica on the quad-rotor is found in table 6.

All 3 Sensors (Orbit Flights 5/3/2021)

On May third 2021 the SKB-1000 with the Trisonica with an iMET-XQ2 and the Nimbus with a 5-HP and the TPH sensor suite were flown. The test provided a good test for the system performance in dynamic atmosphere. The flights were performed at the Oklahoma State University Unmanned Aircraft Flight Station, East of Stillwater Oklahoma. On the date of the flight a small cold front moved through the area between 1500CST and 1800CST.

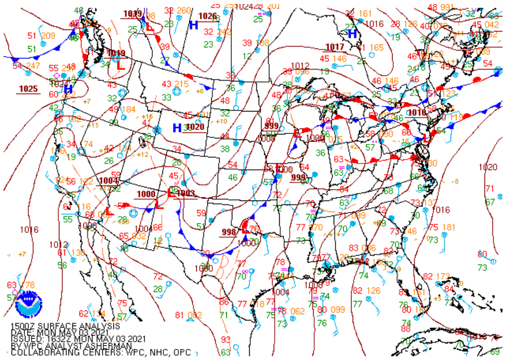


Figure 70: 1500 Surface Map

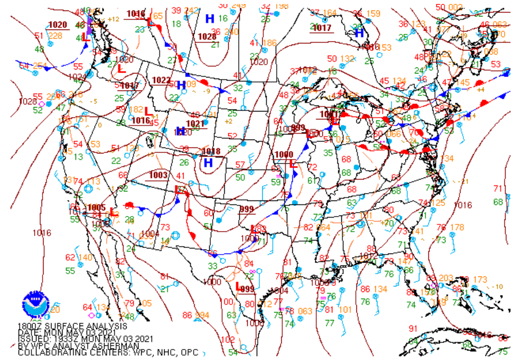


Figure 71: 1800 Surface Map

The Nimbus was flown in a completely autonomous fashion. The SKB-1000 was flown in QLoiter, this mode holds altitude and position. The two flights were flown at round 15:00 CST and 16:00CST aligning with the passing of the cold front and shown in surface charts shown above.

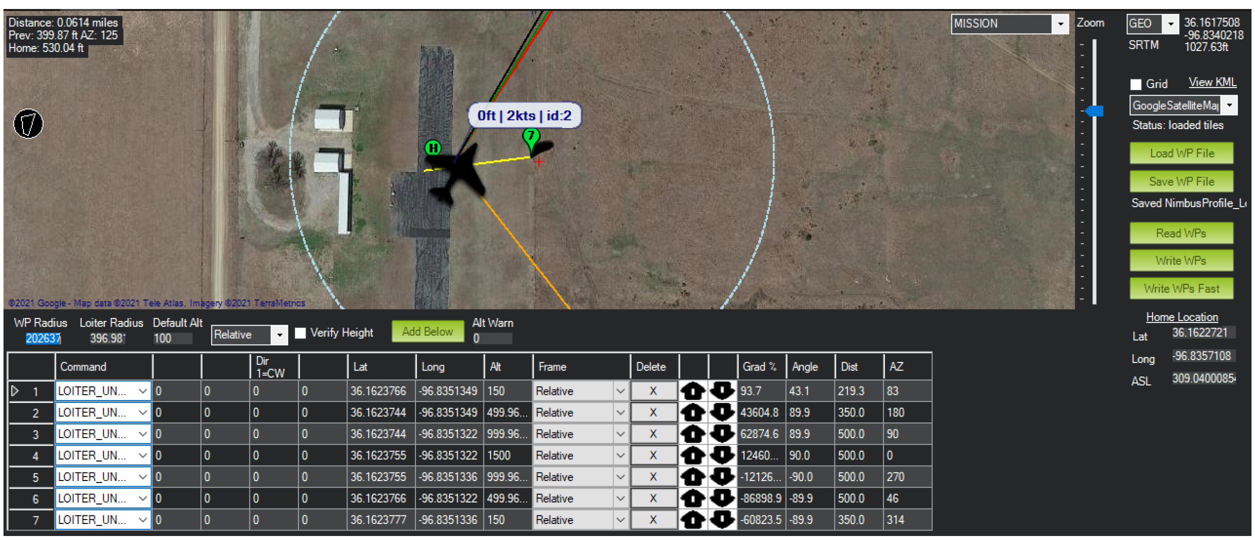


Figure 72: Waypoint Parameters for Nimbus Flight 5/3/2021

The waypoint set up for the Nimbus is shown in Figure 72. The flight was done with 4 personnel two pilots (James Brenner and Kyle Hickman) and two ground station monitors (Eric Abele and Andrew Walsh). The ground station monitors communicated the altitude of each vehicle to each other and the pilots. The ground station operator of the Nimbus would command way point changes (changes in loiter altitude) after the SKB-1000 had reached the current altitude of the Nimbus and 2 full orbit of the Nimbus had been completed. Not ever altitude achieved a full loiters due to battery restraints. Many of the descending loiters were limited to 1 orbit.

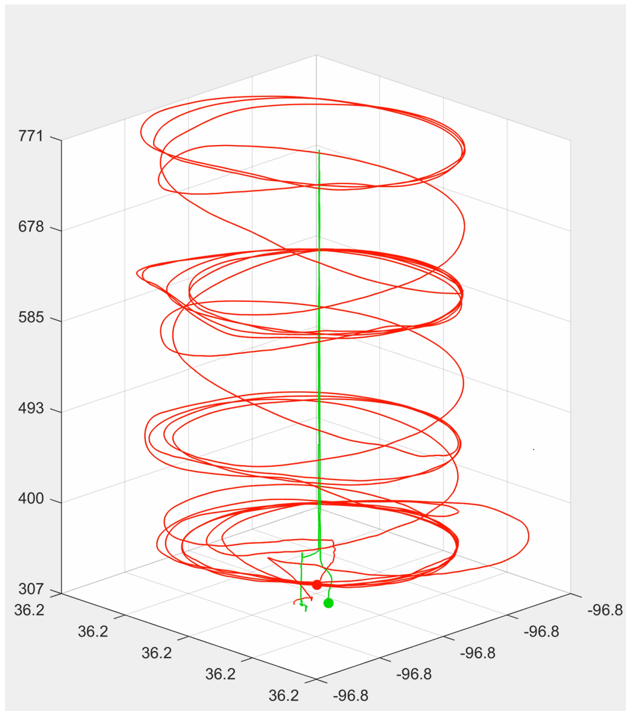


Figure 73: Flight Profile For Nimbus and Skateboard 5/3/2021

Figure 73 shows the actual flight of the Nimbus (Red) and SKB-1000 (Green). This track is for flight 1, however flight 2 follows the same pattern. This pattern was designed to be able to average data at each altitude. Getting 2 full orbits allowed for a the 5HP to fully measure the wind at each altitude since the orbit period must be averaged out from the measurements. The level orbits also provide a time for the 2 vehicles to ensure they are at the same altitude.

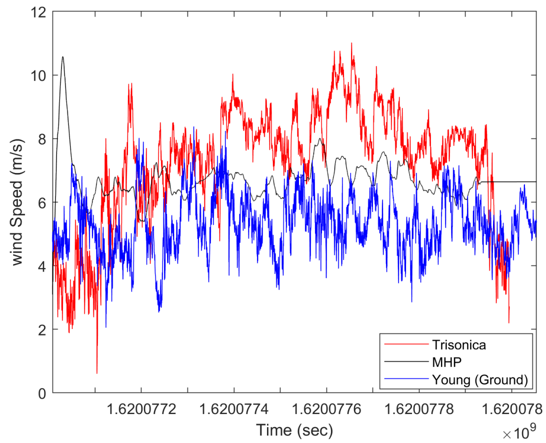


Figure 74: Young, Trisonica, and MHP Wind Speed Plot 5/3/2021 - Flight 1

Figure 74 is a plot of the wind speed measurement of all 3 sensors together. The spike at the beginning of the MHP data is from take off condition and is not included in the analysis.

Table 10: 5-3-2021 Flight 1 Sensor Data Analysis

	Young	Trisonica	Multi-hole Probe
Average Speed (m/s)	5.319	7.596	6.629
Variance	0.897	2.279	0.219
Skewness	0.01	-1.01	0.156
Standard Deviation	0.947	1.51	0.468
Kurtosis	2.826	4.546	3.404

Table 10 displays some different data analysis parameters of each of the data sets. The average speed of the Young is significantly lower than the other 2 sensors because it is ground based and due to the atmospheric boundary layer the airspeed increases as altitude increases. The ground sensor is a good control to have reliable sensor that gives an idea what the atmospheric trends are. The variance are decent indicators of how gusty the wind is. On this first flight wind was had significant gust conditions. It is worth noting that the MHP variance and standard deviation low due to the significant averaging required in the data processing.

Table 11: 5-3-2021 Flight 1 Sensor Comparison

	Correlation Coefficient	RMSE (m/s)
Probe Young	0.116	1.714
Probe Trisonica	0.3884	1.676
Trisonica Young	0.0522	4.349

Table 11 compares the correlation and root mean square error for all three sensors. The Young shows nearly no relationship with either sensor. This is expected since it is measuring the conditions on the ground. The MHP and Trisonica have a moderate relationship. The MHP delay on gust is most likely what causes this lower correlation coefficient.

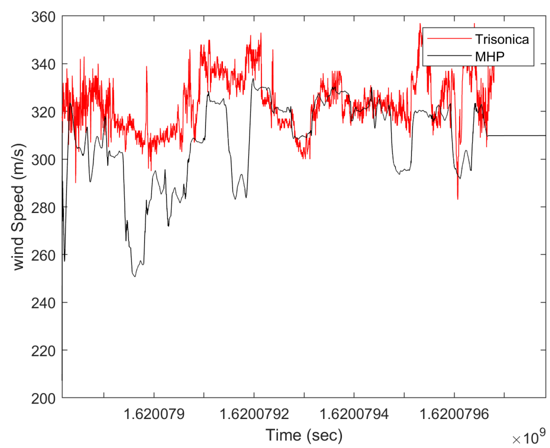


Figure 75: Trisonica, and MHP Wind Angle Plot 5/3/2021-Flight 1

Figure 75 is a plot of the angle through the whole flight of the vertical profile.

Table 12: Flight 1 Angle Data Analysis

	Trisonica	Multi-hole Probe
Average Angle (deg)	312.7	278.9
Variance	512.6	932.6
Skewness	-10	-0.01
Standard Deviation	22.83	30.53
Kurtosis	137.6	1.79

Table 13: Flight 1 Sensor Angle Compare

	Correlation Coefficient	RMSE (deg)
Probe Trisonica	0.06	50.32

Table 12 and 13 is the data from the MHP comparison to the Trisonica. There is significant differences between the two measurements. Not many conclusions can be drawn from this data.

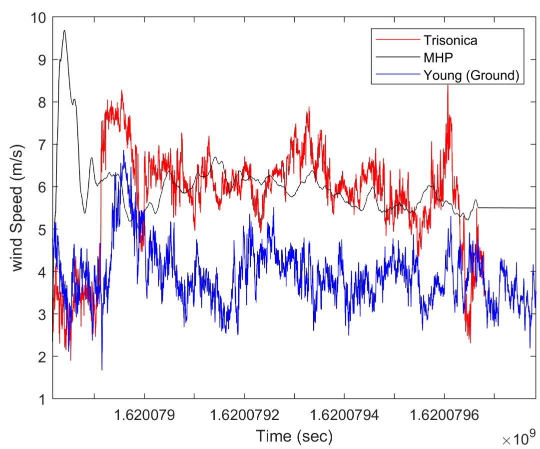


Figure 76: Young, Trisonica, and MHP Wind Speed Plot 5/3/2021-Flight 2

Table 14: 5-3-2021 Flight 2 Data Analysis

	Young	Trisonica	Multi-hole Probe
Average Speed (m/s)	4.017	6.084	5.857
Variance	0.508	0.781	0.129
Skewness	0.929	-0.756	-0.066
Standard Deviation	0.713	0.884	0.36
Kurtosis	4.69	5.217	2.101

Flight 2 had much calmer conditions that flight one. This is illustrated by the variance and the standard deviation for all of the sensors being much lower. This is indicative of more stable and less gusty conditions. As such the MHP performs much better in comparison to the Trisonica.

Table 15: 5-3-2021 Flight 2 Sensor Comparison

	Correlation Coefficient	RMSE (m/s)
Probe Young	0.19	2.061
Probe Trisonica	0.36	0.85
Trisonica Young	0.028	3.86

The correlation coefficients stayed relatively constant with the first flight. The root mean square between the MHP was reduced to ±.85m/s this is most likely due to the more constant wind.

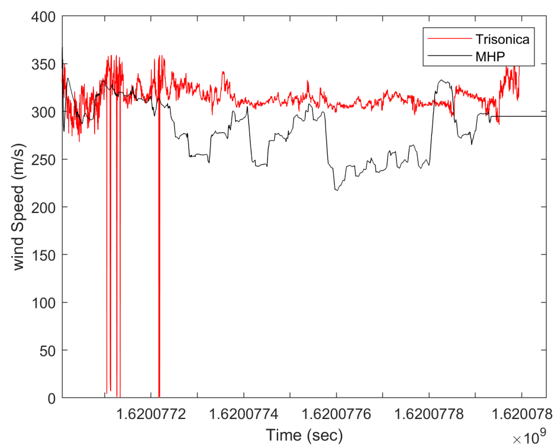


Figure 77: Trisonica, and MHP Wind Direction Plot 5/3/2021-Flight 2

Figure 77 is the wind angle derived from the MHP plotted against the Trisonica angle. There are sections where the probe and Trisonica align well.

Table 16: Flight 2 Data Analysis

	Trisonica	Multi-hole Probe
Average Angle (deg)	323.4	307.7
Variance	137	322
Skewness	0.24	-1.04
Standard Deviation	11.71	17.94
Kurtosis	2.62	3.74

Table 17: Flight 2 Sensor Data Compare

	Correlation Coefficient	RMSE (m/s)
Probe Trisonica	0.41	23.15

Table 16 and 17 are the data from the MHP comparison to the Trisonica. In general they get the same cardinal direction for the wind. A specific angle would not be able to be confirmed from this data.

Wind Square Flights

Winds squares are flown with the Nano Talon. These flight test were performed with the Trisonica on the western side of the square near the midpoint. This allows for two different tests. The first is a wind square average. The second is pulling data from the straight next to the Trisonica this will allow for a comparison of non-averaged data.

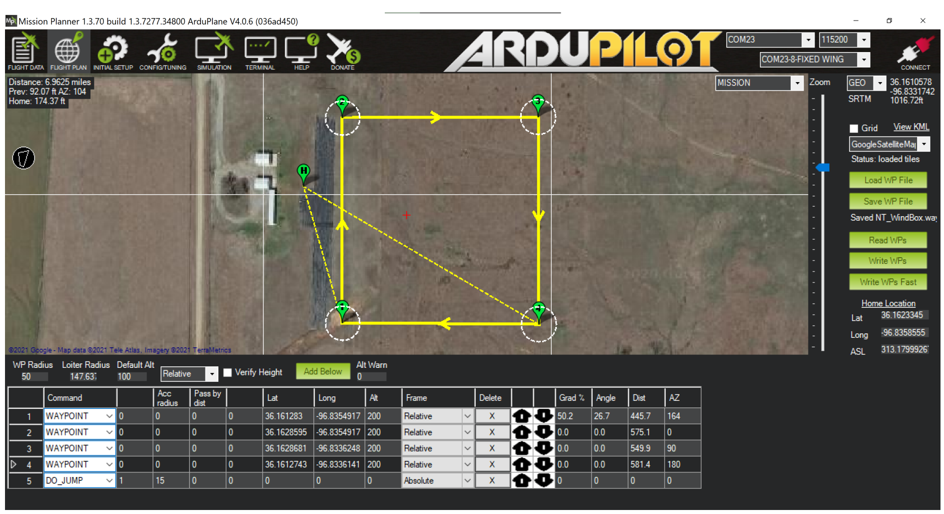


Figure 78: Way-points For Wind Square Calibration Flights Revision For 5-28-2021

Figure 78 are the way-points for the flight. The optimal way point radius for the Nano Talon was around 125ft. This gave the Nano Talon time to full turn and prevented significant overshoots.

Figure 79: Wind Square Flight Path 5-28-2021

Figure 80 is the flight path of the Nano Talon. The red dot on the west side is the location of the SKB-1000 with the Trisonica. Both vehicles were positioned at the same height. The SKB-1000 was left to hover while the Nano Talon flow the square.

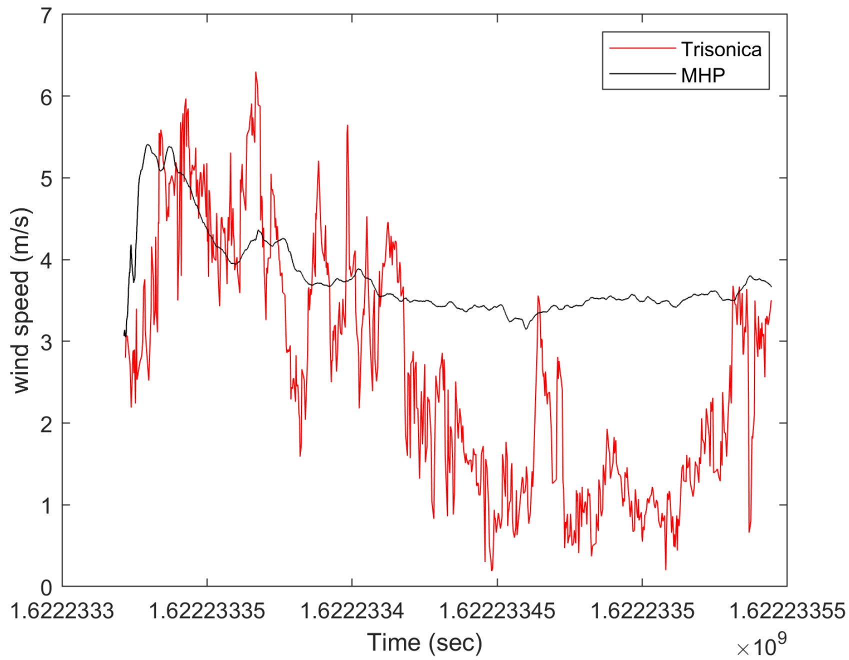


Figure 80: Wind Speed Comparisons 5-28-2021

The conditions for the flights were very low. The winds were variable ranging between 0-6 m/s. This provided a good low wind test opportunity. Due to the wind speed being significantly lower than the aircraft speed any system errors are magnified. This makes the aircraft state removal more significant and will highlight any errors.

Table 18: 5-28 Trisonic MHP Wind Square Data Analysis

	Trisonica	Multi-hole Probe
Average Speed (m/s)	2.69	3.77
Variance	2.2	0.26
Skewness	0.31	1.71
Standard Deviation	1.48	0.51
Kurtosis	0.18	0.24

The Trisonica does show high standard deviation and variances indicating gusts. The MHP does read a much higher speed than the Trisonica. Across multiple flight test the when the wind drops below 3m/s the MHP measurements stall and stop lowering. The cause of this has not been isolated and will need further investigation.

Table 19: 5-28-21 Trisonica MHP Wind Square Comparison

	Correlation Coefficient	RMSE (m/s)
Probe Trisonica	0.69	1.62

The wind square shows very strong correlation to the Trisonica data. This may be attributed to the reduced averaging of the data in the processing of wind squares.

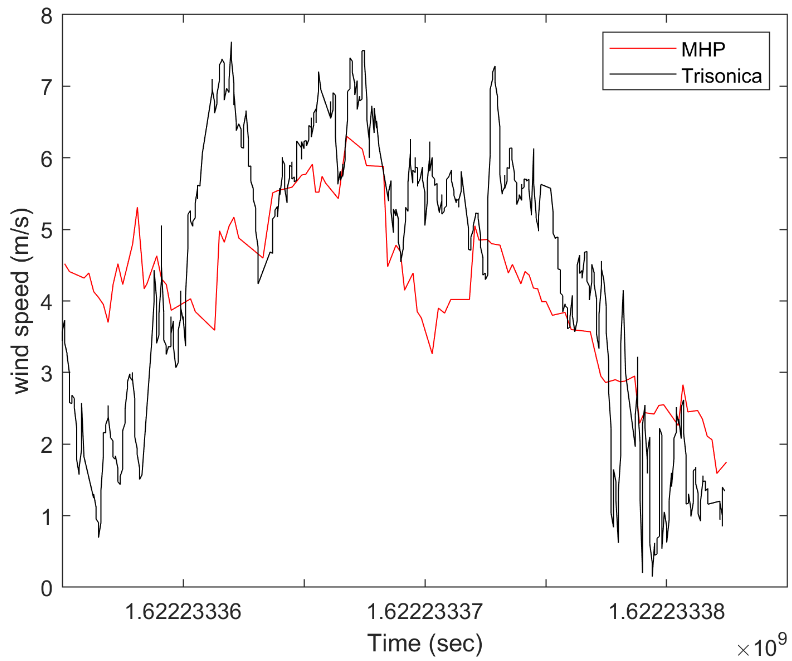


Figure 81: Straight Section Wind Speed Comparisons 5-28-2021

The wind square straights allow for comparison of non averaged MHP wind data. Since the probe is traveling in steady level flight there is no turns that need to be averaged out. Figure 81 is one of these straight sections.

Table 20: 5-28-21 Trisonica MHP Wind Square Steady Level Analysis

	Trisonica	Multi-hole Probe
Average Speed (m/s)	4.13	4.38
Variance	1.19	1.93
Skewness	-0.26	-0.44
Standard Deviation	1.09	1.93
Kurtosis	2.49	1.98

The analysis of this data provides a good insight to the probes ability to measure micro scale atmospheric phenomena. Since the gust are not averaged out in this data it has the closest comparisons to the Trisonica measurements.

Table 21: 5-28-21 Trisonica MHP Wind Square Steady Level Sensor Compare

	Correlation Coefficient	RMSE (m/s)
Probe Trisonica	0.70	1.47

The steady level comparison shows a very strong correlation between data sets. The root mean square error still remains large.

5.1.6 Transect Flights

On June 10th 2021 the Nimbus was used to fly transects at the Choctaw Flight Field near Daisy Oklahoma. The terrain in the area is hilly with dense forest. The flights were flown at 200ft AGL. With a 2 mile straight line flight path. The Young 92000 was posted on a 40ft tower mid flight path.

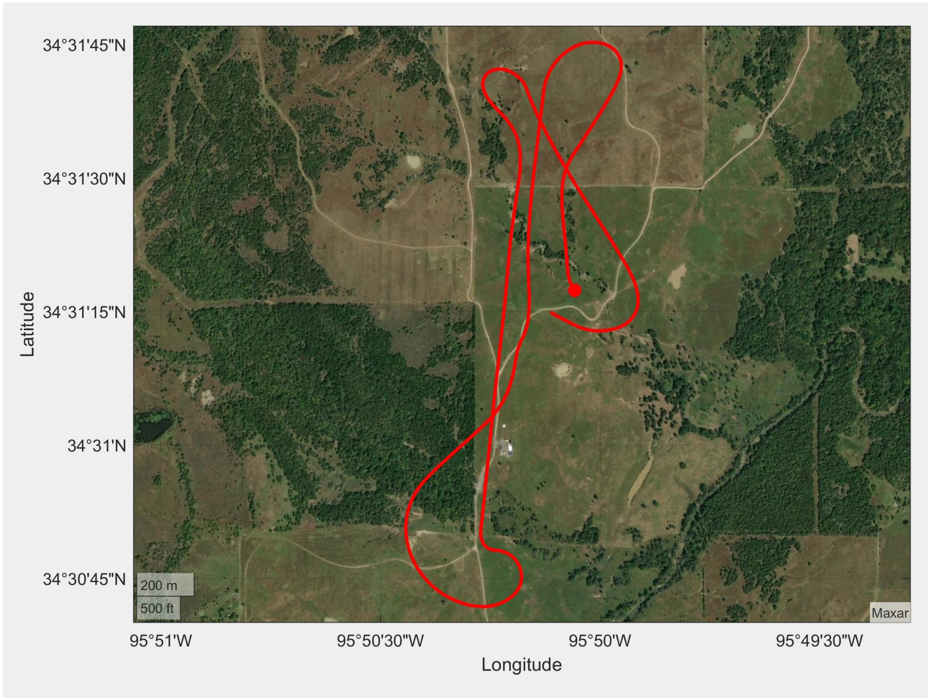


Figure 82: Choctaw Flight Path

Transects, similar to the wind square can provide a good opportunity for for direct comparison between probes since no averaging is required. Figure 82 is the flight path of the nimbus. The Nimbus was able to hold a straight line for a long period. Holding quality steady level flight can be key to good data.

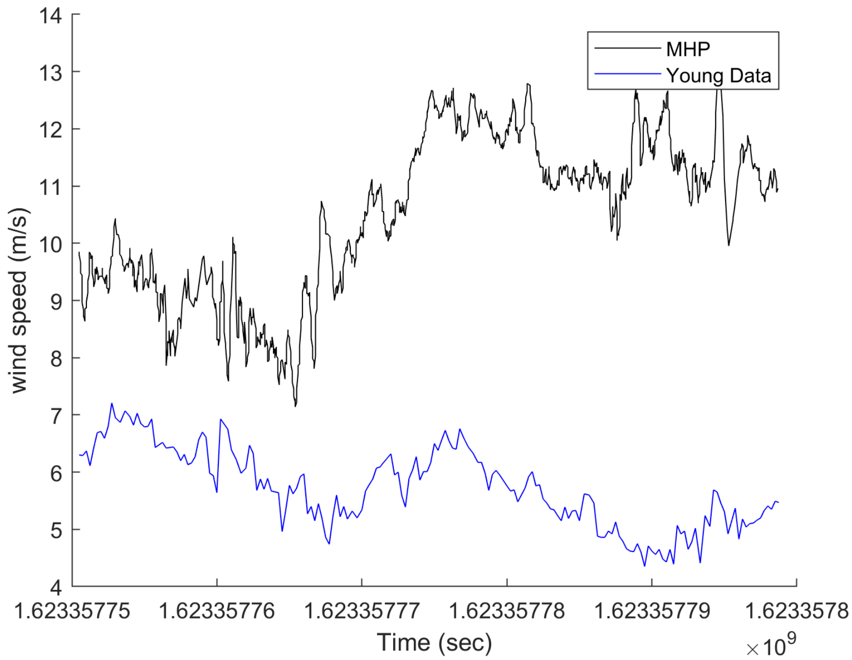


Figure 83: Choctaw Wind Plots

Figure 83 shows the comparison between MHP and the Young. This data snippet is taken from when the Nimbus is flying over the tower during steady level flight.

Table 22: Choctaw Transects Steady Level Upwind Analysis

	Young	Multi-hole Probe
Average Speed (m/s)	5.72	10.49
Variance	0.47	1.91
Skewness	0.037	-0.35
Standard Deviation	0.69	1.38
Kurtosis	2.1	2.06

As expected the average wind speed observed by the probe is higher than the Young due to the altitude change.

Table 23: Choctaw Transects Steady Level Upwind Sensor Compare

	Correlation Coefficient	RMSE (m/s)
Probe Young	0.312	5.862

The correlation of the two data sets does show moderate correlation.

5.2 Use For a Holistic Meteorological Application

One of the major motivations for a cost effective MHP system that can be integrated with a UAS is for weather monitoring and research. A valuable tool as discussed in the Chapter 2, the literature review, are atmospheric soundings. UAS can assist in low level soundings to supplement data in areas that are under sampled by balloons.

Weather Front Flights (5/27/21)

The flights used for the weather soundings are the 5/27/21 flights at the Oklahoma state Unmanned Aircraft Flight Station. The flights were flown on a day with a high severe weather potential. In this section the multi-hole probe measurements are combined with temperature, pressure, and humidity data from I-met XQ2 sensors. These flights were flown with the Nimbus, Nano Talon, and SKB-1000. The SKB-1000 flew vertical profiles while the Nano Talon did orbits around it. The Nimbus performed transects down range.

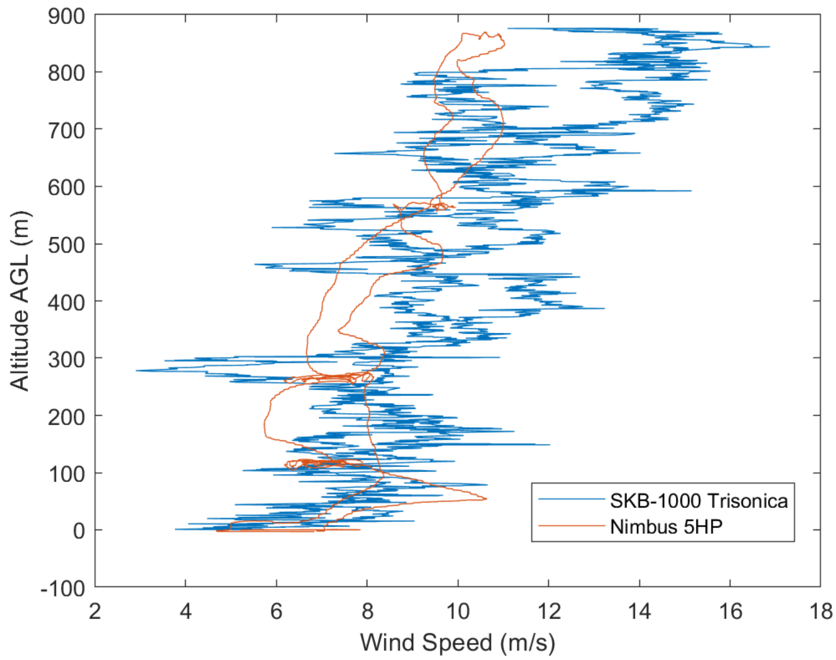


Figure 84: Wind Speed Sounding

Figure 86 is the the MHP and Trisonica wind speeds plotted against altitude. This is a good visualization of the atmospheric boundary layer.

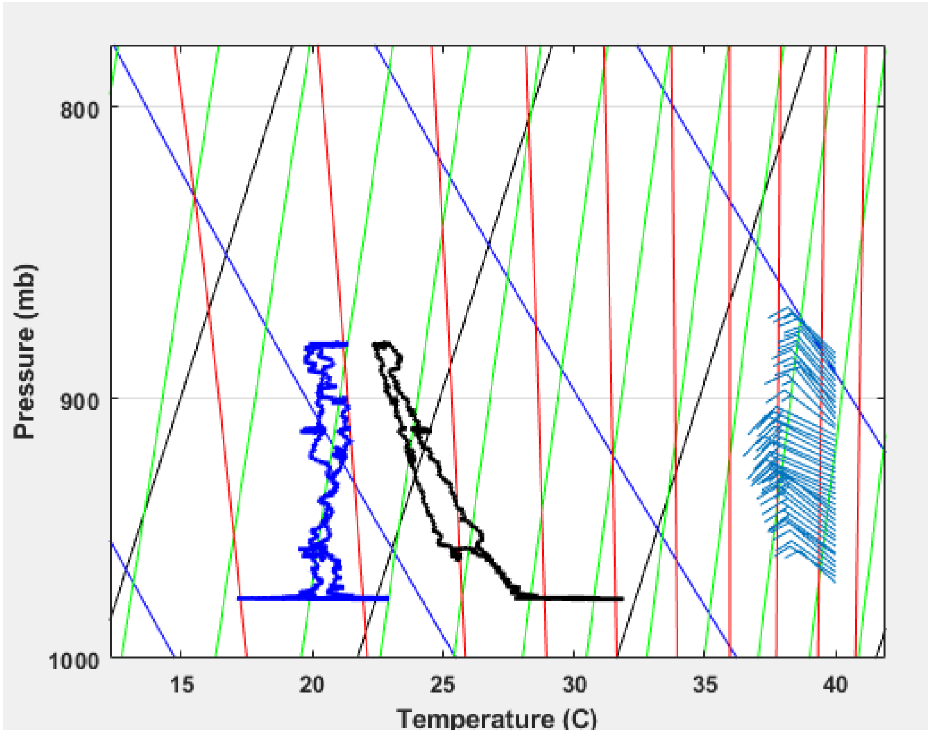


Figure 85: Sounding

Figure 85 is a sounding created from the MHP and I-met data. This data can be shared with meteorologist to help forecast. This holistic application is a demonstration of what the probe can be used for. The blue and black lines are the temperature profiles with the blue barbs on the right side corresponding to wind speed and direction.

CHAPTER VI

CONCLUSION

6.1 Five Hole Probe Comparison to other sensors

The results from the calibrations can now be compared to other known systems. The RMSE and the calculated Gaussian error as calculated in Chapter 5 are compared to the aircraft from Table 3.

Table 24: System Comparison

Vehicle	Probe Type	Speed Accuracy (m/s)	Accuracy Method
M2AV	5HP	0.5	1σ
Blackswift S0	5HP	0.48	N/A
ALADINA	5HP	0.5	Gaussian error
wind RPA	5HP	1.1	Gaussian error
Manta	5HP	0.045	1σ
Scan Eagle	5HP	0.045	1σ
Nano Talon	5HP	0.85	Gaussian error
Nimbus	5HP	1.26	Gaussian error
Nimbus/Nano Talon	5HP	1.12	RMSE

Table 24 is a comparison between the two different probe accuracy methods and other multi-hole probe systems. The Gaussian error method and the RMSE from flight tests show similar accuracies. While the accuracy is not on the level desired for atmospheric measurements it has reached a level that is on par with other systems. The root mean square error is an average of the root mean square over all of the comparison flights between the Trisonica and probe. The final error of the system is around 1.12m/s though this value can vary based on conditions and flight profile. It is important to note that Gaussian Error propagation gives a good baseline for the uncertainty of the system and is not an accuracy. Averaging the two systems together gives an uncertainty of ± 1.05.

6.2 Probe Challenges

One of the main goals of this project was to use a fully additive manufactured probe for UAS measurements. As the 3D printed probe was implemented it was discovered that it was not suited for the rigors of flight operations. The issues with the additive manufactured probes was not an accuracy or design issue. The main issue was a material issue. Finding a material and printer combination that can reach the desired strength and durability while maintaining the required accuracy will be a challenge.

Multiple different materials have been attempted for this project. The two Formlabs materials were most reliable for have clear passage of air through the tubes, as discussed this took extra work to ensure. Specially ordered Nylon prints all had clogged holes, although these prints showed potential for able to be cleared as it was not a total blockage.

A hybrid probe is a good solution until a reliable printer/material selection can be found. The hybrid probe still achieves the goal of keeping costs down. While it does take more man hours to manufacture that cost would most likely be less than having to re-calibrate after breaking a 3D printed probe.

The single most important influence to a probe's accuracy is the quality of the sensors used for vehicle state estimation. This is illustrated well by the Scan Eagle and Manta in table 3. These two systems are able to achieve significantly better accuracies than other systems because of the more accurate state estimation. If cost is the primary objective Pix-hawks are suitable for accurate average wind measurements. If more detailed measurement is needed using a more accurate system to account for aircraft dynamics will be required.

6.3 Future Work

This project can serve as the foundation for different endeavors. One of the major issues that still linger is finding the best material to print the probe out of. Based off the findings of field tests a hybrid probe is still recommended. While the body of the probe and tip will work well 3D printed with the right material, finding a material strong enough to make the tube attachment points will be difficult.

Re-evaluation of the pressure sensors and data acquisition system should be conducted. The flight speed of the vehicles does not encompass much of the ±1psi range of the pressure sensors. A sensor with reduced range would reduce the quantization error and be better suited for this application. The Teensy based data acquisition system has some inherit limitations. A system with a better ADC converter and higher clock rates could be used to increase the resolution and frequency of the data. A properly designed noise filter paired with the system would also help improve results.

One of the major shortcomings of this project has been wind angle accuracy. The wind angle measurements should be further evaluated and refined. The probe should be flown against more known data sources to evaluate this such as tall weather towers. For soundings and ABL research, a higher altitude limit would allow for a more complete data set. A higher altitude waiver for UAS weather operations should be pursued.

A more holistic system design should also be pursued. Improvements like integration of Pixhawk error and variance logs to flag bad data should be investigated. A method to make the identification of steady level flight a non manual process would also improve the ease of data processing. As the system moves beyond the prototype phase better integration with the aircraft should be done. Many of the hardware issues could be avoided if the board and pitot tube had better integration.

6.4 Conclusion

With preliminary flight testing and calibration of the probe done some performance measures can be presented. While these should not be taken as true for every flight due to the variability mounting and atmospheric conditions the parameters can serve as a good baseline.

Table 25: Preliminary System Characteristics

Accuracy (m/s)	1.12
Uncertainty (m/s)	1.05
Response Time (ms)	20
Max Frequency (Hz)	50

The MHP system developed is a good starting point for a reliable and accurate system for atmospheric research. Table 25 shows the preliminary characteristics of the 5HP system. While further analysis of the probe will need to be performed the results from this paper demonstrates a cheap fixed wing wind sensing platform. While challenges with a fully 3D manufactured probe were found the use of one is presented to be feasible. An alternative hybrid probe design that followed the same sizing of the 3D manufactured one was also developed. The probes tend to be of similar accuracy of other probes developed. The probes are also demonstrated in taking data that can support meteorological objectives such as ABL research.

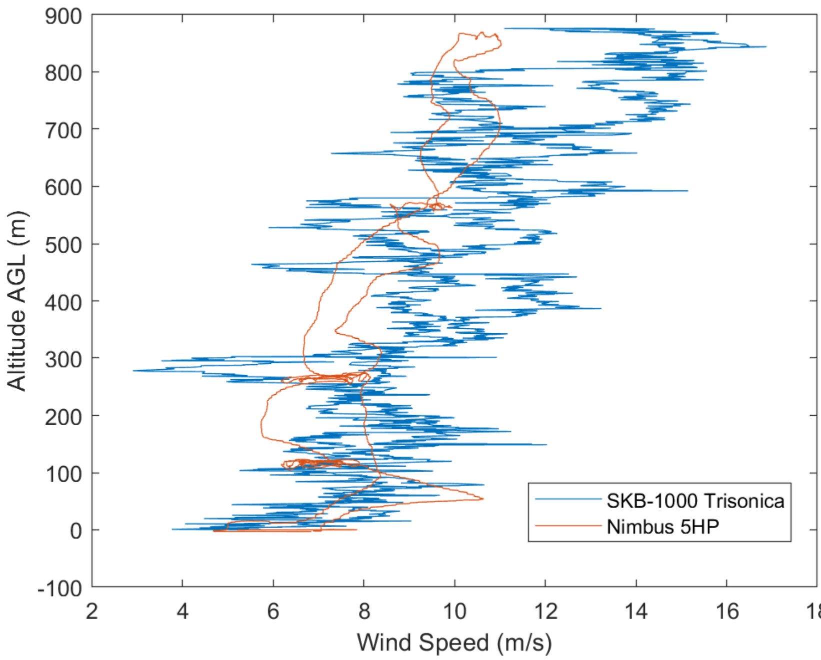


Figure 86: ABL Wind Sounding

REFERENCES

[1] *Measuring the wind vector using the autonomous mini aerial vehicle M2 AV*, Journal of Atmospheric and Oceanic Technology **25** (2008), no. 11, 1969–1982.

[2] *Safe autonomous flight environment (SAFE50) for the notional last "50 ft" of operation of "55 lb" class of UAS*, AIAA Information Systems-AIAA Infotech at Aerospace, 2017 (2017), no. January, 1–7.

[3] *Features: 3d ultrasonic anemometer: Trisonica mini: Wind sensor*, `https://anemoment.com/features/`, May 2021.

[4] *fft*, `https://www.mathworks.com/help/matlab/ref/fft.html`, May 2021.

[5] *Instructions: Responseone model 92000 weather transmitter*, `https://www.fondriest.com/pdf/rm_young_92000_manual.pdf`, May 2021.

[6] *Instructions: Wind monitor model 05103*, `https://www.youngusa.com/wp-content/uploads/2021/02/05103-90N.pdf`, May 2021.

[7] *Material data sheet: Durable*, `https://formlabs-media.formlabs.com/datasheets/1801084-TDS-ENUS-0P.pdf`, May 2021.

[8] *Rawinsondes*, `https://www.sciencedirect.com/topics/earth-and-planetary-sciences/rawinsondes`, May 2021.

[9] *S0 data sheet_v2*, `https://www.https:bst.aero/wp-content/uploads/2021/06/VTOL-S0-Data-Sheet.pdf`, May 2021.

[10] *Skew-t parameters*, `https://www.weather.gov/source/zhu/ZHU_Training_Page/convective_parameters/skewt/skewtinfo.html`, May 2021.

[11] Loiy Al-Ghussain and Sean Bailey, *An Approach to Minimize Aircraft Motion Bias in Multi-Hole Probe Wind Measurements made by Small Unmanned Aerial Systems*, Atmospheric Measurement Techniques Discussions (2020), 1–19.

[12] 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** (1984), no. 5, 384–387.

[13] D. N. Axford, *On the accuracy of wind measurements using an inertial platform in an aircraft, and an example of a measurement of the vertical mesostructure of the atmosphere*, Journal of Applied Meteorology **7** (1968), no. 4, 645–666.

[14] Solmoz K Azertash-Namin, *Evaluation of low-cost multi-hole probes for atmospheric boundary layer investigation*, Masters Thesis, Oklahoma State University (2017).

[15] Lindsay Barbieri, Stephan T. Kral, Sean C.C. Bailey, Amy E. Frazier, Jamey D. Jacob, Joachim Reuder, David Brus, Phillip B. Chilson, Christopher Crick, Carrick Detweiler, Abhiram Doddi, Jack Elston, Hosein Foroutan, Javier González-Rocha, Brian R. Greene, Marcelo I. Guzman, Adam L. Houston, Ashraful Islam, Osku Kemppinen, Dale Lawrence, Elizabeth A. Pillar-Little, Shane D. Ross, Michael P. Sama, David G. Schmale, Travis J. Schuyler, Ajay Shankar, Suzanne W. Smith, Sean Waugh, Cory Dixon, Steve Borenstein, and Gijs De Boer, *Intercomparison of small unmanned aircraft system (sUAS) measurements for atmospheric science during the LAPSE-RATE campaign*, Sensors (Switzerland) **19** (2019), no. 9.

[16] Konrad Bärfuss, Falk Pätzold, Barbara Altstädter, Endres Kathe, Stefan Nowak, Lutz Bretschneider, Ulf Bestmann, and Astrid Lampert, *New setup of the UAS ALADINA for measuring boundary layer properties, atmospheric particles and solar radiation*, Atmosphere **9** (2018), no. 1.

[17] Radiance Calmer, Gregory C. Roberts, Jana Preissler, Kevin J. Sanchez, Solène Derrien, and Colin O'Dowd, *Vertical wind velocity measurements using a five-hole probe with remotely piloted aircraft to study aerosol-cloud interactions*, Atmospheric Measurement Techniques **11** (2018), no. 5, 2583–2599.

[18] Genevieve Comte-bellot, *Hot-wire anemometry*, (1976), no. Thomas 1920, 209–231.

[19] 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, p. 2986.

[20] Thomas J Dudzinski and Lloyd N Krause, *Flow-direction measurement with fixed-position probes*, National Aeronautics and Space Administration, 1969.

[21] K. E. Garman, K. A. Hill, P. Wyss, M. Carlsen, J. R. Zimmerman, B. H. Stirm, T. Q. Carney, R. Santini, and P. B. Shepson, *An airborne and wind tunnel evaluation of a wind turbulence measurement system for aircraft-based flux measurements*, Journal of Atmospheric and Oceanic Technology **23** (2006), no. 12, 1696–1708.

[22] G David Huffman, DC Rabe, and ND Poti, *Flow direction probes from a theoretical and experimental point of view*, Journal of Physics E: Scientific Instruments **13** (1980), no. 7, 751.

[23] Jamey D. Jacob, Phillip B. Chilson, Adam L. Houston, and Suzanne Weaver Smith, *Considerations for atmospheric measurements with small unmanned aircraft systems*, Atmosphere **9** (2018), no. 7, 1–16.

[24] T. G. Konrad, M. L. Hill, J. R. Rowland, and J. H. Meyer, *A Small, Radio-Controlled Aircraft As A Platform for Meteorlogical Sensors*, APL Technical Digest (1970), no. December, 11–19.

[25] L. Kristensen, *The cup anemometer and other exciting instruments*, J. Appl. Meteorol. J. Atmos. Oceanic Technol. J. Appl. Meteorol. J. Atmos. Oceanic Technol **615** (1993), no. April, 1–83.

[26] L Kristensen, *Cups , props and vanes*, **766** (1994), no. August, 1–37.

[27] D H Lenschow, *Bulletin No . 23 MEASUREMENT TECHNIQUES: AIR MOTION SENSING*, (1989), no. 23, 1–29.

[28] Stephanie Mayer, Gautier Hattenberger, Pascal Brisset, Marius Jonassen, and Joachim Reuder, *A 'no-flow-sensor' wind estimation algorithm for unmanned aerial systems*, International Journal of Micro Air Vehicles **4** (2012), no. 1, 15–29.

[29] Renee A. McPherson, Christopher A. Fiebrich, Kenneth C. Crawford, Ronald L. Elliott, James R. Kilby, David L. Grimsley, Janet E. Martinez, Jeffrey B. Basara, Bradley G. Illston, Dale A. Morris, Kevin A. Kloesel, Stephen J. Stadler, Andrea D. Melvin, Albert J. Sutherland, Himanshu Shrivastava, J. D. Carlson, J. Michael Wolfinbarger, Jared P. Bostic, and David B. Demko, *Statewide monitoring of the mesoscale environment: A technical update on the Oklahoma Mesonet*, Journal of Atmospheric and Oceanic Technology **24** (2007), no. 3, 301–321.

[30] Tevis Nichols, Brian Argrow, and Derek Kingston, *Error sensitivity analysis of small UAS wind-sensing systems*, AIAA Information Systems-AIAA Infotech at Aerospace, 2017 (2017), no. January, 1–15.

[31] C. F.R. Nowack, *Improved calibration method for a five-hole spherical Pitot probe*, Journal of Physics E: Scientific Instruments **3** (1970), no. 1, 21–26.

[32] S. Prudden, A. Fisher, M. Marino, A. Mohamed, S. Watkins, and G Wild, *Measuring wind with Small Unmanned Aircraft Systems*, Journal of Wind Engineering and Industrial Aerodynamics **176** (2018), no. December 2017, 197–210.

[33] Alexander Rautenberg, Martin S. Graf, Norman Wildmann, Andreas Platis, and Jens Bange, *Reviewing wind measurement approaches for fixed-wing unmanned aircraft*, Atmosphere **9** (2018), no. 11, 1–24.

[34] Benjamin D. Reineman, Luc Lenain, Nicholas M. Statom, and Kendall K. Melville, *Development and testing of instrumentation for UAV-based flux measurements within terrestrial and marine atmospheric boundary layers*, Journal of Atmospheric and Oceanic Technology **30** (2013), no. 7, 1295–1319.

[35] N. Sitaram and K. Srikanth, *Effect of chamfer angle on the calibration curves of the five hole probes*, International Journal of Rotating Machinery (2014).

[36] Y. Telionis, D. Yang and O. Rediniotis, *Recent developments in multi-hole probe (mhp) technology*, International Congress of Mechanical Engineering (2009).

[37] A. L. Treaster and A. M. Yocum, *The calibration and application of five-hole probes*, ISA transactions (1979).

[38] NOAA US Department of Commerce, *Weather balloon / upper air observations*, `https://www.weather.gov/chs/upperair`, Oct 2018.

[39] M Zeiger, L Chalmeta, and D Telionis, *Tip geometry effects on calibration and performance of seven-hole probes*, 29th AIAA, Fluid Dynamics Conference, 1998, p. 2810.

APPENDICES

All codes used can be found on Github: https://github.com/LeviRoss2/USRI_Sensor_Suite

Codes

```
% Pixhawk DFL Analyzer V4.8.10
% Created by:    Levi Ross | levi.ross@okstate.edu
% Edited by:     Kyle Hickman | kthickm@okstate.edu
% Unmanned Systems Research Institute
% Creation Date: 11/17/2019
% Last Modified: 06/20/2021 - Kyle
%
% New features:
%     * Replay _Parsed files
%     * No longer need to set Ardupilot type, auto-selected from MSG1 log
%     * Added Wind Rose to view estimated wind speed data form Pixhawk
%     * Added PlotMergedData codeset, allowing for simultaneous or series
%      data plots
%     * Added custom .bin -> .mat converter, eliminating the need for
%   Mission Planner
%     * Complete CopterSonde support
%     * Added met data averaging for vertical profiles or stationary hovers
%     * Added Animation of side-by-side T, P, and H graphs against GPS data
%     * 3D-isometric viewing of Multi-Aircraft animation
```

```
%    * Added altitude parsing to allow for multi-aircraft 3D plots
%    * Brought in NKF1 parameters for GPS NED velocities and aircraft
%    attitude states
%    * 5HP data gets averaged down to Kalman Filter Rates
%    * 5HP data has aircraft states removed for 3D wind vector
%    * Added plots for Corrected 5HP data
%    * Added an FFT function and Option for 5HP data analysis
%    * Added ability to average 5HP data for windspeed (period based off
%     FFT peak, must be selected)
%    * Fully implemented 5hp algorithm
%
% General Updates:
%    * Consolidated DFL_New and DFL_Old 5HP plots and output file saving
%
% Bug Fixes:
%    * Properly referencing Temp and Humidty values in Sensor Suite data
%    * Explicitly defined all figures so there's no chance to overwrite
%    * Fixed Figure 5 plot (changed from CTUN(:,2) to 3)
%    * Temperature 1 is Plotting time v time - Fixed
%    * Added animVid initialization for multi vehicle animation (Fixed
%    Triple animate)
%    * Fixed NKF2 issues
%
% General Notes/Unresolved Bugs:
%    * Quadcopter NKF1 imports untested
%    * Code Asks for 5HP when just TPH is "Yes"
%    * Imet Parser date time column call is wrong
%
% Projects In Progress/To Be Done:
%    * Removing IMU states from 5hp data
%    * Include Anemometer Parser Options
%    * Bring in Mesonet Data and/or other ground sources
```

```
%     * Comparison between multiple data sources
%
% Current Fig Count:
%     * Single: 16
%     * Multi: 1

%% Clear All Data
close all
clear all
clc

%% Initialize User-Defined Content
display('Initializing user-defined variable space.');

thrMinPWM = 1100;               % Default, scales THR% plots
thrMaxPWM = 1900;               % Default, scales THR% plots

Single_Multi = 'Single';

% Single Controls
tripleAnimateTPH = 'No';        % Side-by-side T, P, and H plots against ...
    GPS position
graphToggle = 'No';            % Show plots of parsed data
animateToggle = 'No';           % Flight animation based on GPS and Alt(AGL)
recordAnimation = 'No';         % Record animation for future playback
hoverProfile = 'None';          % Profile (vertical) or Hover ...
    (stationary) averaging
stateSpace = 'No';              % Pixhawk recorded SS variable output ...
    (new file)
iMetValue = 'No';               % Load iMet data (XQ1 - small | XQ2 - large)
mhpValue = 'Yes';               % Load 5HP/TPH data (use 5HP)
```

```
mhpAvg = 'Yes';                    % Lets user set averaging bounds for ...
    wind speed averages (not implemented)
tphValue = 'No';                  % Load 5HP/TPH data (use TPH)
overlay = 'No';                   % Show iMet & 5HP/TPH alongside Pixhawk ...
    data plots
gpsOut = 'No';                     % Output GPS data as individual file ...
    (lat, long, alt)
sensorOut = 'No';                  % Output parsed sensor data (iMet ...
    seperate from 5HP/TPH)
attitudeOut = 'No';                % Output parsed attitude data
sensorCompare = 'No';             % Show iMet, 5HP/TPH, and Pixhawk ...
    atmoshperic sensors on same plots
dflNewOld = 'Unknown';
arduPilotType = 'Default';
pitchToggle = 'No';                % TIA-Specific
throttleToggle = 'No';             % TIA-Specific
aircraft = 'N/A';                  % TIA-Specific
tvToggle = 'No';                   % TIA-Specific
indvToggle = 'No';                 % TIA-Specific

% Only used for Single Animation
animateSpeed = 2;                 % Overall speed
animateHeadSize = 3;               % Icon size
animateTailWidth = 2;              % Width of tail
animateTailLength = 10000;           % Length of tail
animateFPS = 30;
animateHeadSize = animateHeadSize + 5;
plotTitle = '';

% Only used for Multi Animation
animation = 'Yes';          % Default: Yes | Yes: turn on animation | No: ...
    turn Off
```

```
simultaneous = 'Yes';      % Default: Yes | Yes: Files in same time ...
    frame | No: Files in concurrent time frames
isometric = 'No';         % Default: Yes | Yes: view in 3D space | No: ...
    view in 2D with Satellite map overlay
azimuthAngle = 15;         % Default: 45 | Rotation angle in x-y plane ...
    to view the 3D plot (positive values -> CCW rotation, negative CW)
elevationAngle = 45;       % Default: 15 | Elevation angle above the X-Y ...
    plane (90 is top down X-Y plot, 0 is looking from the X-Y plane ...
    depending on azimuthAngle)
iterationSkip = 5;        % Default: 5  | High numbers: increase ...
    animation speed, reduce smoothness. If increasing, decrease ...
    animFrameRate to have useful video time
animFrameRate = 30;        % Default: 30 | High numbers: increase ...
    animation speed, increase smoothness. If increasing, decrease ...
    iteration_skip to have useful video time

%% Main Code
display('Executing user-defined operations.');
if(strcmpi(Single_Multi,'Single'))
    %% Gather All Files For Parsing

    % Have user browse for a file, from a specified "starting folder."
    % For convenience in browsing, set a starting folder from which to ...
        browse.
    % Start in the current folder.
    startingFolderDFL = pwd;
    % Get the name of the file that the user wants to use.
    defaultFileNameDFL = fullfile(startingFolderDFL,{'*.bin;*.mat'});
    [baseFileNameDFL, folderDFL] = uigetfile(defaultFileNameDFL, ...
        'Select a Pixhawk DFL file (.bin or .mat only)');
    startingFolderDFL = folderDFL;
    if baseFileNameDFL == 0
```

```
        % User clicked the Cancel button.
        return;
    end

    fullInputMatFileNameDFL = fullfile(folderDFL, baseFileNameDFL);
    %% Pixhawk bin->mat Converter
    if(strcmpi(baseFileNameDFL((end-2):end),'bin'))
        DFL_NewOld = 'New';

        rawDFL = Ardupilog(fullInputMatFileNameDFL);
        %% Check if CopterSonde or Standard DFL file
        Rover = 0;
        Copter = 0;
        QuadPlane = 0;
        Plane = 0;

        %% ArduPilotType Parser (MSG1)
        for i=1:length(rawDFL.MSG.LineNo);
            MSG1{1,i}{1,1} = 'MSG';
            MSG1{1,i}{2,1} = rawDFL.MSG.TimeUS(i);
            MSG1{1,i}{3,1} = rawDFL.MSG.Message(...
                i,1:length(rawDFL.MSG.Message(1,:)));
            if(contains(rawDFL.MSG.Message(i,1:length(...
                    rawDFL.MSG.Message(1,:))),'ArduCopter'))
                Copter = 1;
            elseif(contains(rawDFL.MSG.Message(i,1:length(...
                    rawDFL.MSG.Message(1,:))),'QuadPlane'))
                QuadPlane = 1;
            elseif(contains(rawDFL.MSG.Message(i,1:length(...
                    rawDFL.MSG.Message(1,:))),'ArduRover'))
                Rover = 1;
            elseif(contains(rawDFL.MSG.Message(i,1:length(...
```

```
                rawDFL.MSG.Message(1,:))),'ArduPlane'))
            Plane = 1;
        end
    end

    if(Rover == 1)
        arduPilotType = 'ArduRover';
    elseif(Copter == 1)
        arduPilotType = 'ArduCopter';
    elseif(QuadPlane == 1)
        arduPilotType = 'QuadPlane';
    elseif(Plane == 1)
        arduPilotType = 'FixedWing';
    end

    if(strcmpi(arduPilotType,'FixedWing') |...
            strcmpi(arduPilotType,'QuadPlane'))

        %% GPS
        GPS(:,1) = rawDFL.GPS.LineNo;
        GPS(:,2) = ...
            fix(rawDFL.GPS.TimeS/rawDFL.GPS.fieldMultipliers.TimeUS);
        GPS(:,3) = rawDFL.GPS.Status;
        GPS(:,4) = rawDFL.GPS.GMS;
        GPS(:,5) = rawDFL.GPS.GWk;
        GPS(:,6) = rawDFL.GPS.NSats;
        GPS(:,7) = rawDFL.GPS.HDop;
        GPS(:,8) = rawDFL.GPS.Lat;
        GPS(:,9) = rawDFL.GPS.Lng;
        GPS(:,10) = rawDFL.GPS.Alt;
        GPS(:,11) = rawDFL.GPS.Spd;
        GPS(:,12) = rawDFL.GPS.GCrs;
```

```
        GPS(:,13) = rawDFL.GPS.VZ;
        GPS(:,14) = rawDFL.GPS.Yaw;
        GPS(:,15) = rawDFL.GPS.U;
        GPS(:,16) = rawDFL.GPS.DatenumUTC;
        GPS_label = {'LineNo';'TimeUS';'Status';'GMS';'GWk';...
            'NSats';'HDop';'Lat';'Lng';'Alt';'Spd';'GCrs';...
            'VZ';'Yaw';'U';'DatenumUTC'};
        %% ATT
        ATT(:,1) = rawDFL.ATT.LineNo;
        ATT(:,2) = fix(...
            rawDFL.ATT.TimeS/rawDFL.ATT.fieldMultipliers.TimeUS);
        ATT(:,3) = rawDFL.ATT.DesRoll;
        ATT(:,4) = rawDFL.ATT.Roll;
        ATT(:,5) = rawDFL.ATT.DesPitch;
        ATT(:,6) = rawDFL.ATT.Pitch;
        ATT(:,7) = rawDFL.ATT.DesYaw;
        ATT(:,8) = rawDFL.ATT.Yaw;
        ATT(:,9) = rawDFL.ATT.ErrRP;
        ATT(:,10) = rawDFL.ATT.ErrYaw;
        ATT(:,11) = rawDFL.ATT.DatenumUTC;
        ATT_label = {'LineNo';'TimeUS';'DesRoll';'Roll';'DesPitch';...
            'Pitch';'DesYaw';'Yaw';'ErrRP';'ErrYaw';'DatenumUTC'};
        %% BARO
        BARO(:,1) = rawDFL.BARO.LineNo;
        BARO(:,2) = fix(...
            rawDFL.BARO.TimeS/rawDFL.BARO.fieldMultipliers.TimeUS);
        BARO(:,3) = rawDFL.BARO.Alt;
        BARO(:,4) = rawDFL.BARO.Press;
        BARO(:,5) = rawDFL.BARO.Temp;
        BARO(:,6) = rawDFL.BARO.CRt;
        BARO(:,7) = rawDFL.BARO.SMS;
        BARO(:,8) = rawDFL.BARO.Offset;
```

```
        BARO(:,9) = rawDFL.BARO.GndTemp;
        BARO(:,10) = rawDFL.BARO.Health;
        BARO(:,11) = rawDFL.BARO.DatenumUTC;
        BARO_label = {'LineNo';'TimeUS';'Alt';'Press';'Temp';'CRt';...
            'SMS';'Offset';'GndTemp';'Health';'DatenumUTC'};
        %% CTUN
        CTUN(:,1) = rawDFL.CTUN.LineNo;
        CTUN(:,2) = fix(...
            rawDFL.CTUN.TimeS/rawDFL.CTUN.fieldMultipliers.TimeUS);
        CTUN(:,3) = rawDFL.CTUN.NavRoll;
        CTUN(:,4) = rawDFL.CTUN.Roll;
        CTUN(:,5) = rawDFL.CTUN.NavPitch;
        CTUN(:,6) = rawDFL.CTUN.Pitch;
        CTUN(:,7) = rawDFL.CTUN.ThrOut;
        CTUN(:,8) = rawDFL.CTUN.RdrOut;
        CTUN(:,9) = rawDFL.CTUN.ThrDem;
        CTUN(:,10) = rawDFL.CTUN.Aspd;
        CTUN(:,11) = rawDFL.CTUN.DatenumUTC;
        CTUN_label = {'LineNo';'TimeUS';'NavRoll';'Roll';...
            'NavPitch';'Pitch';'ThrOut';'RdrOut';'ThrDem';...
            'Aspd';'DatenumUTC'};
        %% NKF2
        try
            NKF2(:,1) = rawDFL.NKF2.LineNo;
            NKF2(:,2) = fix(...
                rawDFL.NKF2.TimeS/rawDFL.NKF2.fieldMultipliers.TimeUS);
            NKF2(:,3) = rawDFL.NKF2.AZbias;
            NKF2(:,4) = rawDFL.NKF2.GSX;
            NKF2(:,5) = rawDFL.NKF2.GSY;
            NKF2(:,6) = rawDFL.NKF2.GSZ;
            NKF2(:,7) = rawDFL.NKF2.VWN;
            NKF2(:,8) = rawDFL.NKF2.VWE;
```

```
            NKF2(:,9) = rawDFL.NKF2.MN;
            NKF2(:,10) = rawDFL.NKF2.ME;
            NKF2(:,11) = rawDFL.NKF2.MD;
            NKF2(:,12) = rawDFL.NKF2.MX;
            NKF2(:,13) = rawDFL.NKF2.MY;
            NKF2(:,14) = rawDFL.NKF2.MZ;
            NKF2(:,15) = rawDFL.NKF2.MI;
            NKF2(:,16) = rawDFL.NKF2.DatenumUTC;
            NKF2_label = {'LineNo';'TimeUS';'AZbias';'GSX';'GSY';...
                'GSZ';'VWN';'VWE';'MN';'ME';'MD';'MX';'MY';'MZ';...
                'MI';'DatenumUTC'};

        catch
            NKF2_label = {'LineNo';'TimeUS';'AZbias';'GSX';'GSY';...
                'GSZ';'VWN';'VWE';'MN';'ME';'MD';'MX';'MY';...
                'MZ';'MI';'DatenumUTC'};
        end
        %% XKF2
        try
            XKF2(:,1) = rawDFL.XKF2.LineNo;
            XKF2(:,2) = fix(...
                rawDFL.NKF2.TimeS/rawDFL.XKF2.fieldMultipliers.TimeUS);
            XKF2(:,3) = rawDFL.XKF2.AX;
            XKF2(:,4) = rawDFL.XKF2.AY;
            XKF2(:,5) = rawDFL.XKF2.AZ;
            XKF2(:,6) = rawDFL.XKF2.VWN;
            XKF2(:,7) = rawDFL.XKF2.VWE;
            XKF2(:,8) = rawDFL.XKF2.MN;
            XKF2(:,9) = rawDFL.XKF2.ME;
            XKF2(:,10) = rawDFL.XKF2.MD;
            XKF2(:,11) = rawDFL.XKF2.MX;
            XKF2(:,12) = rawDFL.XKF2.MY;
```

```
        XKF2(:,13) = rawDFL.XKF2.MZ;
        XKF2(:,14) = rawDFL.XKF2.MI;
        XKF2(:,15) = rawDFL.XKF2.DatenumUTC;
        XKF2_label = {'LineNo';'TimeUS';'AX';'AY';'AZ';'VWN';...
            'VWE';'MN';'ME';'MD';'MX';'MY';'MZ';'MI';'DatenumUTC'};

    catch
        XKF2_label = {'LineNo';'TimeUS';'AX';'AY';'AZ';'VWN';...
            'VWE';'MN';'ME';'MD';'MX';'MY';'MZ';'MI';'DatenumUTC'};
    end
    %% RCOU
    RCOU(:,1) = rawDFL.RCOU.LineNo;
    RCOU(:,2) = fix(...
        rawDFL.RCOU.TimeS/rawDFL.RCOU.fieldMultipliers.TimeUS);
    RCOU(:,3) = rawDFL.RCOU.C1;
    RCOU(:,4) = rawDFL.RCOU.C2;
    RCOU(:,5) = rawDFL.RCOU.C3;
    RCOU(:,6) = rawDFL.RCOU.C4;
    RCOU(:,7) = rawDFL.RCOU.C5;
    RCOU(:,8) = rawDFL.RCOU.C6;
    RCOU(:,9) = rawDFL.RCOU.C7;
    RCOU(:,10) = rawDFL.RCOU.C8;
    RCOU(:,11) = rawDFL.RCOU.C9;
    RCOU(:,12) = rawDFL.RCOU.C10;
    RCOU(:,13) = rawDFL.RCOU.C11;
    RCOU(:,14) = rawDFL.RCOU.C12;
    RCOU(:,15) = rawDFL.RCOU.C13;
    RCOU(:,16) = rawDFL.RCOU.C14;
    RCOU(:,17) = rawDFL.RCOU.DatenumUTC;
    RCOU_label = ...
       {'LineNo';'TimeUS';'C1';'C2';'C3';'C4';'C5';'C6';...
        'C7';'C8';'C9';'C10';'C11';'C12';'C13';'C14';'DatenumUTC'};
```

```
%% IMU
IMU(:,1) = rawDFL.IMU.LineNo;
IMU(:,2) = fix(...
    rawDFL.IMU.TimeS/rawDFL.IMU.fieldMultipliers.TimeUS);
IMU(:,3) = rawDFL.IMU.GyrX;
IMU(:,4) = rawDFL.IMU.GyrY;
IMU(:,5) = rawDFL.IMU.GyrZ;
IMU(:,6) = rawDFL.IMU.AccX;
IMU(:,7) = rawDFL.IMU.AccY;
IMU(:,8) = rawDFL.IMU.AccZ;
IMU(:,9) = rawDFL.IMU.EG;
IMU(:,10) = rawDFL.IMU.EA;
IMU(:,11) = rawDFL.IMU.T;
IMU(:,12) = rawDFL.IMU.GH;
IMU(:,13) = rawDFL.IMU.AH;
IMU(:,14) = rawDFL.IMU.GHz;
IMU(:,15) = rawDFL.IMU.AHz;
IMU(:,16) = rawDFL.IMU.DatenumUTC;
IMU_label = {'LineNo';'TimeUS';'GyrX';'GyrY';'GyrZ';'AccX';...
    'AccY';'AccZ';'EG';'EA';'T';'GH';'AH';'GHz';'AHz';..
    'DatenumUTC'};

% Save as new file for integration with normal code
[¬, baseNameNoExtDFL, ¬] = fileparts(baseFileNameDFL);
baseFileName = sprintf('%s.mat', baseNameNoExtDFL);
fullParsedMatFileName = fullfile(folderDFL, baseFileName);
% Save file with parsed data as the original filename plus ...
   the added portion
save(fullParsedMatFileName,'GPS','GPS_label','ATT',...
    'ATT_label','BARO','BARO_label','CTUN','CTUN_label',...
    'NKF2','NKF2_label','XKF2','XKF2_label','IMU',...
    'IMU_label','RCOU','RCOU_label','MSG1');
```

```
        fullInputMatFileNameDFL = fullParsedMatFileName;

    elseif(strcmpi(arduPilotType,'ArduCopter') |...
            strcmpi(arduPilotType,'CopterSonde'))

        try
            rawDFL.IMET;
            arduPilotType = 'CopterSonde';
            %% WIND
            WIND(:,1) = rawDFL.WIND.LineNo;
            WIND(:,2) = rawDFL.WIND.Time;
            WIND(:,3) = rawDFL.WIND.wdir;
            WIND(:,4) = rawDFL.WIND.wspeed;
            WIND(:,5) = rawDFL.WIND.R13;
            WIND(:,6) = rawDFL.WIND.R23;
            WIND(:,7) = rawDFL.WIND.R33;
            WIND(:,8) = rawDFL.WIND.DatenumUTC;
            WIND_label = {'LineNo';'TimeUS';'wdir';'wspeed';...
                'R13';'R23';'R33';'DatenumUTC'};
            %% IMET
            IMET(:,1) = rawDFL.IMET.LineNo;
            IMET(:,2) = rawDFL.IMET.Time;
            IMET(:,3) = rawDFL.IMET.R1./100;
            IMET(:,4) = rawDFL.IMET.R2./100;
            IMET(:,5) = rawDFL.IMET.R3./100;
            IMET(:,6) = rawDFL.IMET.R4./100;
            IMET(:,7) = rawDFL.IMET.T1;
            IMET(:,8) = rawDFL.IMET.T2;
            IMET(:,9) = rawDFL.IMET.T3;
            IMET(:,10) = rawDFL.IMET.T4;
            IMET(:,11) = rawDFL.IMET.Hth1;
```

```
            IMET(:,12) = rawDFL.IMET.Hth2;
            IMET(:,13) = rawDFL.IMET.Hth3;
            IMET(:,14) = rawDFL.IMET.Hth4;
            IMET(:,15) = rawDFL.IMET.Fan;
            IMET(:,16) = rawDFL.IMET.DatenumUTC;
            IMET_label = {'LineNo';'TimeUS';'RH1';'RH2';'RH3';'RH4';...
                'T1';'T2';'T3';'T4';'Health1';'Health2';'Health3';...
                'Health4';'Fan';'DatenumUTC'};
        end

        %% GPS
        GPS(:,1) = rawDFL.GPS.LineNo;
        GPS(:,2) = rawDFL.GPS.TimeUS;
        GPS(:,3) = rawDFL.GPS.Status;
        GPS(:,4) = rawDFL.GPS.GMS;
        GPS(:,5) = rawDFL.GPS.GWk;
        GPS(:,6) = rawDFL.GPS.NSats;
        GPS(:,7) = rawDFL.GPS.HDop;
        GPS(:,8) = rawDFL.GPS.Lat;
        GPS(:,9) = rawDFL.GPS.Lng;
        GPS(:,10) = rawDFL.GPS.Alt;
        GPS(:,11) = rawDFL.GPS.Spd;
        GPS(:,12) = rawDFL.GPS.GCrs;
        GPS(:,13) = rawDFL.GPS.VZ;
        GPS(:,14) = rawDFL.GPS.U;
        GPS(:,15) = rawDFL.GPS.DatenumUTC;
        GPS_label = {'LineNo';'TimeUS';'Status';'GMS';'GWk';'NSats';...
            'HDop';'Lat';'Lng';'Alt';'Spd';'GCrs';'VZ';'U';...
            'DatenumUTC'};
        %% ATT
        ATT(:,1) = rawDFL.ATT.LineNo;
        ATT(:,2) = rawDFL.ATT.TimeUS;
```

```
        ATT(:,3) = rawDFL.ATT.DesRoll;
        ATT(:,4) = rawDFL.ATT.Roll;
        ATT(:,5) = rawDFL.ATT.DesPitch;
        ATT(:,6) = rawDFL.ATT.Pitch;
        ATT(:,7) = rawDFL.ATT.DesYaw;
        ATT(:,8) = rawDFL.ATT.Yaw;
        ATT(:,9) = rawDFL.ATT.ErrRP;
        ATT(:,10) = rawDFL.ATT.ErrYaw;
        ATT(:,11) = rawDFL.ATT.DatenumUTC;
        ATT_label = {'LineNo';'TimeUS';'DesRoll';'Roll';'DesPitch';...
            'Pitch';'DesYaw';'Yaw';'ErrRP';'ErrYaw';'DatenumUTC'};
        %% BARO
        BARO(:,1) = rawDFL.BARO.LineNo;
        BARO(:,2) = rawDFL.BARO.TimeUS;
        BARO(:,3) = rawDFL.BARO.Alt;
        BARO(:,4) = rawDFL.BARO.Press;
        BARO(:,5) = rawDFL.BARO.Temp;
        BARO(:,6) = rawDFL.BARO.CRt;
        BARO(:,7) = rawDFL.BARO.SMS;
        BARO(:,8) = rawDFL.BARO.Offset;
        BARO(:,9) = rawDFL.BARO.GndTemp;
        BARO(:,10) = rawDFL.BARO.DatenumUTC;
        BARO_label = {'LineNo';'TimeUS';'Alt';'Press';'Temp';...
            'CRt';'SMS';'Offset';'GndTemp';'DatenumUTC'};
        %% RCOU
        RCOU(:,1) = rawDFL.RCOU.LineNo;
        RCOU(:,2) = rawDFL.RCOU.TimeUS;
        RCOU(:,3) = rawDFL.RCOU.C1;
        RCOU(:,4) = rawDFL.RCOU.C2;
        RCOU(:,5) = rawDFL.RCOU.C3;
        RCOU(:,6) = rawDFL.RCOU.C4;
        RCOU(:,7) = rawDFL.RCOU.C5;
```

```
        RCOU(:,8) = rawDFL.RCOU.C6;
        RCOU(:,9) = rawDFL.RCOU.C7;
        RCOU(:,10) = rawDFL.RCOU.C8;
        RCOU(:,11) = rawDFL.RCOU.C9;
        RCOU(:,12) = rawDFL.RCOU.C10;
        RCOU(:,13) = rawDFL.RCOU.C11;
        RCOU(:,14) = rawDFL.RCOU.C12;
        RCOU(:,15) = rawDFL.RCOU.C13;
        RCOU(:,16) = rawDFL.RCOU.C14;
        RCOU(:,17) = rawDFL.RCOU.DatenumUTC;
        RCOU_label = {'LineNo';'TimeUS';'C1';'C2';'C3';'C4';'C5';...
            'C6';'C7';'C8';'C9';'C10';'C11';'C12';'C13';'C14';...
            'DatenumUTC'};
        %% IMU
        IMU(:,1) = rawDFL.IMU.LineNo;
        IMU(:,2) = rawDFL.IMU.TimeUS;
        IMU(:,3) = rawDFL.IMU.GyrX;
        IMU(:,4) = rawDFL.IMU.GyrY;
        IMU(:,5) = rawDFL.IMU.GyrZ;
        IMU(:,6) = rawDFL.IMU.AccX;
        IMU(:,7) = rawDFL.IMU.AccY;
        IMU(:,8) = rawDFL.IMU.AccZ;
        IMU(:,9) = rawDFL.IMU.EG;
        IMU(:,10) = rawDFL.IMU.EA;
        IMU(:,11) = rawDFL.IMU.T;
        IMU(:,12) = rawDFL.IMU.GH;
        IMU(:,13) = rawDFL.IMU.AH;
        IMU(:,14) = rawDFL.IMU.GHz;
        IMU(:,15) = rawDFL.IMU.AHz;
        IMU(:,16) = rawDFL.IMU.DatenumUTC;
        IMU_label = {'LineNo';'TimeUS';'GyrX';'GyrY';'GyrZ';...
            'AccX';'AccY';'AccZ';'EG';'EA';'T';'GH';'AH';'GHz';...
```

```
        'AHz';'DatenumUTC'};
    %% NKF2
    try
        NKF2(:,1) = rawDFL.NKF2.LineNo;
        NKF2(:,2) = fix(...
            rawDFL.NKF2.TimeS/rawDFL.NKF2.fieldMultipliers.TimeUS);
        NKF2(:,3) = rawDFL.NKF2.AZbias;
        NKF2(:,4) = rawDFL.NKF2.GSX;
        NKF2(:,5) = rawDFL.NKF2.GSY;
        NKF2(:,6) = rawDFL.NKF2.GSZ;
        NKF2(:,7) = rawDFL.NKF2.VWN;
        NKF2(:,8) = rawDFL.NKF2.VWE;
        NKF2(:,9) = rawDFL.NKF2.MN;
        NKF2(:,10) = rawDFL.NKF2.ME;
        NKF2(:,11) = rawDFL.NKF2.MD;
        NKF2(:,12) = rawDFL.NKF2.MX;
        NKF2(:,13) = rawDFL.NKF2.MY;
        NKF2(:,14) = rawDFL.NKF2.MZ;
        NKF2(:,15) = rawDFL.NKF2.MI;
        NKF2(:,16) = rawDFL.NKF2.DatenumUTC;
        NKF2_label = {'LineNo';'TimeUS';'AZbias';'GSX';'GSY';...
            'GSZ';'VWN';'VWE';'MN';'ME';'MD';'MX';'MY';...
            'MZ';'MI';'DatenumUTC'};

    catch
        NKF2_label = {'LineNo';'TimeUS';'AZbias';'GSX';'GSY';...
            'GSZ';'VWN';'VWE';'MN';'ME';'MD';'MX';'MY';...
            'MZ';'MI';'DatenumUTC'};
    end


    % Save asnew file for integration with normal code
```

```
            [¬, baseNameNoExtDFL, ¬] = fileparts(baseFileNameDFL);
            baseFileName = sprintf('%s.mat', baseNameNoExtDFL);
            fullParsedMatFileName = fullfile(folderDFL, baseFileName);
            % Save file with parsed data as the original filename plus ...
                the added portion

            if(strcmpi(arduPilotType,'ArduCopter'))
                save(fullParsedMatFileName,'GPS','GPS_label','ATT',...
                    'ATT_label','BARO','BARO_label','NKF2','NKF2_label',...
                    'IMU','IMU_label','RCOU','RCOU_label','MSG1');
            elseif(strcmpi(arduPilotType,'CopterSonde'))
                save(fullParsedMatFileName,'GPS','GPS_label','WIND',...
                    'WIND_label','IMET','IMET_label','ATT','ATT_label',...
                    'BARO','BARO_label','NKF2','NKF2_label','IMU',...
                    'IMU_label','RCOU','RCOU_label','MSG1');
            end
            fullInputMatFileNameDFL = fullParsedMatFileName;
        end
    end
    %% Gather Additional Files
    variableInfo = who('-file',fullInputMatFileNameDFL);

    if(length(variableInfo)<20 && strcmpi(DFL_NewOld,'Unknown')...
            && ismember('GPS_table',variableInfo))


        load(fullInputMatFileNameDFL);
        DFL_NewOld = 'Old';

        if(ismember('MHP_table',variableInfo))
            mhpValue = 'Yes';
        else mhpValue = 'No';
```

```
        end

        if(ismember('TPH_table',variableInfo))
            tphValue = 'Yes';
        else tphValue = 'No';
        end

        if(ismember('iMet_table',variableInfo))
            iMetValue = 'Yes';
        else iMetValue = 'No';
        end

        if(ismember('CSIMET_table',variableInfo))
            arduPilotType = 'CopterSonde';
        end

    else
        DFL_NewOld = 'New';

        % Parser for ArduPilotType
        if(strcmpi(arduPilotType,'Default'))

            if(ismember('IMET',variableInfo))
                arduPilotType = 'Coptersonde';
            else
                load(fullInputMatFileNameDFL,'MSG1');

                QuadPlane = 0;
                ArduPlane = 0;
                for(i=1:length(MSG1))
                    try
                        if(strcmpi(MSG1{1,i}{3,1}(1,1:9),{'QuadPlane'}))
```

```
                            QuadPlane = 1;
                        elseif(strcmpi(MSG1{1,i}{3,1}(1,1:9),{'ArduPlane'}))
                            ArduPlane = 1;
                        elseif(strcmpi(MSG1{1,i}{3,1}(1,1:10),{'ArduCopter'}))
                            arduPilotType = 'ArduCopter';
                        end

                    catch
                    end
                end

                if((QuadPlane + ArduPlane) == 2)
                    arduPilotType = 'Quad-Plane';
                elseif((QuadPlane + ArduPlane) == 1)
                    arduPilotType = 'Fixed Wing';
                end
            end
        end

        % Execution of ArduPilotType
        if(strcmpi(arduPilotType,'Fixed Wing'))
            load(fullInputMatFileNameDFL,'ATT','ATT_label','BARO',...
                'BARO_label','CTUN','CTUN_label','GPS','GPS_label',...
                'IMU','IMU_label','NKF1','NKF1_label','NKF2','NKF2_label',...
                'RCOU','RCOU_label');

            % Pre-parse for only relevant data series
            ATT = [ATT(:,1),ATT(:,2),ATT(:,4),ATT(:,6),ATT(:,8)];
            ATT_label = ...
                [ATT_label(1),ATT_label(2),ATT_label(4),ATT_label(6),ATT_label(8)];
            BARO = [BARO(:,1:4)];
            BARO_label = [BARO_label(1:4)];
```

```
            CTUN = ...
                [CTUN(:,1),CTUN(:,2),CTUN(:,4),CTUN(:,6),CTUN(:,8),CTUN(:,10)];
            CTUN_label = [CTUN_label(1),CTUN_label(2),CTUN_label(4),...
                CTUN_label(6),CTUN_label(8),CTUN_label(10)];
            NKF1 = [NKF1(:,1),NKF1(:,2), NKF1(:,3), NKF1(:,4), ...
                NKF1(:,5),...
                NKF1(:,6), NKF1(:,7), NKF1(:,8), NKF1(:,9), NKF1(:,10), ...
                    NKF1(:,11)];%, NKF1(:,12), NKF1(:,13), NKF1(:,14), ...
                    NKF1(:,15), NKF1(:,16)];
            NKF1_label = [NKF1_label(1),NKF1_label(2), NKF1_label(3),...
                NKF1_label(4), NKF1_label(5), NKF1_label(6), ...
                    NKF1_label(7),...
                NKF1_label(8), NKF1_label(9), NKF1_label(10), ...
                    NKF1_label(11)]; %, NKF1_label(12), NKF1_label(13), ...
                    NKF1_label(14), NKF1_label(15), NKF1_label(16)];
            GPS = ...
                [GPS(:,1),GPS(:,2),GPS(:,4),GPS(:,5),GPS(:,8),GPS(:,9),...
                GPS(:,10),GPS(:,11)];
            GPS_label = ...
                [GPS_label(1),GPS_label(2),GPS_label(4),GPS_label(5),...
                GPS_label(8),GPS_label(9),GPS_label(10),GPS_label(11)];
            IMU = [IMU(:,1:8)];
            IMU_label = [IMU_label(1:8)];
            NKF2 = [NKF2(:,1),NKF2(:,2),NKF2(:,7),NKF2(:,8)];
            NKF2_label = ...
                [NKF2_label(1),NKF2_label(2),NKF2_label(7),NKF2_label(8)];
            RCOU = [RCOU(:,1:6)];
            RCOU_label = [RCOU_label(1:6)];
        end

        if(strcmpi(arduPilotType,'Quad-Plane'))
            load(fullInputMatFileNameDFL,'ATT','ATT_label','BARO',...
```

```
        'BARO_label','CTUN','CTUN_label','IMU','IMU_label',...
        'GPS','GPS_label','XKF1','XKF1_label','XKF2','XKF2_label',...
        'RCOU','RCOU_label');

    % Pre-parse for only relevant data series
    ATT = [ATT(:,1),ATT(:,2),ATT(:,4),ATT(:,6),ATT(:,8)];
    ATT_label = ...
       [ATT_label(1),ATT_label(2),ATT_label(4),ATT_label(6),ATT_label(8)];
    BARO = [BARO(:,1:4)];
    BARO_label = [BARO_label(1:4)];
    CTUN = ...
       [CTUN(:,1),CTUN(:,2),CTUN(:,4),CTUN(:,6),CTUN(:,8),CTUN(:,10)];
    CTUN_label = [CTUN_label(1),CTUN_label(2),CTUN_label(4),...
        CTUN_label(6),CTUN_label(8),CTUN_label(10)];
    NKF1 = [XKF1(:,1),XKF1(:,2), XKF1(:,3), XKF1(:,4), ...
       XKF1(:,5),...
        XKF1(:,6), XKF1(:,7), XKF1(:,8), XKF1(:,9), XKF1(:,10), ...
           XKF1(:,11)]; %, XKF1(:,12), XKF1(:,13), XKF1(:,14), ...
           XKF1(:,15), XKF1(:,16)];
    NKF1_label = [XKF1_label(1),XKF1_label(2), XKF1_label(3),...
        XKF1_label(4), XKF1_label(5), XKF1_label(6), ...
           XKF1_label(7),...
        XKF1_label(8), XKF1_label(9), XKF1_label(10), ...
           XKF1_label(11)]; %, XKF1_label(12), XKF1_label(13), ...
           XKF1_label(14), XKF1_label(15), XKF1_label(16)];
    GPS=[GPS(:,1),GPS(:,2),GPS(:,4),GPS(:,5),GPS(:,8),GPS(:,9),...
        GPS(:,10),GPS(:,11)];
    GPS_label = [GPS_label(1),GPS_label(2),GPS_label(4),...
        GPS_label(5),GPS_label(8),GPS_label(9),GPS_label(10),...
        GPS_label(11)];
    IMU = [IMU(:,1:8)];
    IMU_label = [IMU_label(1:8)];
```

```
        NKF2 = [XKF2(:,1),XKF2(:,2),XKF2(:,6),XKF2(:,7)];
        NKF2_label = [XKF2_label(1),XKF2_label(2),XKF2_label(6),...
            XKF2_label(7)];
        RCOU = [RCOU(:,1:6)];
        RCOU_label = [RCOU_label(1:6)];
    end

    % Not configured OF NOTE: Have not implemented NKF1 in DFL load
    if(strcmpi(arduPilotType,'ArduCopter'))
        load(fullInputMatFileNameDFL,'ATT','ATT_label','BARO',...
            'BARO_label','IMU','IMU_label','GPS','GPS_label','NKF1',...
            'NKF1_label','NKF2','NKF2_label','RCOU','RCOU_label');

        % Pre-parse for only relevant data series
        ATT = [ATT(:,1),ATT(:,2),ATT(:,4),ATT(:,6),ATT(:,8)];
        ATT_label = [ATT_label(1),ATT_label(2),ATT_label(4),...
            ATT_label(6),ATT_label(8)];
        BARO = [BARO(:,1:4)];
        BARO_label = [BARO_label(1:4)];
        GPS=[GPS(:,1),GPS(:,2),GPS(:,4),GPS(:,5),GPS(:,8),GPS(:,9),...
            GPS(:,10),GPS(:,11)];
        GPS_label = [GPS_label(1),GPS_label(2),GPS_label(4),...
            GPS_label(5),GPS_label(8),GPS_label(9),GPS_label(10),...
            GPS_label(11)];
        IMU = [IMU(:,1:8)];
        IMU_label = [IMU_label(1:8)];
        %new
        NKF1 = [NKF1(:,1),NKF1(:,2), NKF1(:,3), NKF1(:,4), ...
           NKF1(:,5),...
            NKF1(:,6), NKF1(:,7), NKF1(:,8)];%, NKF1(:,9), ...
                NKF1(:,10), NKF1(:,11), NKF1(:,12), NKF1(:,13), ...
                NKF1(:,14), NKF1(:,15), NKF1(:,16)];
```

```
            NKF1_label = [NKF1_label(1),NKF1_label(2), NKF1_label(3),...
                NKF1_label(4), NKF1_label(5), NKF1_label(6), ...
                    NKF1_label(7),...
                NKF1_label(8)];%, NKF1_label(9), NKF1_label(10), ...
                    NKF1_label(11), NKF1_label(12), NKF1_label(13), ...
                    NKF1_label(14), NKF1_label(15), NKF1_label(16)];
            %new
            NKF2 = [NKF2(:,1),NKF2(:,2),NKF2(:,6),NKF2(:,7)];
            NKF2_label = [NKF2_label(1),NKF2_label(2),NKF2_label(6),...
                NKF2_label(7)];
            RCOU = [RCOU(:,1:6)];
            RCOU_label = [RCOU_label(1:6)];
        end

        % Not configured
        if(strcmpi(arduPilotType,'CopterSonde'))
            load(fullInputMatFileNameDFL,'ATT','ATT_label','WIND',...
                'WIND_label','IMET','IMET_label','BARO','BARO_label',...
                'IMU','IMU_label','GPS','GPS_label','RCOU','RCOU_label');

            % Pre-parse for only relevant data series
            ATT = [ATT(:,1),ATT(:,2),ATT(:,4),ATT(:,6),ATT(:,8)];
            ATT_label = [ATT_label(1),ATT_label(2),ATT_label(4),...
                ATT_label(6),ATT_label(8)];
            BARO = [BARO(:,1:4)];
            BARO_label = [BARO_label(1:4)];
            GPS=[GPS(:,1),GPS(:,2),GPS(:,4),GPS(:,5),GPS(:,8),GPS(:,9),...
                GPS(:,10),GPS(:,11)];
            GPS_label = [GPS_label(1),GPS_label(2),GPS_label(4),...
                GPS_label(5),GPS_label(8),GPS_label(9),GPS_label(10),...
                GPS_label(11)];
            IMU = [IMU(:,1:8)];
```

```
            IMU_label = [IMU_label(1:8)];
            RCOU = [RCOU(:,1:6)];
            RCOU_label = [RCOU_label(1:6)];
        end

        % Not configured, placeholder only
        if(strcmpi(arduPilotType,'ArduRover'))
            load(fullInputMatFileNameDFL,'ATT','ATT_label','BARO',...
                'BARO_label','CTUN','CTUN_label','GPS','GPS_label','IMU',...
                'IMU_label','NKF2','NKF2_label','RCOU','RCOU_label',...
                'STAT','STAT_label');

            % Pre-parse for only relevant data series
            ATT = [ATT(:,1),ATT(:,2),ATT(:,4),ATT(:,6),ATT(:,8)];
            ATT_label = [ATT_label(1),ATT_label(2),ATT_label(4),...
                ATT_label(6),ATT_label(8)];
            BARO = [BARO(:,1:4)];
            BARO_label = [BARO_label(1:4)];
            CTUN = [CTUN(:,1),CTUN(:,2),CTUN(:,4),CTUN(:,6),CTUN(:,8),...
                CTUN(:,10)];
            CTUN_label = [CTUN_label(1),CTUN_label(2),CTUN_label(4),...
                CTUN_label(6),CTUN_label(8),CTUN_label(10)];
            GPS = [GPS(:,1),GPS(:,2),GPS(:,4),GPS(:,5),GPS(:,8),...
                GPS(:,9),GPS(:,10),GPS(:,11)];
            GPS_label = [GPS_label(1),GPS_label(2),GPS_label(4),...
                GPS_label(5),GPS_label(8),GPS_label(9),...
                GPS_label(10),GPS_label(11)];
            IMU = [IMU(:,1:8)];
            IMU_label = [IMU_label(1:8)];
            NKF2 = [NKF2(:,1),NKF2(:,2),NKF2(:,7),NKF2(:,8)];
            NKF2_label = [NKF2_label(1),NKF2_label(2),NKF2_label(7),...
                NKF2_label(8)];
```

```
            RCOU = [RCOU(:,1:6)];
            RCOU_label = [RCOU_label(1:6)];
            STAT = [STAT(:,1:5)];
            STAT_label = [STAT_label(1:5)];
        end

    end

    % Get filename without the extension, used by Save Function
    [¬, baseNameNoExtDFL, ¬] = fileparts(baseFileNameDFL);

    if (strcmpi(pitchToggle,'Yes'))

        if(strcmpi(aircraft,'TIA'))
            pitchMap=[1139,-7.9 ; 1150,-7.1 ; 1200,-6.3 ; 1250,-5.6 ;...
                1300,-4.8 ; 1350,-4.0 ; 1400,-3.2 ; 1450,-1.6 ; ...
                    1500,0.0 ;...
                1550,6.3 ; 1600,7.9 ; 1650,9.5 ; 1700,11.0 ; 1750,12.5 ...
                    ; 1800,14.0 ; 1850,15.5 ; 1900,17.0 ; 1939,17.0];
        else
            % Have user browse for a file, from a specified "starting ...
                folder."
            % For convenience in browsing, set a starting folder from ...
                which to browse.
            % Start in the current folder.
            startingFolder = pwd;

            % Get the name of the file that the user wants to use.
            defaultFileName = fullfile(startingFolder, '*.txt');
            [baseFileName, folder] = uigetfile(defaultFileName,...
                'Select a Pitch PWM text file');
            if baseFileName == 0
```

```
                % User clicked the Cancel button.
                return;
            end

            % Get the name of the input .mat file.
            fullInputMatFileName = fullfile(folder, baseFileName);
            pitchMap = load(fullInputMatFileName);
        end
    end

    if (strcmpi(throttleToggle,'Yes'))

        if(strcmpi(aircraft,'TIA'))
            throttleMap=[0,0.00 ; 10,1.02 ; 20,1.91 ; 30,2.85 ; 40,3.93 ...
                ; 50,5.00 ; 60,6.31 ; 70,7.92 ; 80,9.57 ; 89,11.17 ; ...
                100,13.37];
        else
            % Have user browse for a file, from a specified "starting ...
                folder."
            % For convenience in browsing, set a starting folder from ...
                which to browse.
            % Start in the current folder.
            startingFolder = pwd;

            % Get the name of the file that the user wants to use.
            defaultFileName = fullfile(startingFolder, '*.txt');
            [baseFileName, folder] = uigetfile(defaultFileName, 'Select ...
                a Throttle Mapping text file');
            if baseFileName == 0
                % User clicked the Cancel button.
                return;
            end
```

```
            % Get the name of the input .mat file.
            fullInputMatFileName = fullfile(folder, baseFileName);
            throttleMap = load(fullInputMatFileName);
        end
    end
    %% Set Parsing Bounds
    if(strcmpi(DFL_NewOld,'New'))

        fig1=figure(1);
        fig1.Name =...
'Raw data from DFL. Click on graph for upper and lower bound for parsing.';

        if(strcmpi(arduPilotType,'ArduCopter') ...
                | strcmpi(arduPilotType,'CopterSonde'))
            plt1 = subplot(4,1,1);
            plot(GPS(:,2),GPS(:,8),'b')
            title('Groundspeed vs Time')
            ylabel({'Groundspeed (blue)';'(m/s)'})
        else
            % Groundspeed plot
            plt1 = subplot(4,1,1);
            plot(GPS(:,2),GPS(:,8),'b',CTUN(:,2),CTUN(:,6),'r')
            title('Groundspeed, Airspeed vs Time')
            ylabel({'Groundspeed (blue)';'Airspeed (red)';'(m/s)'})
        end

        % Throttle Output
        plt2 = subplot(4,1,2);
        plot(RCOU(:,2),((RCOU(:,5)-thrMinPWM)/(thrMaxPWM-thrMinPWM)*100),'b')
        title('Throttle vs Time')
        ylabel({'Throttle';'(%)'})
```

```
ylim([0 100])

% For the dotted line along x-axis of pitch plot
zero=int8(zeros(length(ATT(:,2)),1));

% Aircraft Pitch angle: Can change ylim to something more relevant.
% TIV uses -20 to 50 to see high AoA landing
plt3 = subplot(4,1,3);
plot(ATT(:,2),ATT(:,4),'b',ATT(:,2),zero,'r:')
title('Aircraft Pitch Angle vs Time')
ylabel({'Aircraft Pitch';'Angle (  )'})
ylim([-10 40])

% Altitude plot
plt4 = subplot(4,1,4);
plot(GPS(:,2),GPS(:,7),'b')
title('Altitude vs Time')
ylabel({'Altitude MSL';'(m)'})
xlabel('Time (microseconds)')
linkaxes([plt1 plt2 plt3 plt4],'x')
xlim([min(GPS(:,2)) max(GPS(:,2))])

m=0;
while true
    [horiz, vert, button] = ginput(1);
    if isempty(horiz) || button(1) == 3; break; end
    m = m+1;
    x_m(m) = horiz(1); % save all points you continue getting
    hold on
    if(strcmpi(arduPilotType,'ArduCopter')...
            | strcmpi(arduPilotType,'CopterSonde'))
```

```
        else y_va(m)=CTUN(find(CTUN(:,2)≥x_m(m),1,'first'),6);    ...
            % Airspeed
        end
        y_vg(m)=GPS(find(GPS(:,2)≥x_m(m),1,'first'),8);      % ...
            Groundspeed
        y_thr(m)=RCOU(find(RCOU(:,2)≥x_m(m),1,'first'),5);     % ...
            Throttle Percent
        y_pitch(m)=ATT(find(ATT(:,2)≥x_m(m),1,'first'),4); % ...
            Aircraft Pitch
        y_alt(m)=GPS(find(GPS(:,2)≥x_m(m),1,'first'),7);    % Altitude

        if(strcmpi(arduPilotType,'ArduCopter')...
                | strcmpi(arduPilotType,'CopterSonde'))
            % Groundspeed plot
            subplot(4,1,1)
            plot(GPS(:,2),GPS(:,8),'b',x_m,y_vg,'kx')
            title('Groundspeed vs Time')
            ylabel({'Groundspeed (blue)';'(m/s)'})
        else
            % Groundspeed plot
            subplot(4,1,1)
            plot(GPS(:,2),GPS(:,8),'b',CTUN(:,2),...
                CTUN(:,6),'r',x_m,y_vg,'kx',x_m,y_va,'kx')
            title('Groundspeed, Airspeed vs Time')
            ylabel({'Groundspeed (blue)';'Airspeed (red)';'(m/s)'})
        end

        % Throttle Output
        subplot(4,1,2)
        plot(RCOU(:,2),((RCOU(:,5)-thrMinPWM)/...
            (thrMaxPWM-thrMinPWM)*100),'b',x_m, ((y_thr-thrMinPWM)/...
            (thrMaxPWM-thrMinPWM)*100),'kx')
```

```
        title('Throttle vs Time')
        ylabel({'Throttle';'(%)'})
        ylim([0 100])

        % For the dotted line along x-axis of pitch plot
        zero=int8(zeros(length(ATT(:,2)),1));

        % Aircraft Pitch angle: Can change ylim to something more ...
            relevant.
        % TIV uses -20 to 50 to see high AoA landing
        subplot(4,1,3)
        plot(ATT(:,2),ATT(:,4),'b',ATT(:,2),zero,'r:',x_m,y_pitch,'kx')
        title('Aircraft Pitch Angle vs Time')
        ylabel({'Aircraft Pitch';'Angle (  )'})
        ylim([-10 40])

        % Altitude plot
        subplot(4,1,4)
        plot(GPS(:,2),GPS(:,7),'b',x_m,y_alt,'kx')
        title('Altitude vs Time')
        ylabel({'Altitude MSL';'(m)'})
        xlabel('Time (microseconds)')
        xlim([min(GPS(:,2)) max(GPS(:,2))])

        drawnow

        if(m≥2)
            break;
        end

    end

```

```
        fig2=figure(2);
        fig2.Name = 'Preview of user-parsed DFL data.';

        if(strcmpi(arduPilotType,'ArduCopter') |...
                strcmpi(arduPilotType,'CopterSonde'))
            % Groundspeed plot
            plt1 = subplot(4,1,1);
            plot(GPS(:,2)/1000000,GPS(:,8),'b')
            title('Groundspeed vs Time')
            ylabel({'Groundspeed (blue)';'(m/s)'})
        else
            % Groundspeed plot
            plt1 = subplot(4,1,1);
            plot(GPS(:,2)/1000000,GPS(:,8),'b',CTUN(:,2)/1000000,CTUN(:,6),'r')
            title('Groundspeed, Airspeed vs Time')
            ylabel({'Groundspeed (blue)';'Airspeed (red)';'(m/s)'})
        end

        % Throttle Output
        plt2 = subplot(4,1,2);
        plot(RCOU(:,2)/1000000,((RCOU(:,5)-thrMinPWM)/(thrMaxPWM-thrMinPWM)*100),'b')
        title('Throttle vs Time')
        ylabel({'Throttle';'(%)'})
        ylim([0 100])

        % For the dotted line along x-axis of pitch plot
        zero=int8(zeros(length(ATT(:,2)),1));

        % Aircraft Pitch angle: Can change ylim to something more relevant.
        % TIV uses -20 to 50 to see high AoA landing
        plt3 = subplot(4,1,3);
        plot(ATT(:,2)/1000000,ATT(:,4),'b',ATT(:,2)/1000000,zero,'r:')
```

```
        title('Aircraft Pitch Angle vs Time')
        ylabel({'Aircraft Pitch';'Angle (  )'})
        ylim([-10 40])

        % Altitude plot
        plt4 = subplot(4,1,4);
        plot(GPS(:,2)/1000000,GPS(:,7),'b')
        title('Altitude vs Time')
        ylabel({'Altitude MSL';'(m)'})
        xlabel('Time (seconds)')
        linkaxes([plt1 plt2 plt3 plt4],'x')
        xlim([x_m(1)/1000000 x_m(2)/1000000])

    end
    %% Configure Start/Stop Conditions
    tic
    if(strcmpi(DFL_NewOld,'New'))

        % Finding the applicable data range based on minimum settings above
        % GPS line number of starting relavent data (used for parsing)
        TO = IMU(find(IMU(:,2)≥x_m(1), 1, 'first'),1);
        LND = IMU(find(IMU(:,2)≥x_m(2), 1, 'first'),1);

        % Potision in each major dataset (GPS, CTUN, NKF1, NKF2, RCOU) ... for Takeoff (TO)
        % and Landing (LND)
        if(strcmpi(arduPilotType,'ArduCopter'))
            TO_NKF1 = find(NKF1(:,1)>TO,1,'first')-1;
            LND_NKF1 = find(NKF1(:,1)>LND,1,'first')-1;
            TO_NKF2 = find(NKF2(:,1)>TO,1,'first')-1;
            LND_NKF2 = find(NKF2(:,1)>LND,1,'first')-1;
        elseif(strcmpi(arduPilotType,'CopterSonde'))
```

```
            TO_WIND = find(WIND(:,1)>TO,1,'first')-1;
            LND_WIND = find(WIND(:,1)>LND,1,'first')-1;
            TO_IMET = find(IMET(:,1)>TO,1,'first')-1;
            LND_IMET = find(IMET(:,1)>LND,1,'first')-1;
        else
            TO_CTUN = find(CTUN(:,1)>TO,1,'first')-1;
            LND_CTUN = find(CTUN(:,1)>LND,1,'first')-1;
            TO_NKF1 = find(NKF1(:,1)>TO,1,'first')-1;
            LND_NKF1 = find(NKF1(:,1)>LND,1,'first')-1;
            TO_NKF2 = find(NKF2(:,1)>TO,1,'first')-1;
            LND_NKF2 = find(NKF2(:,1)>LND,1,'first')-1;
        end
        TO_GPS = find(GPS(:,1)>TO,1,'first')-1;
        LND_GPS = find(GPS(:,1)>LND,1,'first')-1;
        TO_ATT = find(ATT(:,1)>TO,1,'first')-1;
        LND_ATT = find(ATT(:,1)>LND,1,'first')-1;
        TO_RCOU = find(RCOU(:,1)>TO,1,'first')-1;
        LND_RCOU = find(RCOU(:,1)>LND,1,'first')-1;
        TO_IMU = find(IMU(:,1)>TO,1,'first')-1;
        LND_IMU = find(IMU(:,1)>LND,1,'first')-1;
        TO_BARO = find(BARO(:,1)>TO,1,'first')-1;
        LND_BARO = find(BARO(:,1)>LND,1,'first')-1;

        %%%%%%%%% GPS Logs Parsing %%%%%%%%%
        % Latitude
        x=GPS(TO_GPS:LND_GPS,5);
        % Longitude
        y=GPS(TO_GPS:LND_GPS,6);
        % Altitude
        z=GPS(TO_GPS:LND_GPS,7);
        z_AGL = z(:)-z(1);

```

```
        %%%%%%%%% Airspeed and Windspeed Data %%%%%%%%%
        % Ground speed
        v_g=GPS(TO_GPS:LND_GPS,8);
        % Max ground speed
        max_v_g=max(GPS(TO_GPS:LND_GPS,8));
        if(strcmpi(arduPilotType,'CopterSonde'))
            wind = WIND(TO_WIND:LND_WIND,4);
            VWN = cos(deg2rad(WIND(TO_WIND:LND_WIND,3))).*wind;
            VWE = sin(deg2rad(WIND(TO_WIND:LND_WIND,3))).*wind;
        elseif(strcmpi(arduPilotType,'ArduCopter'))
            % North winds
            VWN=NKF2(TO_NKF2:LND_NKF2,3);
            % East winds
            VWE=NKF2(TO_NKF2:LND_NKF2,4);
            % Wind Speed
            wind=(VWN.^2+VWE.^2).^0.5;
            % Wind Angle
            windANG=atan2d(VWN,VWE);
        else
            % Airspeed
            v_a=CTUN(TO_CTUN:LND_CTUN,6);
            % North winds
            VWN=NKF2(TO_NKF2:LND_NKF2,3);
            % East winds
            VWE=NKF2(TO_NKF2:LND_NKF2,4);
            % Wind speed
            wind=(VWN.^2+VWE.^2).^0.5;
            % Wind Angle
            windANG=atan2d(VWN,VWE);
        end

        %%%%%%%%% Aircraft Data %%%%%%%%%
```

```
        % Pitch PWM signal
        pitchPWM=RCOU(TO_RCOU:LND_RCOU,3);
        % Aircraft pitch angle
        pitchAC=ATT(TO_ATT:LND_ATT,4);
        % Aircraft roll angle
        rollAC = ATT(TO_ATT:LND_ATT,3);
        % Aircraft yaw (Earth reference, degrees)
        yawAC = ATT(TO_ATT:LND_ATT,5);
        % Throttle output from Pixhawk
        thr=(RCOU(TO_RCOU:LND_RCOU,5)-thrMinPWM)/(thrMaxPWM-thrMinPWM)*100;


        %%%% EKF Aircraft Data %%%%
        % EKF Roll
        rollEKF=NKF1(TO_NKF1:LND_NKF1,3);
        % EKF Pitch
        pitchEKF=NKF1(TO_NKF1:LND_NKF1,4);
        % EKF Yaw
        yawEKF=NKF1(TO_NKF1:LND_NKF1,5);
        % North Aircraft Velocity
        vnEKF=NKF1(TO_NKF1:LND_NKF1,6);
        % East Aircraft Velocity
        veEKF=NKF1(TO_NKF1:LND_NKF1,7);
        % Down Aircraft Velocity
        vdEKF=NKF1(TO_NKF1:LND_NKF1,8);
        % Position North
        pnEKF=NKF1(TO_NKF1:LND_NKF1,9);
        % Position North
        peEKF=NKF1(TO_NKF1:LND_NKF1,10);
        % Position North
        pdEKF=NKF1(TO_NKF1:LND_NKF1,11);
    end
```

```
    %% Parse All Data and Save To Respective Tables
    if(strcmpi(DFL_NewOld,'New'))
        % Parsed GPS data output
        GPS_LN = GPS(TO_GPS:LND_GPS,1);
        GPS_time = GPS(TO_GPS:LND_GPS,2);
        GPS_ms = GPS(TO_GPS:LND_GPS,3);
        GPS_wk = GPS(TO_GPS:LND_GPS,4);
        GPS_time_out= (GPS_time-min(GPS_time))/1000000;
        % Convert GPS timestamps to UTC time
        leap_second_table = datenum...
            (['Jul 01 1981';'Jul 01 1982';'Jul 01 1983';'Jul 01 ...
                1985';'Jan 01 1988';...
            'Jan 01 1990';'Jan 01 1991';'Jul 01 1992';'Jul 01 1993';...
            'Jul 01 1994';'Jan 01 1996';'Jul 01 1997';'Jan 01 1999';...
            'Jan 01 2006';'Jan 01 2009';'Jul 01 2012';'Jul 01 ...
                2015';'Jan 01 2017']...
            , 'mmm dd yyyy');
        gps_zero_datenum = datenum('1980-01-06 ...
            00:00:00.000','yyyy-mm-dd HH:MM:SS.FFF');
        days_since_gps_zero = GPS_wk*7 + GPS_ms/1e3/60/60/24;
        recv_gps_datenum = gps_zero_datenum + days_since_gps_zero;
        leapseconds = 18;
        recv_utc_datenum = recv_gps_datenum - ((leapseconds)/60/60/24);
        GPS_date=datestr(recv_utc_datenum,'mmm-dd-yyyy');
        GPS_full_time=datestr(recv_utc_datenum,'HH:MM:SS.FFF');
        tempGPS_date = datetime(GPS_date,'InputFormat','MMM-dd-yyyy');
        tempGPS_time = datetime(GPS_full_time,'Format','HH:mm:ss.S');
        tempGPS_comb = tempGPS_date + timeofday(tempGPS_time);
        GPS_final = datetime(tempGPS_comb,'Format','MMM-dd-yyyy ...
            HH:mm:ss.S');
        % Output GPS table
        GPS = [GPS_LN, GPS_time, GPS_time_out, GPS_date, GPS_full_time,...
```

```
            x, y, z, z_AGL, v_g];
        GPS_label = {'Line No','Time since boot (us)',...
            'Time from Arming (sec)','UTC Date','UTC Time','Lattitude',...
            'Longitude','Altitude (m, MSL)','Altitude (m, AGL)','Groundspeed (m/s)'};
        GPS_table = table(GPS_LN, GPS_time, GPS_time_out,tempGPS_date,...
            tempGPS_time, x, y, z, z_AGL, v_g, 'VariableNames',...
            {'Line Number','Time from boot (us)','Time from parse (sec)',...
            'UTC Date','UTC Time','Lat','Long','Altitude (m, MSL)',...
            'Altitude (m, AGL)','Groundspeed (m/s)'});

        % Parsed Attidue data output
        ATT_LN = ATT(TO_ATT:LND_ATT,1);
        ATT_time = ATT(TO_ATT:LND_ATT,2);
        ATT_time_out= (ATT_time-min(ATT_time))/1000000;
        ATT = [ATT_LN, ATT_time, ATT_time_out, rollAC, pitchAC, yawAC];
        ATT_label = {'Line No','Time since boot (us)',...
            'Time from Arming (sec)','Aircraft Roll (deg)',...
            'Aircraft Pitch (deg)','Aircraft Yaw (deg)'};
        ATT_table = table(ATT_LN,ATT_time,ATT_time_out, rollAC,...
            pitchAC, yawAC, 'VariableNames', {'Line Number',...
            'Time from boot (us)','Time from parse (sec)',...
            'Aircraft Roll (deg)','Aircraft Pitch (deg)',...
            'Aircraft Yaw (deg, magnetic)'});

        if(strcmpi(arduPilotType,'ArduCopter') | strcmpi(arduPilotType,'CopterSonde'))
        else
            % Parsed CTUN data output
            CTUN_LN = CTUN(TO_CTUN:LND_CTUN,1);
            CTUN_time = CTUN(TO_CTUN:LND_CTUN,2);
```

```matlab
        CTUN_time_out= (CTUN_time-min(CTUN_time))/1000000;
        CTUN = [CTUN_LN, CTUN_time, CTUN_time_out, v_a];
        CTUN_label = {'Line No','Time since boot (us)',...
            'Time from Arming (sec)','Airspeed (m/s)'};
        CTUN_table = table(CTUN_LN,CTUN_time,CTUN_time_out,v_a,...
            'VariableNames',{'Line Number','Time from boot (us)',...
            'Time from parse (sec)','Airspeed (m/s)'});
    end

    if(strcmpi(arduPilotType,'CopterSonde'))
        % Parsed WIND (NKF2) data output
        NKF2_LN = WIND(TO_WIND:LND_WIND,1);
        NKF2_time = WIND(TO_WIND:LND_WIND,2);
        NKF2_time_out= (NKF2_time-min(NKF2_time))/1000000;
        NKF2 = [NKF2_LN, NKF2_time, NKF2_time_out, VWN, VWE];
        NKF2_label = {'Line No','Time since boot (us)',...
            'Time from Arming (sec)','North Wind Vector ...
                (m/s)','East Wind Vector (m/s)'};
        NKF2_table = table(NKF2_LN, NKF2_time, NKF2_time_out,...
            VWN, VWE, wind, windANG, 'VariableNames',{'Line Number',...
            'Time from boot (us)','Time from parse (sec)',...
            'North Wind Vector (m/s)','East Wind Vector (m/s)',...
            'Wind Speed (m/s)', 'Wind Angle (deg)'});

        NKF1_LN = NKF1(TO_NKF1:LND_NKF1,1);
        NKF1_time = NKF1(TO_NKF1:LND_NKF1,2);
        NKF1_time_out= (NKF1_time-min(NKF1_time))/1000000;
        NKF1 = [NKF1_LN, NKF1_time, NKF1_time_out, Roll, Pitch,...
            Yaw, VN, VE, VD, PN, PE, PD];
        NKF1_label = {'Line No','Time since boot (us)','Time from ...
           Arming (sec)',...
            'Roll','Pitch','Yaw','VN','VE','VD','PN','PE','PD'};
```

```
            NKF1_table = table(NKF1_LN, NKF1_time, NKF1_time_out, Roll,...
                Pitch, Yaw, VN, VE, VD,PN,PE,PD, 'VariableNames',...
                {'Line Number','Time from boot (us)','Time from parse ...
                    (sec)',...
                'Roll (deg)','Pitch (deg)','Yaw (deg)','Velocity North ...
                    (m/s)',...
                'Velocity East (m/s)','Velocity Down (m/s)',...
                'Position North (m)','Position East (m)','Position Down ...
                    (m)'});

            IMET_LN = IMET(TO_IMET:LND_IMET,1);
            IMET_time = IMET(TO_IMET:LND_IMET,2);
            IMET_time_out= (IMET_time-min(IMET_time))/1000000;
            CSIMET_table = table(IMET_LN, IMET_time, IMET_time_out,...
                IMET(TO_IMET:LND_IMET,3), IMET(TO_IMET:LND_IMET,4),...
                IMET(TO_IMET:LND_IMET,5), IMET(TO_IMET:LND_IMET,6),...
                IMET(TO_IMET:LND_IMET,7), IMET(TO_IMET:LND_IMET,8),...
                IMET(TO_IMET:LND_IMET,9), IMET(TO_IMET:LND_IMET,10),...
                IMET(TO_IMET:LND_IMET,11), IMET(TO_IMET:LND_IMET,12), ...
                    IMET(TO_IMET:LND_IMET,13), ...
                    IMET(TO_IMET:LND_IMET,14), ...
                    IMET(TO_IMET:LND_IMET,15), ...
                    IMET(TO_IMET:LND_IMET,16),'VariableNames',{'LineNo','TimeUS','Time .
                    from parse ...
                    (sec)','RH1','RH2','RH3','RH4','T1','T2','T3','T4','Health1','Health

        else
            % Parsed NKF2 data output
            NKF2_LN = NKF2(TO_NKF2:LND_NKF2,1);
            NKF2_time = NKF2(TO_NKF2:LND_NKF2,2);
            NKF2_time_out= (NKF2_time-min(NKF2_time))/1000000;
            NKF2 = [NKF2_LN, NKF2_time, NKF2_time_out, VWN, VWE];
```

```
        NKF2_label = {'Line No','Time since boot (us)',...
            'Time from Arming (sec)','North Wind Vector ...
               (m/s)','East Wind Vector (m/s)'};
        NKF2_table = table(NKF2_LN, NKF2_time, NKF2_time_out,...
            VWN, VWE, wind,windANG,'VariableNames',{'Line Number',...
            'Time from boot (us)','Time from parse (sec)',...
            'North Wind Vector (m/s)','East Wind Vector (m/s)',...
            'Wind Speed (m/s)','Wind Angle (deg)'});

        NKF1_LN = NKF1(TO_NKF1:LND_NKF1,1);
        NKF1_time = NKF1(TO_NKF1:LND_NKF1,2);
        NKF1_time_out= (NKF1_time-min(NKF1_time))/1000000;
        NKF1 = [NKF1_LN, NKF1_time, NKF1_time_out, rollEKF,...
            pitchEKF, yawEKF, vnEKF, veEKF, vdEKF, pnEKF, peEKF, ...
               pdEKF];
        NKF1_label = {'Line No','Time since boot (us)',...
            'Time from Arming (sec)','Roll','Pitch','Yaw',...
            'VN','VE','VD','PN','PE','PD'};
        NKF1_table = table(NKF1_LN, NKF1_time, NKF1_time_out,...
            rollEKF, pitchEKF, yawEKF, vnEKF, veEKF, vdEKF,...
            pnEKF, peEKF, pdEKF, 'VariableNames',{'Line Number',...
            'Time from boot (us)','Time from parse (sec)','Roll',...
            'Pitch','Yaw','VN', 'VE', 'VD','PN','PE','PD'});

    end

    % Parsed RCOU data output
    RCOU_LN = RCOU(TO_RCOU:LND_RCOU,1);
    RCOU_time = RCOU(TO_RCOU:LND_RCOU,2);
    RCOU_pitch = RCOU(TO_RCOU:LND_RCOU,3);
    RCOU_roll = RCOU(TO_RCOU:LND_RCOU,4);
    RCOU_thr = RCOU(TO_RCOU:LND_RCOU,5);
```

```
        RCOU_yaw = RCOU(TO_RCOU:LND_RCOU,6);
        RCOU_time_out = (RCOU_time-min(RCOU_time))/1000000;
        RCOU = [RCOU_LN, RCOU_time, RCOU_time_out, RCOU_pitch,...
            RCOU_roll, RCOU_thr, RCOU_yaw];
        RCOU_label = {'Line No','Time since boot (us)',...
            'Time from Arming (sec)','C1 - Pitch PWM',...
            ' C2 - Roll PWM','C3 - Throttle PWM','C4 - Yaw PWM'};
        RCOU_table = table(RCOU_LN, RCOU_time, RCOU_time_out,...
            RCOU_pitch, RCOU_roll, thr,RCOU_thr, RCOU_yaw,...
            'VariableNames',{'Line Number','Time from boot (us)',...
            'Time from parse (sec)','Pitch PWM','Roll PWM','Throttle ...
                (%)',...
            'Throttle PWM','Yaw PWM'});

        % Parsed IMU data output
        IMU_LN = IMU(TO_IMU:LND_IMU,1);
        IMU_time = IMU(TO_IMU:LND_IMU,2);
        IMU_GyrX = IMU(TO_IMU:LND_IMU,3);
        IMU_GyrY = IMU(TO_IMU:LND_IMU,4);
        IMU_GyrZ = IMU(TO_IMU:LND_IMU,5);
        IMU_AccX = IMU(TO_IMU:LND_IMU,6);
        IMU_AccY = IMU(TO_IMU:LND_IMU,7);
        IMU_AccZ = IMU(TO_IMU:LND_IMU,8);
        IMU_time_out = (IMU_time-min(IMU_time))/1000000;
        IMU = [IMU_LN, IMU_time, IMU_time_out, IMU_GyrX, IMU_GyrY,...
            IMU_GyrZ, IMU_AccX, IMU_AccY, IMU_AccZ];
        IMU_label = {'Line No','Time since boot (us)','Time from parse ...
            (sec)',...
            'X Gyro rotation (  /sec)','Y Gyro rotation (  /sec)','Z ...
                Gyro rotation (  /sec)',...
            'X Acceleration (  /sec/sec)','Y Acceleration ...
                (  /sec/sec)','Z Acceleration (  /sec/sec)'};
```

```
        IMU_table = table(IMU_LN,IMU_time,IMU_time_out,IMU_GyrX, ...
            IMU_GyrY,...
            IMU_GyrZ, IMU_AccX, IMU_AccY, IMU_AccZ,'VariableNames',...
            {'Line Number','Time from boot (us)','Time from parse ...
                (sec)',...
            'X Gyro rotation (  /sec)','Y Gyro rotation (  /sec)',...
            'Z Gyro rotation (  /sec)','X Acceleration (  /sec/sec)',...
            'Y Acceleration (  /sec/sec)','Z Acceleration (  /sec/sec)'});

        % Parsed Pixhawk barometric data
        BARO_LN = BARO(TO_BARO:LND_BARO,1);
        BARO_time = BARO(TO_BARO:LND_BARO,2);
        BARO_time_out = (BARO_time-min(BARO_time))/1000000;
        BARO_alt = BARO(TO_BARO:LND_BARO,3);
        BARO_press = BARO(TO_BARO:LND_BARO,4)/100;
        BARO = [BARO_LN, BARO_time, BARO_time_out, BARO_press, BARO_alt];
        BARO_label = {'Line No','Time since boot (us)',...
            'Time from parse (sec)','Barometric pressure (mbar)',...
            'Barometric Altitude (m, AGL)'};
        BARO_table = table(BARO_LN,BARO_time,BARO_time_out,BARO_press,...
            BARO_alt,'VariableNames',{'Line Number','Time from boot ...
                (us)',...
            'Time from parse (sec)','Barometric pressure (mbar)',...
            'Barometric Altitude (m, AGL)'});

        % Get the name of the intput.mat file and save as input_parsed.mat
        baseFileName = sprintf('%s_Parsed.mat', baseNameNoExtDFL);
        fullParsedMatFileName = fullfile(folderDFL, baseFileName);
        app.LocationofOutputFilesEditField.Value = fullParsedMatFileName;
        % Save file with parsed data as the original filename plus the ...
            added portion

```

```
        if(strcmpi(arduPilotType,'ArduCopter'))
            save(fullParsedMatFileName,'ATT_table','GPS_table','NKF2_table',...
                'RCOU_table','BARO_table','IMU_table');
        elseif(strcmpi(arduPilotType,'CopterSonde'))
            save(fullParsedMatFileName,'ATT_table','GPS_table','NKF2_table',...
                'RCOU_table','BARO_table','IMU_table','CSIMET_table');
        else
            save(fullParsedMatFileName,'ATT_table','GPS_table','CTUN_table',...
                'NKF2_table','RCOU_table','BARO_table','IMU_table');
        end
    end
    %% iMet Data Parsing and Output
    if(strcmpi(DFL_NewOld,'New'))
        if (strcmpi(iMetValue,'Yes'))

            % Get the name of the file that the user wants to use.
            defaultFileNameiMet = fullfile(startingFolderDFL, '*.csv');
            [baseFileNameiMet, folderiMet] = ...
                uigetfile(defaultFileNameiMet,...
                'Select an iMet .CSV file');
            if baseFileNameiMet == 0
                % User clicked the Cancel button.
                return;
            end

            % Get the name of the input .mat file.
            fullInputMatFileNameiMet = fullfile(folderiMet, ...
                baseFileNameiMet);
            % Get filename without the extension, used by Save Function
            [¬, baseNameNoExtiMet, ¬] = fileparts(baseFileNameiMet);
            % Load file in
            iMetData_temp = readtable(fullInputMatFileNameiMet);
```

```
        iMetData = iMetData_temp(2:end,:);
        clear iMetData_temp

        dataLen = width(iMetData);

        if(dataLen == 46)
            % Convert CSV data to general table to datetime array
            iMet_date_ref = iMetData(:,40);
            iMet_date_conv = table2array(iMet_date_ref);
            iMet_date = datetime(iMet_date_conv,'InputFormat',...
                'yyyy/MM/dd','Format','MMM-dd-yyyy');
            iMet_time_ref = iMetData(:,41);
            iMet_time_conv = table2array(iMet_time_ref);
            iMet_time = datetime(datevec(iMet_time_conv),'Format',...
                'HH:mm:ss');
            Pix_time = GPS(:,3);

        elseif(dataLen == 47)
            % Convert CSV data to general table to datetime array
            iMet_date_ref = iMetData(:,41);
            iMet_date_conv = table2array(iMet_date_ref);
            iMet_date = datetime(iMet_date_conv,'InputFormat',...
                'yyyy/MM/dd','Format','MMM-dd-yyyy');
            iMet_time_ref = iMetData(:,42);
            iMet_time_conv = table2array(iMet_time_ref);
            iMet_time = datetime(datevec(iMet_time_conv),'Format',...
                'HH:mm:ss');
            Pix_time = GPS(:,3);
        end

        % Convert datetime arrays into datetime vectors
        [iM_y, iM_m, iM_d] = datevec(iMet_date(:,1));
```

```
[ no, no, no,iM_h, iM_M, iM_s] = datevec(iMet_time);
iM_vec = [iM_y iM_m iM_d iM_h iM_M iM_s];

GPS_vec = datevec(GPS_final);

% Convert datetime vectors datetime serials
GPS_serial = datenum(GPS_vec);
iMet_serial = datenum(iM_vec);

% Find iMet data at start and end of Pixhawk parsing
TO_iMet = find(iMet_serial(:)≥min(GPS_serial),1,'first');
LND_iMet = find(iMet_serial(:)≥max(GPS_serial),1,'first');

% Parse iMet data
iM_red = iM_vec(TO_iMet:LND_iMet,:);
iMet_serial = iMet_serial(TO_iMet:LND_iMet);

% Add Pixhawk time since arming variable
iter = 1;
for i=1:length(iMet_serial)
    for j=iter:length(GPS_serial)
        if(iMet_serial(i)==GPS_serial(j))
            iM_Pix(i,1)=GPS_time_out(j);
            iter = j;
        end
    end
end

for row = 1:length(iM_Pix)
    if iM_Pix(row,1) == 0
        iM_Pix(row,1) = (iM_Pix(row-1,1)+iM_Pix(row+1,1))/2;
    end
```

```
        end

        if(dataLen == 46)
            i=TO_iMet:LND_iMet;
            iM_time_temp(:,1) = iMet_time(i,1);
            iM_date_temp(:,1) = iMet_date(i,1);
            iM_pres_temp(:,1) = iMetData(i,37);
            iM_temp_temp(:,1) = iMetData(i,38);
            iM_humid_temp(:,1) = iMetData(i,39);
            iM_lat_temp(:,1) = iMetData(i,42);
            iM_long_temp(:,1) = iMetData(i,43);
            iM_alt_temp(:,1) = iMetData(i,44);
            iM_sat_temp(:,1) = iMetData(i,45);
        elseif(dataLen == 47)
            i=TO_iMet:LND_iMet;
            iM_time_temp(:,1) = iMet_time(i,1);
            iM_date_temp(:,1) = iMet_date(i,1);
            iM_pres_temp(:,1) = iMetData(i,37);
            iM_temp_temp(:,1) = iMetData(i,38);
            iM_humid_temp(:,1) = iMetData(i,39);
            iM_humid_temp_t(:,1) = iMetData(i,40);
            iM_lat_temp(:,1) = iMetData(i,43);
            iM_long_temp(:,1) = iMetData(i,44);
            iM_alt_temp(:,1) = iMetData(i,45);
            iM_sat_temp(:,1) = iMetData(i,46);
        end

        len1 = size(iM_Pix,1);
        len2 = size(iM_time_temp,1);

        if(dataLen == 46)
            if(len1>len2)
```

```
                iM_Pix = iM_Pix(1:len2);
                iM_time = datestr(iM_time_temp(1:len2,1),'HH:MM:ss');
                iM_date = ...
                   datestr(iM_date_temp(1:len2,1),'mmm-dd-yyyy');
                iM_pres = table2array(iM_pres_temp(1:len2,1));
                iM_temp = table2array(iM_temp_temp(1:len2,1));
                iM_humid = table2array(iM_humid_temp(1:len2,1));
                iM_lat = table2array(iM_lat_temp(1:len2,1));
                iM_long = table2array(iM_long_temp(1:len2,1));
                iM_alt = table2array(iM_alt_temp(1:len2,1));
                iM_sat = table2array(iM_sat_temp(1:len2,1));
            elseif(len2>len1)
                iM_Pix = iM_Pix(1:len1);
                iM_time = datestr(iM_time_temp(1:len1,1),'HH:MM:ss');
                iM_date = ...
                   datestr(iM_date_temp(1:len1,1),'mmm-dd-yyyy');
                iM_pres = table2array(iM_pres_temp(1:len1,1));
                iM_temp = table2array(iM_temp_temp(1:len1,1));
                iM_humid = table2array(iM_humid_temp(1:len1,1));
                iM_lat = table2array(iM_lat_temp(1:len1,1));
                iM_long = table2array(iM_long_temp(1:len1,1));
                iM_alt = table2array(iM_alt_temp(1:len1,1));
                iM_sat = table2array(iM_sat_temp(1:len1,1));
            else
                iM_Pix = iM_Pix(1:len1);
                iM_time = datestr(iM_time_temp(:,1),'HH:MM:ss');
                iM_date = datestr(iM_date_temp(:,1),'mmm-dd-yyyy');
                iM_pres = table2array(iM_pres_temp(:,1));
                iM_temp = table2array(iM_temp_temp(:,1));
                iM_humid = table2array(iM_humid_temp(:,1));
                iM_lat = table2array(iM_lat_temp(:,1));
                iM_long = table2array(iM_long_temp(:,1));
```

```
                iM_alt = table2array(iM_alt_temp(:,1));
                iM_sat = table2array(iM_sat_temp(:,1));
            end

            iMet_table = table(iM_Pix, iM_date, iM_time, iM_pres,...
                iM_temp, iM_humid, iM_lat, iM_long, iM_alt, iM_sat,...
                'VariableNames',{'Time from Arming (sec)','Date ...
                   UTC',...
                'Time UTC','Barometric Pressure (hPa)',...
                'Air Temp ( C )','Relative Humidity (%)',...
                'GPS Lat','GPS Long','GPS Alt (m)','Sat Count'});
            save(fullParsedMatFileName,'iMet_table','-append');

        elseif(dataLen == 47)
            if(len1>len2)
                iM_Pix = iM_Pix(1:len2);
                iM_time = datestr(iM_time_temp(1:len2,1),'HH:MM:ss');
                iM_date = ...
                   datestr(iM_date_temp(1:len2,1),'mmm-dd-yyyy');
                iM_pres = table2array(iM_pres_temp(1:len2,1));
                iM_temp = table2array(iM_temp_temp(1:len2,1));
                iM_humid = table2array(iM_humid_temp(1:len2,1));
                iM_humid_t = table2array(iM_humid_temp_t(1:len2,1));
                iM_lat = table2array(iM_lat_temp(1:len2,1));
                iM_long = table2array(iM_long_temp(1:len2,1));
                iM_alt = table2array(iM_alt_temp(1:len2,1));
                iM_sat = table2array(iM_sat_temp(1:len2,1));
            elseif(len2>len1)
                iM_Pix = iM_Pix(1:len1);
                iM_time = datestr(iM_time_temp(1:len1,1),'HH:MM:ss');
                iM_date = ...
                   datestr(iM_date_temp(1:len1,1),'mmm-dd-yyyy');
```

```
                iM_pres = table2array(iM_pres_temp(1:len1,1));
                iM_temp = table2array(iM_temp_temp(1:len1,1));
                iM_humid = table2array(iM_humid_temp(1:len1,1));
                iM_humid_t = table2array(iM_humid_temp_t(1:len1,1));
                iM_lat = table2array(iM_lat_temp(1:len1,1));
                iM_long = table2array(iM_long_temp(1:len1,1));
                iM_alt = table2array(iM_alt_temp(1:len1,1));
                iM_sat = table2array(iM_sat_temp(1:len1,1));
            else
                iM_Pix = iM_Pix(1:len1);
                iM_time = datestr(iM_time_temp(:,1),'HH:MM:ss');
                iM_date = datestr(iM_date_temp(:,1),'mmm-dd-yyyy');
                iM_pres = table2array(iM_pres_temp(:,1));
                iM_temp = table2array(iM_temp_temp(:,1));
                iM_humid = table2array(iM_humid_temp(:,1));
                iM_humid_t = table2array(iM_humid_temp_t(:,1));
                iM_lat = table2array(iM_lat_temp(:,1));
                iM_long = table2array(iM_long_temp(:,1));
                iM_alt = table2array(iM_alt_temp(:,1));
                iM_sat = table2array(iM_sat_temp(:,1));
            end

            iMet_table = table(iM_Pix, iM_date, iM_time, iM_pres,...
                iM_temp, iM_humid, iM_humid_t, iM_lat, iM_long, ...
                   iM_alt,...
                iM_sat,'VariableNames',{'Time from Arming (sec)',...
                'Date UTC','Time UTC','Barometric Pressure (hPa)',...
                'Air Temp ( C )','Relative Humidity (%)',...
                'Humidity Temperature ( C )','GPS Lat','GPS Long',...
                'GPS Alt (m)','Sat Count'});
            save(fullParsedMatFileName,'iMet_table','-append');
        end
```

```
            fig3=figure(3);
            fig3.Name = 'Parsed iMet data.';
            set(fig3,'defaultLegendAutoUpdate','off');
            yyaxis left
            plt = plot(iM_Pix(:), iM_temp(:),'y-',iM_Pix(:),iM_humid,'c-');
            title('Temp, Humidity, and Pressure vs Time')
            xlabel('Time (ms)');
            ylabel('Temp ( C ) and Humidity (%)');

            yyaxis right
            plt = plot(iM_Pix(:), iM_pres(:),'k-');
            ylabel('Pressure (hPa)');
            legend({'iMet Temp','iMet Humid','iMet ...
                Pres'},'Location','southeast')

            if(strcmpi(sensorOut,'Yes'))
                % Get the name of the input.mat file and save as ...
                    input_Parsed_TPH.csv
                baseFileName = sprintf('%s_iMet.csv', baseNameNoExtDFL);
                fullOutputMatFileName = fullfile(folderDFL, baseFileName);
                % Write data to text file
                writetable(iMet_table, fullOutputMatFileName);
            end
        end

    elseif(strcmpi(DFL_NewOld,'Old'))
        if(exist('iMet_table','var'))
            if (strcmpi(iMetValue,'Yes'))

                fig3=figure(3);
                fig3.Name = 'Parsed iMet data.';
```

```
                set(fig3,'defaultLegendAutoUpdate','off');
                yyaxis left
                plt = plot(iMet_table{:,1}, iMet_table{:,5},'y-',...
                    iMet_table{:,1},iMet_table{:,6},'c-');
                title('Temp, Humidity, and Pressure vs Time')
                xlabel('Time (ms)');
                ylabel('Temp ( C ) and Humidity (%)');

                yyaxis right
                plt = plot(iMet_table{:,1}, iMet_table{:,4},'k-');
                ylabel('Pressure (hPa)');
                legend({'iMet Temp','iMet Humid','iMet Pres'},'Location','southeast')

                if(strcmpi(sensorOut,'Yes'))
                    % Get the name of the input.mat file and save as input_Parsed_TPH.csv
                    baseFileName = sprintf('%s_iMet_NS.csv', baseNameNoExtDFL);
                    fullOutputMatFileName = fullfile(folderDFL, baseFileName);
                    % Write data to text file
                    writetable(iMet_table, fullOutputMatFileName);
                end
            end
        end
    end
    %% 5HP Data Parsing and Output
    if (strcmpi(mhpValue,'Yes') | strcmpi(tphValue,'Yes'))
        if(strcmpi(DFL_NewOld,'New'))
            % Get the name of the file that the user wants to use.
            defaultFileNameMHP = fullfile(startingFolderDFL, '*.csv');
```

```
        [baseFileNameMHP, folderMHP] = uigetfile(defaultFileNameMHP,...
            'Select a 5HP/TPH .CSV file');
        if baseFileNameMHP == 0
            % User clicked the Cancel button.
            return;
        end

        % Get the name of the input .mat file.
        fullInputMatFileNameMHP = fullfile(folderMHP, baseFileNameMHP);
        % Get filename without the extension, used by Save Function
        [¬, baseNameNoExtMHP, ¬] = fileparts(baseFileNameMHP);

        data = readmatrix(fullInputMatFileNameMHP);
        data(isnan(data)) = -1;

        list = {'Probe 1', 'Probe 2', 'Probe 3', 'Metal Probe'};
        [Probe, tf] = listdlg('ListString', list, ...
           'SelectionMode','single');

        if Probe==1
            %%
            Probe_matrix =reshape([-45 5.4503245929122812 -40 ...
               3.6344070134585587 ...
                -35 2.6910078916419944 -30 2.0741227786203615 ...
                -25 1.6177891317119037 -20 1.3047139241276557 ...
                -15 0.99765981580647944 -10 0.72488341948754176 ...
                -5 0.42287858693961322 0 0.11269996848776841 5 ...
                -0.18844278909812986 10 -0.51552305940602694 15 ...
                -0.82527532291594985 20 -1.0827862740935248 25 ...
                -1.3747195966697383 30 -1.7399669009211871 35 ...
                -2.2261994229329716 40 -2.83647895638291 45 ...
                   -3.7775455967008882 ...
```

```
-45 5.6088602743478519 -40 3.8471216439542708 ...
-35 2.758014522232052 -30 2.1393612470974079 -25 ...
1.6846516032825758 -20 1.2812487641404717 -15 ...
0.99769242818437875 -10 0.7108466648491224 -5 ...
0.39522993924508154 0 0.055666210297281971 5 ...
    -0.26040468531481231 ...
10 -0.60604077802968226 15 -0.91007141723872353 ...
20 -1.2115256139194885 25 -1.5914907297890892 ...
30 -2.0768170988494874 35 -2.7082025856213936 ...
40 -3.6672797432896682 45 -5.11308906747139 -45 ...
0.40153363672050385 -40 0.58252778411973083 -35 ...
0.76207165540900135 -30 0.91124757982052851 -25 ...
1.0297461281014839 -20 1.1162791106122567 -15 ...
1.1602402228108957 -10 1.1682637817844033 -5 ...
    1.1717460580991859 ...
0 1.1705980442512705 5 1.1742807474113173 10 ...
    1.1750681375621821 ...
15 1.1761267609636077 20 1.1541350590812145 25 ...
1.0909288620176854 30 0.99477056787113616 35 ...
    0.8677013434826869 ...
40 0.71724821586753418 45 0.55890366509478218 ...
-45 0.45693864133494283 -40 0.63502634767094046 ...
-35 0.81900446405055538 -30 0.98505044428739341 ...
-25 1.0967311790193095 -20 1.1760591378284164 ...
-15 1.2105339434476543 -10 1.225378332775521 -5 ...
1.2208141810618345 0 1.2167930824763777 5 ...
    1.213004218757056 ...
10 1.202293865820901 15 1.1664968024605715 20 ...
1.1003915504970527 25 0.99694948648152915 30 ...
    0.86274285528505079 ...
35 0.70827366374370193 40 0.5425004640691633 45 ...
0.3714015525546091], 2, 19, 4);
```

```
        %%
    end
    if Probe==2
        %%
        Probe_matrix= reshape([-45 8.0413125757286483 -40 4.595840988208745 -35 ...
            3.1960983613503395 -30 2.3438686451680391 -25 ...
            1.7998009251294775 -20 1.3568843785915969 -15 ...
            1.0417025439286454 -10 0.6719271337912911 -5 0.31594859127806313 ...
            0 -0.027775383920766113 5 -0.38525870149950231 ...
            10 -0.76967960249393741 15 -1.0870073516266328 ...
            20 -1.378333118288662 25 -1.7338076966226417 30 ...
            -2.1416605956464423 35 -2.672348394280156 40 -3.4957062360550872 ...
            45 -4.9947263023335022 -45 5.4644672199708184 ...
            -40 3.7051459239570832 -35 2.7802502582437754 ...
            -30 2.1909163813094055 -25 1.745552975533075 -20 ...
            1.3193857267226774 -15 0.989568212585923 -10 0.68976185302006032 ...
            -5 0.34537219692610427 0 0.00070104058184836159 ...
            5 -0.35188215190568023 10 -0.70334097827774844 ...
            15 -1.0151026074394103 20 -1.3372013554379576 ...
            25 -1.7119408626720383 30 -2.1780237220699625 ...
            35 -2.7495578563807559 40 -3.577709672348806 45 ...
            -4.9039267737627332 -45 0.27126131923904334 -40 ...
            0.438829235484768 -35 0.6069713454411777 -30 0.76007586890568246 ...
            -25 0.8757553537646996 -20 0.97230935336276991 ...
            -15 1.0163124604875853 -10 1.0494565940075169 ...
            -5 1.0440081543412782 0 1.0521140361945402 5 1.0617346368847278 ...
```

```
                10 1.0640740038270755 15 1.0530416652892758 20 ...
                1.0150305427048212 25 0.93933046605058423 30 ...
                    0.84632064730305567 ...
                35 0.71412230618293648 40 0.56257867097066894 ...
                45 0.4088708445506693 -45 0.40039661393475073 ...
                -40 0.56253456955569414 -35 0.71741173034906691 ...
                -30 0.83895154863814991 -25 0.945508433447679 ...
                -20 1.048989338380182 -15 1.1079676863677939 -10 ...
                1.1217507631454158 -5 1.1213365893922569 0 ...
                    1.1102349448662681 ...
                5 1.1103023687637184 10 1.0907471029595337 15 ...
                1.0731456995653772 20 1.0326480971143717 25 ...
                    0.96008313343852325 ...
                30 0.85664753210489064 35 0.74804723263594852 ...
                40 0.60691269332640829 45 0.47048729685494273], 2, ...
                    19, 4);
            %%
        end
        if Probe==3
            %%
            Probe_matrix= reshape([-45 8.0413125757286483 -40 ...
                4.595840988208745 -35 ...
                3.1960983613503395 -30 2.3438686451680391 -25 ...
                1.7998009251294775 -20 1.3568843785915969 -15 ...
                1.0417025439286454 -10 0.6719271337912911 -5 ...
                    0.31594859127806313 ...
                0 -0.027775383920766113 5 -0.38525870149950231 ...
                10 -0.76967960249393741 15 -1.0870073516266328 ...
                20 -1.378333118288662 25 -1.7338076966226417 30 ...
                -2.1416605956464423 35 -2.672348394280156 40 ...
                    -3.4957062360550872 ...
                45 -4.9947263023335022 -45 5.4644672199708184 ...
```

```
-40 3.7051459239570832 -35 2.7802502582437754 ...
-30 2.1909163813094055 -25 1.745552975533075 -20 ...
1.3193857267226774 -15 0.989568212585923 -10 ...
   0.68976185302006032 ...
-5 0.34537219692610427 0 0.00070104058184836159 ...
5 -0.35188215190568023 10 -0.70334097827774844 ...
15 -1.0151026074394103 20 -1.3372013554379576 ...
25 -1.7119408626720383 30 -2.1780237220699625 ...
35 -2.7495578563807559 40 -3.577709672348806 45 ...
-4.9039267737627332 -45 0.27126131923904334 -40 ...
0.438829235484768 -35 0.6069713454411777 -30 ...
   0.76007586890568246 ...
-25 0.8757553537646996 -20 0.97230935336276991 ...
-15 1.0163124604875853 -10 1.0494565940075169 ...
-5 1.0440081543412782 0 1.0521140361945402 5 ...
   1.0617346368847278 ...
10 1.0640740038270755 15 1.0530416652892758 20 ...
1.0150305427048212 25 0.93933046605058423 30 ...
   0.84632064730305567 ...
35 0.71412230618293648 40 0.56257867097066894 ...
45 0.4088708445506693 -45 0.40039661393475073 ...
-40 0.56253456955569414 -35 0.71741173034906691 ...
-30 0.83895154863814991 -25 0.945508433447679 ...
-20 1.048989338380182 -15 1.1079676863677939 -10 ...
1.1217507631454158 -5 1.1213365893922569 0 ...
   1.1102349448662681 ...
5 1.1103023687637184 10 1.0907471029595337 15 ...
1.0731456995653772 20 1.0326480971143717 25 ...
   0.96008313343852325 ...
30 0.85664753210489064 35 0.74804723263594852 ...
40 0.60691269332640829 45 0.47048729685494273], 2, ...
   19, 4);
```

```
        %%
    end
    if Probe==4
        %%
        Probe_matrix=reshape([-45 6.8270286994312324 -40 ...
            3.9272264704959947 ...
                -35 2.7098630146680343 -30 2.0311527134073506 ...
                -25 1.5648399326122149 -20 1.2097555239893774 ...
                -15 0.99401689018022032 -10 0.7109304970157786 ...
                -5 0.39798725583059297 0 0.10610929666786524 5 ...
                -0.16157327211291675 10 -0.45721492832552874 ...
                    15 ...
                -0.73051006787347961 20 -1.0206346484460449 25 ...
                -1.3758360236897476 30 -1.7831603701441343 35 ...
                -2.377357311604094 40 -3.464604213809888 45 ...
                    -5.902768693651109 ...
                -45 7.4133939397891409 -40 4.1198674730828735 ...
                -35 2.7770897996831905 -30 2.0145359604997175 ...
                -25 1.5078827091567195 -20 1.1318223410844281 ...
                -15 0.86433128769217948 -10 0.5558516056739059 ...
                -5 0.26359452098651637 0 -0.028656763433599269 ...
                5 -0.34839374505470783 10 -0.65948481869867559 ...
                15 -0.96825549560038959 20 -1.3025459837101847 ...
                25 -1.7237901125815069 30 -2.2978387224876857 ...
                35 -3.2329872283805456 40 -5.1776511809601873 ...
                45 -13.031656173662968 -45 0.32254048615943415 ...
                -40 0.52205954626286333 -35 ...
                    0.68966672023071152 ...
                -30 0.831269693794436 -25 0.9490842976748094 ...
                    -20 ...
                1.0427975635210041 -15 1.0930205988684738 -10 ...
```

```
                        1.1249929311810012 -5 1.1393737524485081 0 ...
                            1.1365874970433751 ...
                        5 1.1398409797574214 10 1.1325220268485772 15 ...
                        1.1008008398450571 20 1.0435222498791885 25 ...
                            0.96164376880806623 ...
                        30 0.84143329041748371 35 0.69288464852455156 ...
                        40 0.52373434130229513 45 0.32906938976100375 ...
                        -45 0.290471428675032 -40 0.50325598712500985 ...
                        -35 0.68854268208577729 -30 ...
                            0.84765291448530922 ...
                        -25 0.97823914779086751 -20 1.0667329519213851 ...
                        -15 1.1137584193430363 -10 1.1390572418989386 ...
                        -5 1.1394729063509146 0 1.1372390682609228 5 ...
                            1.1429838322264543 ...
                        10 1.1285412771221324 15 1.0857994109712712 20 ...
                        1.0227196623145753 25 0.90281324987619327 30 ...
                            0.75359921806475 ...
                        35 0.5861378670636509 40 0.38887273357229107 ...
                            45 ...
                        0.15966154155110446], 2, 19, 4);
                %%
        end
        nrows = length(data(:,1));
        ncols = length(data(1,:));

        rho=1.197; %kg/m3 Lets move this down (calc it)

        CTUNTemp = CTUN;

        Pixcount=0;
        SScount=0;

```

```
        i=1:nrows; % vectorize instead of for loop (speed)

        time = data(i,1);
        PitotB1 = data(i,2);
        PitotB2 = data(i,3);
        AlphaB1 = data(i,6);
        AlphaB2 = data(i,7);
        BetaB1 = data(i,10);
        BetaB2 = data(i,11);
        H1 = data(i,15);
        T1 = data(i,16);
        H2 = data(i,19);
        T2 = data(i,20);
        H3 = data(i,23);
        T3 = data(i,24);
        UnixT = data(i,26);
        PixT = data(i,27);

        PitotCount = ((PitotB1*256)+PitotB2);
        AlphaCount =  ((AlphaB1*256)+AlphaB2);
        BetaCount  =  ((BetaB1*256)+BetaB2);

        Pitot_psi=((PitotCount-1638)*(1+1))/(14745-1638)-1;
        Alpha_psi=((AlphaCount-1638)*(1+1))/(14745-1638)-1;
        Beta_psi=((BetaCount-1638)*(1+1))/(14745-1638)-1;

        Pitot_pa=Pitot_psi*6894.74;
        Alpha_pa=Alpha_psi*6894.74;
        Beta_pa=Beta_psi*6894.74;

        %Cp Calc
        CP_a=Alpha_pa./Pitot_pa;
```

```
            CP_b=Beta_pa./Pitot_pa;

            % Calculate alpha and beta probe values
            Alpha=interp1(Probe_matrix(2,:,1), Probe_matrix(1,:,1),...
                CP_a(i), 'linear', 45); %just doing 1d interp for now ...
                    until more speeds ran
            Beta=interp1(Probe_matrix(2,:,2), Probe_matrix(1,:,2),...
            CP_b(i), 'linear', 45);

            CP_pitot1 = interp1(Probe_matrix(1,:,4),...
                Probe_matrix(2,:,4), Beta(i), 'makima', .5);
            CP_pitot2 = interp1(Probe_matrix(1,:,3),...
                Probe_matrix(2,:,3), Alpha(i),'makima', .5);

            for i=1:nrows
                UnixTime = UnixT(i);
                Temp = T1(i);
                if(UnixTime≠-1)
                    Pixcount=Pixcount+1;
                    PixData(Pixcount,1)=time(i);    % Sensor board time
                    PixData(Pixcount,2)=UnixTime;   % Unix time from Pix GPS
                    PixData(Pixcount,3)=PixT(i);    % Pixhawk board time
                end

                if(Temp≠-1)
                    SScount=SScount+1;
                    THSense(SScount,1)=time(i);  % Sensor board time
                    THSense(SScount,2)=-1;     % Will become Pixhawk ...
                        board time
                    THSense(SScount,3)=T1(i);
                    THSense(SScount,4)=T2(i);
                    THSense(SScount,5)=T3(i);
```

```
                    THSense(SScount,6)=H1(i);
                    THSense(SScount,7)=H2(i);
                    THSense(SScount,8)=H3(i);
                end

                if CP_pitot1(i) > CP_pitot2(i)
                    CP_pitot(i) = CP_pitot2(i);
                else
                    CP_pitot(i) = CP_pitot1(i);
                end

                U(i) = ((2/rho)*(abs(Pitot_pa(i)/CP_pitot(i)))) .^.5;

            end

            u = U'.*cosd(Alpha).*cosd(Beta);
            v = U'.*sind(Beta);
            w = U'.*sind(Alpha).*cosd(Beta);
            total = sqrt(abs(u).^2 + abs(v).^2 + abs(2).^2);

            for i=1:nrows
            MHPData(i,1)=time(i);             % Sensor board time
            MHPData(i,2)=-1;               % Will become Pixhawk board time
            MHPData(i,3)=u(i);               % probe u velocity
            MHPData(i,4)=v(i);               % probe v velocity
            MHPData(i,5)=w(i);               % probe w velocity
            MHPData(i,6)=Alpha(i);           % alpha angle of the probe
            MHPData(i,7)=Beta(i);            % beta angle of the probe
            MHPData(i,8)=0;
            MHPData(i,9)=0;
%            MHPData(i,8)=Alpha_MA(i);
%            MHPData(i,9)=Beta_MA(i);
```

```
MHPData(i,10)=total(i);         % calculated total velocity ...
    of the probe (body frame)
MHPData(i,11)=0;
MHPData(i,12)=0;
MHPData(i,13)=0;
MHPData(i,14)=0;
end

% calculation of difference between teensy time and pixtime
if (PixData(:,3) == -1)
    serialGPS = posixtime(GPS_final);
    tempPixTime = GPS_table(:,2)/100000;

    for i=length(GPS_table(:,1))
        offset(i) = serialGPS(i) - tempPixTime(i);
    end
    AvOffset = mean(offset);
    PixData(:,3) = PixData(:,2) - AvOffset(:);
end

for i=1:length(PixData(:,1))
    PixOff(i)=PixData(i,3)-PixData(i,1);
    GPS_off(i)=PixData(i,2)-PixData(i,1)/1000;
end

PixAv=mean(PixOff);
GPS_av = mean(GPS_off);

leapseconds_unix = 28;

% Offsets between board time and Pix time
MHPData(:,2)=round((MHPData(:,1)+PixAv),0);
```

```
        MHPData_Unix=round((MHPData(:,1)/1000)+GPS_av,1);

        %% REST OF CODE

        MHP_DateTime=datetime(MHPData_Unix,...
            'ConvertFrom','posixTime','Format','MMM-dd-yyyy ...
               HH:mm:ss.S');
        MHP_Date=datestr(MHP_DateTime,'mmm-dd-yyyy');
        MHP_Time=datestr(MHP_DateTime,'HH:MM:SS.FFF');

        if(exist('THSense','var'))
            THSense(:,2)=round((THSense(:,1)+PixAv),0);
            THSense_Unix=round((THSense(:,1)/1000)+GPS_av,1);

            TH_DateTime=datetime(THSense_Unix,'ConvertFrom',...
                'posixTime','Format','MMM-dd-yyyy HH:mm:ss.S');
            TH_Date=datestr(TH_DateTime,'mmm-dd-yyyy');
            TH_Time=datestr(TH_DateTime,'HH:MM:SS.FFF');
        end

        if (min(MHP_DateTime) < min(GPS_final))
            TO_MHP = find(MHP_DateTime(:)≥min(GPS_final),1,'first');
        else
            TO_MHP = 1;
        end

        if(exist('THSense','var'))
            % 5HP and TPH parsing
            if (min(TH_DateTime) < min(GPS_final))
                TO_TPH = find(TH_DateTime(:)≥min(GPS_final),1,'first');
            else
                TO_TPH = 1;
```

```
            end
        end

        if (max(MHP_DateTime) > max(GPS_final))
            LND_MHP = find(MHP_DateTime(:)≥max(GPS_final),1,'first');
        else
            LND_MHP = length(MHP_DateTime);
        end

        if(exist('THSense','var'))
            if (max(TH_DateTime) > max(GPS_final))
                LND_TPH = ...
                   find(TH_DateTime(:)≥max(GPS_final),1,'first');
            else
                LND_TPH = length(TH_DateTime);
            end

            TPH_entry = THSense(TO_TPH:LND_TPH,:);
            TH_Date = TH_Date(TO_TPH:LND_TPH,:);
            TH_Time = TH_Time(TO_TPH:LND_TPH,:);
            TPH_time_out = (TPH_entry(:,1)-min(TPH_entry(:,1)))/1000;
        end

        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;

        Pix_time_out = (PixData(:,1)-min(PixData(:,1)))/1000;

        % Create tables with the data and variable names
```

```
            MHP_table = ...
                table(MHP_entry(:,1),MHP_entry(:,2),MHP_time_out,...
                MHP_Date, MHP_Time,MHP_entry(:,3),MHP_entry(:,4),...
                MHP_entry(:,5),MHP_entry(:,6),MHP_entry(:,7),MHP_entry(:,8),...
                MHP_entry(:,9),MHP_entry(:,10),MHP_entry(:,11),...
                MHP_entry(:,12),MHP_entry(:,13),MHP_entry(:,14),...
                'VariableNames', {'Board Time from PowerUp (msec)',...
                'Pix Time from PowerUp (msec)','Pix time from parse',...
                'UTC Date','UTC Time','U (m/s)','V (m/s)','W (m/s)',...
                'Alpha(deg)','Beta(deg)','Alpha Mean Aver.',...
                'Beta Mean Aver.','Total Velocity (m/s)','TBD',...
                'TBD2','TBD3','TBD4'} );
            save(fullParsedMatFileName,'MHP_table','-append');

            PixData_table = ...
                table(PixData(:,1),PixData(:,2),Pix_time_out,...
                PixData(:,3),'VariableNames',{'Sensor board time (ms)',...
                'GPS Unix Time (sec)','Pix board time (sec)',...
                'Pix board time (ms)'});
            save(fullParsedMatFileName,'PixData_table','-append');

            if(exist('THSense','var'))
                TPH_table = table(TPH_entry(:,1),TPH_entry(:,2),...
                    TPH_time_out,TH_Date, TH_Time, TPH_entry(:,3),...
                    TPH_entry(:,4),TPH_entry(:,5),TPH_entry(:,6),...
                    TPH_entry(:,7),TPH_entry(:,8) , 'VariableNames'...
                    , {'Board Time from PowerUp (msec)',...
                    'Pixhawk Time from PowerUp (msec)',...
                    'Pix Time from parse','UTC Date','UTC Time',...
                    'Temp 1 ( C )','Temp 2 ( C )','Temp 3 ( C )',...
                    'Humidity 1 (%)','Humidity 2 (%)','Humidity 3 (%)'} );
                save(fullParsedMatFileName,'TPH_table','-append');
```

```
            end

        end
        if(exist('TPH_table','var'))
            TPH = [table2array(TPH_table(:,1:3)) ...
                table2array(TPH_table(:,6:end))];
        end
        if(exist('MHP_table','var'))
            MHP = [table2array(MHP_table(:,1:3)) ...
                table2array(MHP_table(:,6:end))];
            CTUN = table2array(CTUN_table);
            NKF1 = table2array(NKF1_table);
        end

                    %% ADD CODE HERE

%                      count = 0;
%                      for i=2:(length(NKF2(:,1))-1)
%                          if(NKF2(i,4)≠NKF2(i-1,4) | NKF2(i,5)≠NKF2(i-1,5))
%                              count = count+1;
%                              trueNKF2(count,:) = NKF2(i,:);
%                          end
%                      end
%
%% Kolmogorov -5/3 law comparison

%fs_avgmhp = round(length(MHP(:,3))/MHP(end,3));

%[freq, psdx] = Kolmogorov(MHP(:,11),fs_avgmhp);

                         %% Removal of aircraft velocities from 5hp data ...
                             (no removal of rotational velocities)
```

```
        % Variables to locate/Initialize

        bDATA = [NKF1(:,4),NKF1(:,5),NKF1(:,6)];

        % U_ac aircraft velocity [U_ac, V_ac, W_ac]
        % GPS/Filter data
        gDATA = [NKF1(:,7), NKF1(:,8), NKF1(:,9)]; % this is VN, ... VE, VD (will need to be rotated into body frame)

        % data fusion for 5hp data and pixhawk velocities
        % Pixhawk Kalman Filter Runs at ¬25hz (not formated to run yet)

         for i = 1:length(NKF1(:,3))
            NKF1_5HP(i,1) = find(NKF1(i,3)≥MHP(:,3),1,'last');
            NKF1_IMU(i,1) = find(NKF1(i,3)≥IMU(:,3),1,'last');
         end

         ProbeSpeed = MHP(:,11);

         %Windowing of data for fusion
         for i = 1:length(NKF1(:,3))-1
            avgMHP(i,1) = ...
                mean(ProbeSpeed(NKF1_5HP(i,1):NKF1_5HP(i+1,1))); % ... average multi hole probe speed
            avgMHP(i,2) = ...
                mean(Alpha(NKF1_5HP(i,1):NKF1_5HP(i+1,1))); % ... Average probe alpha
            avgMHP(i,3) = ...
                mean(Beta(NKF1_5HP(i,1):NKF1_5HP(i+1,1))); % average ... probe Beta
            avgMHPBoardTime(i,1) = MHP(NKF1_5HP(i,1),3);
            avgUnixT(i,1) = MHPData_Unix(NKF1_5HP(i,1),1);
```

```
        avgIMU(i,1) = IMU(NKF1_IMU(i,1),3);
        avgIMU(i,2) = ...
            mean(IMU((NKF1_IMU(i,1):NKF1_IMU(i+1,1)),4)); % ...
            Average IMU gyro x
        avgIMU(i,3) = ...
            mean(IMU((NKF1_IMU(i,1):NKF1_IMU(i+1,1)),5)); % ...
            average IMU gyro y
        avgIMU(i,4) = ...
            mean(IMU((NKF1_IMU(i,1):NKF1_IMU(i+1,1)),6)); % ...
            average IMU gyro z
    end

    avgMHP(length(NKF1(:,3)),1) = ProbeSpeed(length(NKF1(:,3)),1);
    avgMHPBoardTime(length(NKF1(:,3)),1) = avgMHPBoardTime(i,1);
    avgIMU(length(NKF1(:,3)),1) = avgIMU(i,1);
    avgUnixT(length(NKF1(:,3)),1) = MHPData_Unix(end,1);


    % Rate Data
    rDATA = [avgIMU(:,2),avgIMU(:,3), avgIMU(:,4)];

    %Function Calls

    %Hard Code Length of probe from center

    L = .40; % Nimbus
    %L = .23;  % NT

    %Basic function just removes first order aircraft states
    [mhpVEL, mhpANG, mhpSPEED]=MHPA_Rautenberg2018(avgMHP, ...
        gDATA, bDATA);
    %Adds removal of second order acceleration terms
```

```
                [mhpVELr, mhpANGr, mhpSPEEDr]=MHPA_wRATES(avgMHP, gDATA, ...
                    bDATA, rDATA,L);

                for i = 1:length(GPS(:,8))
                    GPS_5HP(i,1) = ...
                        find(GPS(i,3)≥avgMHPBoardTime(:,1),1,'last');
                end

                %windowing for alt vs windspeed plots
                for i = 1:length(GPS(:,8))-1
                     avgWS4alt(i,1) = ...
                         mean(mhpSPEED(GPS_5HP(i,1):GPS_5HP(i+1,1)));
                end
%
                avgWS4alt(length(GPS(:,8)),1) = mhpSPEED(length(GPS(:,8)),1);

                %% All the spectral Analysis

                fs_avg = round(length(NKF1_5HP)/MHP(end,3)); % post window ...
                    sampling frequency
                L_avg = length(NKF1_5HP); % sample length

                [P1_avg, f_avg] = freq_this(mhpSPEED, fs_avg, L_avg);

                figure(12)
                plot(f_avg,P1_avg)
                title('Single-Sided Amplitude Spectrum of X(t)')
                xlabel('f (Hz)')
                ylabel('|P1(f)|')
                xlim([0 .5])
                if(strcmpi(mhpAvg,'Yes'))
```

```
                [loop_freq, vert, button] = ginput(1); %grab the peak ...
                    of the fft graph (should be the frequency of the orbit)

                orbitTIME = 1/loop_freq;

                n_samples = round(orbitTIME*fs_avg); % number of ...
                    samples to average over

                 % averaging of orbits

                 for i = 1:length(NKF1(:,3))
                     if i < n_samples + 1
                         windSPEEDavg(i,1) = mean(mhpSPEED(1:i+1));
                         windANGLEavg(i,1) = mean(mhpANG(1:i+1));
                         mhpVNavg(i,1) = mean(mhpVEL((1:i+1),1));
                         mhpVEavg(i,1) = mean(mhpVEL((1:i+1),2));
                         mhpVDavg(i,1) = mean(mhpVEL((1:i+1),3));
                     else
                         windSPEEDavg(i,1) = mean(mhpSPEED(i-n_samples:i));
                         windANGLEavg(i,1) = mean(mhpANG(i-n_samples:i));
                         mhpVNavg(i,1) = mean(mhpVEL((i-n_samples:i),1));
                         mhpVEavg(i,1) = mean(mhpVEL((i-n_samples:i),2));
                         mhpVDavg(i,1) = mean(mhpVEL((i-n_samples:i),3));
                     end
                 end

                 %windowing for alt vs windspeed plots
             for i = 1:length(GPS(:,8))-1
                 avgWS4alt2(i,1) = ...
                     mean(windSPEEDavg(GPS_5HP(i,1):GPS_5HP(i+1,1)));
             end
%
```

```
            avgWS4alt2(length(GPS(:,8)),1) = ...
                windSPEEDavg(length(GPS(:,8)),1);

%             figure(69)
%             plot(avgWS4alt2(:,1),GPS_mat(:,7))
%             xlabel('Wind Speed (m/s)')
%             ylabel('Altitude AGL (m)')

            end



            %% 5HP & TPH plotting

        if(strcmpi(mhpValue,'Yes') && strcmpi(tphValue,'Yes') && ...
            exist('THSense','var'))

            fig4=figure(4);
            fig4.Name = 'Parsed 5HP and TPH Data';
            set(fig4,'defaultLegendAutoUpdate','off');
            subplot(2,1,1);
            plt2 = ...
                plot(TPH(:,2)/1000,TPH(:,4),'r-',TPH(:,2)/1000,TPH(:,5),'b-',TPH(:,2)/10
            title('Temp and Humidity vs Time')
            legend({'Temp 1','Temp 2', 'Temp 3','Humid 1','Humid 2', ...
                'Humid 3'},'Location','southeast')
            xlabel('Time (sec)');
            ylabel('Temp ( C ) and Humidity (%)');
            xlim([min(TPH(:,2)/1000) max(TPH(:,2)/1000)])

            subplot(2,1,2);
```

```
plt3 = ...
    plot(MHP(:,3),MHP(:,4),'r',MHP(:,3),MHP(:,5),'b',MHP(:,3),MHP(:,6),'g');
title('u, v, w (raw)')
legend({'u','v','w'},'Location','northwest')
ylabel('Velocity (m/s)');
xlim([(min(MHP(:,3))) (max(MHP(:,3)))])

fig18=figure(18);
fig18.Name='MHP Wind Frame';
plt3 = ...
    plot(MHP(:,3),MHP(:,11),'-r',MHP(:,3),MHP(:,7),'-b',MHP(:,3),MHP(:,8),'-
ylabel('Airspeed (m/s), Alpha and Beta(deg)');
ylim([-10,20])
title('Airspeed, Alpha, and Beta (raw)')
legend({'Airspeed','Alpha','Beta'},'Location','northwest')
xlim([(min(MHP(:,3))) (max(MHP(:,3)))])

fig5=figure(5);
fig5.Name = 'MHP vs Pix Airspeeds';
set(fig5,'defaultLegendAutoUpdate','off');
subplot(2,1,1);
plt1 = plot(MHP(:,3),MHP(:,4),'r',CTUN(:,3),CTUN(:,4),'k');
title('MHP Pitot and Pix Airspeeds with Time')
legend({'MHP Pitot', 'Pix Arspd'},'Location','northwest')
ylabel('Airspeed (m/s)');
xlim([(min(MHP(:,3))) (max(MHP(:,3)))])

subplot(2,1,2)
plt2 = ...
    plot(MHP(:,3),MHP(:,4),'r',MHP(:,3),MHP(:,5),'b',MHP(:,3),MHP(:,6),'g',C
title('MHP Pitot, Alpha, Beta, and Pix Airspeeds with Time')
```

```
        legend({'MHP-Pitot','MHP-Alpha','MHP-Beta', 'Pix ...
            Arspd'},'Location','northwest')
        ylabel('Airspeed (m/s)');
        xlim([(min(MHP(:,3))) (max(MHP(:,3)))])

        fig11 = figure(11);
        fig11.Name = 'MHP vs. Pix Windspeeds';
        plt = plot(NKF1(:,2),mhpWIND,NKF1(:,2),wind);
        legend({'MHP Wind Estimation', 'PixWind Estimation'}, ...
            'Location', 'Best')
        title('MHP vs. Pix Windspeeds (Body Removed)')
        xlabel('time (ms)')
        ylabel('wind speed (m/s)')

        fig16 = figure(16);
        fig16.Name = 'MHP vs. Pix Wind Vector';
        plt = plot(NKF2(:,2), VWN, NKF2(:,2), VWE, ...
            NKF1(:,2),mhpVEL(:,1), NKF1(:,2), mhpVEL(:,2));
        legend({'VWN Pix', 'VWE Pix', 'VWN MHP', 'VWE MHP'}, ...
            'Location', 'Best')
        title('MHP vs. Pix Vector')
        xlabel('time(ms)')
        ylabel('wind speed (m/s)')

        fig20 = figure(20);
        fig20.Name = 'MHP vs. Pix Wind Vector';
        plt = plot(NKF1(:,2),mhpVEL(:,1), NKF1(:,2), ...
            mhpVEL(:,2),NKF1(:,2),mhpVEL(:,3));
        legend({'North Wind', 'East Wind', 'Down Wind'}, ...
            'Location', 'Best')
        title('MHP Ground Based Wind Speed')
        xlabel('time(ms)')
```

```
        ylabel('wind speed (m/s)')

        if(strcmpi(mhpAvg,'Yes'))
            fig13 = figure(13);
            fig13.Name = 'MHP vs. Pix Windspeeds';
            plt = plot(NKF1(:,2),windSPEEDavg,NKF2(:,2),wind);
            legend({'MHP Wind Estimation', 'PixWind Estimation'}, ...
                'Location', 'Best')
            title('MHP vs. Pix Windspeeds (Orbit Averaged)')
            xlabel('time (ms)')
            ylabel('wind speed (m/s)')

            fig14 = figure(14);
            fig14.Name = 'MHP vs. Pix Angles';
            plt = plot(NKF1(:,2),windANGLEavg,NKF2(:,2),windANG);
            legend('MHP Wind Angle Estimation', 'PixWind Estimation')
            title({'MHP vs. Pix Angle (Orbit Averaged)'}, ...
                'Location', 'Best')
            xlabel('time (ms)')
            ylabel('wind speed (deg)')

            fig17 = figure(17);
            fig17.Name = 'MHP vs. Pix Wind Vector';
            plt = plot(NKF2(:,2), VWN, NKF2(:,2), VWE, ...
                NKF1(:,2),mhpVNavg(:,1), NKF1(:,2), mhpVEavg(:,2));
            legend({'VWN Pix', 'VWE Pix', 'VWN MHP', 'VWE MHP'}, ...
                'Location', 'Best')
            title('MHP vs. Pix Vector')
            xlabel('time(ms)')
            ylabel('wind speed (m/s)')
```

```
            fig21 = figure(21);
            fig21.Name = 'MHP Wind Vector';
            plt = plot(NKF1(:,2),mhpVNavg(:,1), NKF1(:,2), ...
                mhpVEavg(:,2),NKF1(:,2),mhpVDavg(:,3));
            legend({'North Wind', 'East Wind', 'Down Wind'}, ...
                'Location', 'Best')
            title('MHP Ground Based Wind Speed Averaged')
            xlabel('time(ms)')
            ylabel('wind speed (m/s)')
            end

        if(strcmpi(sensorOut,'Yes'))
            if(strcmpi(DFL_NewOld,'New'))
                % Output parsed TPH data
                baseFileName = sprintf('%s_Parsed_TPH.csv', ...
                    baseNameNoExtDFL);
                fullOutputMatFileName = fullfile(folderDFL, ...
                    baseFileName);
                % Write data to text file
                writetable(TPH_table, fullOutputMatFileName);

                % Get the name of the input.mat file and save as ...
                    input_Parsed_MHP.csv
                baseFileName = sprintf('%s_Parsed_MHP.csv', ...
                    baseNameNoExtDFL);
                fullOutputMatFileName = fullfile(folderDFL, ...
                    baseFileName);
                % Write data to text file
                writetable(MHP_table, fullOutputMatFileName);
            else
                % Output parsed TPH data
```

```
                baseFileName = sprintf('%s_TPH_NS.csv', ...
                    baseNameNoExtDFL);
                fullOutputMatFileName = fullfile(folderDFL, ...
                    baseFileName);
                % Write data to text file
                writetable(TPH_table, fullOutputMatFileName);

                % Get the name of the input.mat file and save as ...
                    input_Parsed_MHP.csv
                baseFileName = sprintf('%s_MHP_NS.csv', ...
                    baseNameNoExtDFL);
                fullOutputMatFileName = fullfile(folderDFL, ...
                    baseFileName);
                % Write data to text file
                writetable(MHP_table, fullOutputMatFileName);
            end
        end

    elseif(strcmpi(mhpValue,'Yes') && (strcmpi(tphValue,'No') | ¬...
        exist('THSense','var')))

        fig4=figure(4);
        fig4.Name = 'Parsed 5HP Data';
        set(fig4,'defaultLegendAutoUpdate','off');
        plt3 = ...
            plot(MHP(:,3),MHP(:,4),'r',MHP(:,3),MHP(:,5),'b',MHP(:,3),MHP(:,6),'g');
        title('u, v, w(raw)')
        legend({'u','v','w'},'Location','northwest')
        xlabel('Time (sec)');
        ylabel('Velocity (m/s)');
        xlim([(min(MHP(:,3))) (max(MHP(:,3)))])

```

```
fig5=figure(5);
fig5.Name = 'MHP vs Pix Airspeeds';
set(fig5,'defaultLegendAutoUpdate','off');
subplot(2,1,1);
plt1 = plot(MHP(:,3),MHP(:,4),'r',CTUN(:,3),CTUN(:,4),'k');
title('MHP Pitot and Pix Airspeeds with Time')
legend({'MHP Pitot', 'Pix Arspd'},'Location','northwest')
ylabel('Airspeed (m/s)');
xlim([(min(MHP(:,3))) (max(MHP(:,3)))])

subplot(2,1,2)
plt2 = ...
    plot(MHP(:,3),MHP(:,4),'r',MHP(:,3),MHP(:,5),'b',MHP(:,3),MHP(:,6),'g',C
title('MHP Pitot, Alpha, Beta, and Pix Airspeeds with Time')
legend({'MHP-u','MHP-v','MHP-w', 'Pix ...
    Arspd'},'Location','northwest')
ylabel('Airspeed (m/s)');
xlim([(min(MHP(:,3))) (max(MHP(:,3)))])

fig11 = figure(11);
fig11.Name = 'MHP vs. Pix Windspeeds';
plt = plot(NKF1(:,2),mhpSPEED,NKF2(:,2),wind);
legend({'MHP Wind Estimation', 'PixWind Estimation'}, ...
    'Location', 'Best')
title('MHP vs. Pix Windspeeds')
xlabel('time (ms)')
ylabel('wind speed (m/s)')

fig14 = figure(14);
fig14.Name = 'MHP vs. Pix Wind Angles';
plt = plot(NKF1(:,2),mhpANG,NKF2(:,2),windANG);
```

```
        legend({'MHP Wind Angle Estimation', 'PixWind Estimation'}, ...
            'Location', 'Best')
        title('MHP vs. Pix Wind Angles')
        xlabel('time (ms)')
        ylabel('wind angle (deg)')

        fig16 = figure(16);
        fig16.Name = 'MHP vs. Pix Wind Vector';
        plt = plot(NKF2(:,2), VWN, NKF2(:,2), VWE, ...
            NKF1(:,2),mhpVEL(:,1), NKF1(:,2), mhpVEL(:,2));
        legend({'VWN Pix', 'VWE Pix', 'VWN MHP', 'VWE MHP'}, ...
            'Location', 'Best')
        title('MHP vs. Pix Vector')
        xlabel('time(ms)')
        ylabel('wind speed (m/s)')

        fig20 = figure(20);
        fig20.Name = 'MHP vs. Pix Wind Vector';
        plt = plot(NKF1(:,2),mhpVEL(:,1), NKF1(:,2), ...
            mhpVEL(:,2),NKF1(:,2),mhpVEL(:,3));
        legend({'North Wind', 'East Wind', 'Down Wind'}, ...
            'Location', 'Best')
        title('MHP Ground Based Wind Speed')
        xlabel('time(ms)')
        ylabel('wind speed (m/s)')

        if(strcmpi(mhpAvg,'Yes'))
            fig12 = figure(12);
            fig12.Name = 'MHP vs. Pix Windspeeds';
            plt = plot(NKF1(:,2),windSPEEDavg,NKF2(:,2),wind);
            legend({'MHP Wind Estimation', 'PixWind Estimation'}, ...
                'Location', 'Best')
```

```
            title('MHP vs. Pix Windspeeds (Orbit Averaged)')
            xlabel('time (ms)')
            ylabel('wind speed (m/s)')

            fig15 = figure(15);
            fig15.Name = 'MHP vs. Pix Windspeeds';
            plt = plot(NKF1(:,2),windANGLEavg,NKF2(:,2),windANG);
            legend({'MHP Wind Estimation', 'PixWind Estimation'}, ...
                'Location', 'Best')
            title('MHP vs. Pix Wind Angles (Orbit Averaged)')
            xlabel('time (ms)')
            ylabel('wind angle (deg)')

            fig17 = figure(17);
            fig17.Name = 'MHP vs. Pix Wind Vector';
            plt = plot(NKF2(:,2), VWN, NKF2(:,2), VWE, ...
                NKF1(:,2),mhpVNavg(:,1), NKF1(:,2), mhpVEavg(:,1));
            legend({'VWN Pix', 'VWE Pix', 'VWN MHP', 'VWE MHP'}, ...
                'Location', 'Best')
            title('MHP vs. Pix Vector')
            xlabel('time(ms)')
            ylabel('wind speed (m/s)')

            fig21 = figure(21);
            fig21.Name = 'MHP Wind Vector';
            plt = plot(NKF1(:,2),mhpVNavg(:,1), NKF1(:,2), ...
                mhpVEavg(:,1),NKF1(:,2),mhpVDavg(:,1));
            legend({'North Wind', 'East Wind', 'Down Wind'}, ...
                'Location', 'Best')
            title('MHP Ground Based Wind Speed Averaged')
            xlabel('time(ms)')
            ylabel('wind speed (m/s)')
```

```
        end

        if(strcmpi(sensorOut,'Yes'))
            if(strcmpi(DFL_NewOld,'New'))
                % Get the name of the input.mat file and save as ... input_Parsed_MHP.csv
                baseFileName = sprintf('%s_Parsed_MHP.csv', ...
                    baseNameNoExtDFL);
                fullOutputMatFileName = fullfile(folderDFL, ...
                    baseFileName);
                % Write data to text file
                writetable(MHP_table, fullOutputMatFileName);
            else
                % Get the name of the input.mat file and save as ... input_Parsed_MHP.csv
                baseFileName = sprintf('%s_MHP_NS.csv', ...
                    baseNameNoExtDFL);
                fullOutputMatFileName = fullfile(folderDFL, ...
                    baseFileName);
                % Write data to text file
                writetable(MHP_table, fullOutputMatFileName);
            end
        end


    elseif(strcmpi(mhpValue,'No') && strcmpi(tphValue,'Yes') && ...
        exist('THSense','var'))

        fig4=figure(4);
        fig4.Name = 'Parsed TPH Data';
```

```
            plt2 = ...
                plot(TPH(:,2)/1000,TPH(:,3),'r-',TPH(:,2)/1000,TPH(:,4),'b-',TPH(:,2)/10
            title('Temp and Humidity vs Time')
            legend({'Temp 1','Temp 2', 'Temp 3','Humid 1','Humid 2', ...
                'Humid 3'},'Location','southeast')
            xlabel('Time (sec)');
            ylabel('Temp ( C ) and Humidity (%)');
            xlim([(min(TPH(:,2)/1000)) (max(TPH(:,2)/1000))])

            if(strcmpi(sensorOut,'Yes'))
                if(strcmpi(DFL_NewOld,'New'))
                    % Output parsed TPH data
                    baseFileName = sprintf('%s_Parsed_TPH.csv', ...
                        baseNameNoExtDFL);
                    fullOutputMatFileName = fullfile(folderDFL, ...
                        baseFileName);
                    % Write data to text file
                    writetable(TPH_table, fullOutputMatFileName);
                else
                    % Output parsed TPH data
                    baseFileName = sprintf('%s_TPH_NS.csv', ...
                        baseNameNoExtDFL);
                    fullOutputMatFileName = fullfile(folderDFL, ...
                        baseFileName);
                    % Write data to text file
                    writetable(TPH_table, fullOutputMatFileName);
                end
            end
        end
    end
    %% CSV Output of GPS Data
    if (strcmpi(gpsOut,'Yes'))
```

```
        if(strcmpi(DFL_NewOld,'New'))
            % Get the name of the input.mat file and save as input_GPS.csv
            baseFileName = sprintf('%s_GPS.csv', baseNameNoExtDFL);
        elseif(strcmpi(DFL_NewOld,'Old'))
            % Get the name of the input.mat file and save as ...
                input_GPS_NS.csv
            baseFileName = sprintf('%s_GPS_NS.csv', baseNameNoExtDFL);
        end
        fullOutputMatFileName = fullfile(folderDFL, baseFileName);
        % Write data to text file
        writetable(GPS_table, fullOutputMatFileName);
    end
    %% CSV Output of Aircraft Attitude Data
    if (strcmpi(attitudeOut,'Yes'))
        if(strcmpi(DFL_NewOld,'New'))
            % Get the name of the input.mat file and save as ...
                input_Attitude.csv
            baseFileName = sprintf('%s_Attitude.csv', baseNameNoExtDFL);
        elseif(strcmpi(DFL_NewOld,'Old'))
            % Get the name of the input.mat file and save as ...
                input_Attitude_NS.csv
            baseFileName = sprintf('%s_Attitude_NS.csv', baseNameNoExtDFL);
        end
        fullOutputMatFileName = fullfile(folderDFL, baseFileName);
        % Write data to text file
        writetable(ATT_table, fullOutputMatFileName);
    end
    %% CSV Output of IMU Data
    if (strcmpi(stateSpace,'Yes'))
        if(strcmpi(DFL_NewOld,'New'))
            % Get the name of the input.mat file and save as input_IMU.csv
            baseFileName = sprintf('%s_IMU.csv', baseNameNoExtDFL);
```

```
    elseif(strcmpi(DFL_NewOld,'Old'))
        % Get the name of the input.mat file and save as ...
            input_IMU_NS.csv
        baseFileName = sprintf('%s_IMU_NS.csv', baseNameNoExtDFL);
    end
    fullOutputMatFileName = fullfile(folderDFL, baseFileName);
    % Write data to text file
    writetable(IMU_table, fullOutputMatFileName);
end
%% Plot Data According To User Input
if (strcmpi(graphToggle,'Yes'))
    % Converted timestamps to time (in seconds) from TO to LND
    t_GPS = table2array(GPS_table(:,3));
    t_NKF = table2array(NKF2_table(:,3));
    t_RCOU = table2array(RCOU_table(:,3));
    t_BARO = table2array(BARO_table(:,3));
    t_ATT = table2array(ATT_table(:,3));

    if(strcmpi(arduPilotType,'ArduCopter'))
    elseif(strcmpi(arduPilotType,'CopterSonde'))
        t_CSIMET = table2array(CSIMET_table(:,3));
        CSIMET_mat = table2array(CSIMET_table);
    else
        t_ctun = table2array(CTUN_table(:,3));
        CTUN_mat = table2array(CTUN_table);
    end

    t_low = min(GPS_table{:,3});
    t_high = max(GPS_table{:,3});
    GPS_mat = [table2array(GPS_table(:,1:3)) ...
       table2array(GPS_table(:,6:end))];
    ATT_mat = table2array(ATT_table);
```

```
BARO_mat = table2array(BARO_table);
IMU_mat = table2array(IMU_table);
if(exist('MHP_table','var') && ¬exist('MHP','var'))
    MHP = [table2array(MHP_table(:,1:3)) ...
        table2array(MHP_table(:,6:end))];
end
if(exist('TPH_table','var') && ¬exist('TPH','var'))
    TPH = [table2array(TPH_table(:,1:3)) ...
        table2array(TPH_table(:,6:end))];
end
if(exist('PixData_table','var') && ¬exist('PixData','var'))
    PixData = table2array(PixData_table);
end
if(exist('iMet_table','var') && ¬exist('iMet','var'))
    iMet = [table2array(iMet_table(:,1)) ...
        table2array(iMet_table(:,4:end))];
end
NKF2_mat = table2array(NKF2_table);
RCOU_mat = table2array(RCOU_table);



if (strcmpi(overlay,'Yes'))

    fig6=figure(6);
    fig6.Name = 'Data Plots from Parsed Sensor data and ...
        Autopilot DFL';

    if(strcmpi(mhpValue,'Yes'))
         if(exist('MHP','var'))
             % Groundspeed plot
             plt1 = subplot(5,1,1);
```

```
                plot(t_GPS,GPS_mat(:,8),'k',t_ctun,CTUN_mat(:,4),'b',t_NKF,NKF2_mat
                title('Groundspeed (black), Airspeed (blue), ...
                    Windspeed (red), MHP-Pitot (green) vs Time')
                ylabel({'Velocity (m/s)'})
            end
        elseif(strcmpi(arduPilotType,'ArduCopter') | ...
                strcmpi(arduPilotType,'CopterSonde'))
            % Groundspeed plot
            plt1 = subplot(5,1,1);
            plot(t_GPS,GPS_mat(:,8),'k',t_NKF,NKF2_mat(:,6),'r')
            title('Groundspeed (black), Windspeed (red) vs Time')
            ylabel({'Velocity (m/s)'})
        else
            % Groundspeed plot
            plt1 = subplot(5,1,1);
            plot(t_GPS,GPS_mat(:,8),'k',t_ctun,CTUN_mat(:,4),'b',t_NKF,NKF2_mat(:,6)
            title('Groundspeed (black), Airspeed (blue), Windspeed ...
                (red) vs Time')
            ylabel({'Velocity (m/s)'})
        end

        if(strcmpi(iMetValue,'Yes'))
            if(exist('iMet_table','var'))
                % Barometric Pressure of Pixhawk and Sensor Packages
                plt4 = subplot(5,1,4);
                plot(t_BARO, ...
                    BARO_mat(:,4),'b-',iMet(:,1),iMet(:,2),'r-');
                title('Pixhawk (internal) vs iMet (external) ...
                    Atmospheric Pressure Measurements')
                %xticks([150:20:370])
                ylabel({'Pixhawk Pressure (blue)';'iMet Pressure ...
                    (red)';'(mbar)'});
```

```
    end
else
    % Barometric Pressure of Pixhawk and Sensor Packages
    plt4 = subplot(5,1,4);
    plot(t_BARO,BARO_mat(:,4),'b-');
    title('Pixhawk (internal) Atmospheric Pressure ...
        Measurements')
    %xticks([150:20:370])
    ylabel({'Pixhawk Pressure (blue)';'(mbar)'});
end

if(strcmpi(arduPilotType,'CopterSonde'))
    plt11 = subplot(2,1,1);
    colors = {'k','r','b','g','k-','r-','b-','g-'};
    for i=0:3
        if(min(CSIMET_mat(:,i+4))>0)
            hold on
            plot(t_CSIMET,CSIMET_mat(:,i+4)/100,colors{i+1});
            plot(t_CSIMET,CSIMET_mat(:,i+8)-273.15,colors{i+5});
        end
    end
    hold off
    title('CopterSonde Temperature and Humidity');
    ylabel({'Temp ( C ) and Humidity (%)'});

    plt12 = subplot(2,1,2);
    plot(BARO_mat(:,3),BARO_mat(:,4),'k');
    title('Pixhawk (internal) Pressure Readings');
    ylabel({'Pressure (mbar)'});
    xlabel({'Time (sec)'});
    linkaxes([plt11 plt12],'x')
    xlim([min(t_CSIMET) max(t_CSIMET)])
```

```
        else
            % Throttle Output
            plt2 = subplot(5,1,2 );
            plot(t_RCOU,RCOU_mat(:,6),'b')
            title('Throttle vs Time')
            ylabel({'Throttle';'(%)'})
            %xticks([150:20:370])
            ylim([0 100])

            % For the dotted line along x-axis of pitch plot
            zero=int8(zeros(length(ATT_mat(:,5)),1));

            % Aircraft Pitch angle: Can change ylim to something ...
                more relevant.
            % TIV uses -20 to 50 to see high AoA landing
            plt3 = subplot(5,1,3);
            plot(t_ATT,ATT_mat(:,5),'b',t_ATT,zero,'r:')
            title('Aircraft Pitch Angle vs Time')
            ylabel({'Aircraft Pitch';'Angle (  )'})
            %xticks([150:20:370])
            ylim([-20 50])

            % Altitude plot
            plt5 = subplot(5,1,5);
            plot(t_GPS,GPS_mat(:,7),'b')
            title('Altitude vs Time')
            %xticks([150:20:370])
            ylabel({'Altitude AGL';'(m)'})
            xlabel('Time (seconds)')

            linkaxes([plt1 plt2 plt3 plt4 plt5],'x')
            xlim([t_low t_high])
```

```
            end

            if(strcmpi(tphValue,'Yes') && strcmpi(mhpValue,'Yes') && ...
                strcmpi(iMetValue,'Yes'))
                 if(exist('TPH','var') && exist('MHP','var') && ...
                     exist('iMet','var'))
                      fig7=figure(7);
                      fig7.Name = 'Pixhawk, iMet, TPH, and MHP Data ...
                          Comparisons';

                      if(strcmpi(arduPilotType,'ArduCopter') | ...
                          strcmpi(arduPilotType,'CopterSonde'))
                           plt1 = subplot(3,1,1);
                           plot(MHP(:,3),MHP(:,4),'r',MHP(:,3),MHP(:,5),'b',MHP(:,3),MHP(:,
                           title('5HP Alpha, Beta, and Pitot Velocities');
                           ylabel({'Velocity (m/s)'});
                      else
                           plt1 = subplot(3,1,1);
                           plot(t_ctun,CTUN_mat(:,4),'k',MHP(:,3),MHP(:,4),'r',MHP(:,3),MHP
                           title('Pixhawk Airspeed vs 5HP Alpha, Beta, and ...
                               Pitot Velocities');
                           ylabel({'Velocity (m/s)'});
                      end

                      plt2 = subplot(3,1,2);
                      plot(iMet(:,1),iMet(:,3),'k',iMet(:,1),iMet(:,4),'k-',TPH(:,3),TPH(:
                      title('iMet vs Sensor Package Temperature and ...
                          Humidity');
                      ylabel({'Temp ( C ) and Humidity (%)'});

                      plt3 = subplot(3,1,3);
                      plot(BARO_mat(:,3),BARO_mat(:,4),'k',iMet(:,1),iMet(:,2),'r');
```

```
                    title('Pixhawk (internal) vs iMet (external) ...
                        Pressure Readings');
                    ylabel({'Pressure (mbar)'});
                    xlabel({'Time (sec)'});
                    linkaxes([plt1 plt2 plt3],'x')
                    xlim([min(t_BARO) max(t_BARO)])
                end
            end

        else

            fig6=figure(6);
            fig6.Name = 'Data Plots from Parsed Autopilot DFL';

            if(strcmpi(arduPilotType,'ArduCopter') | ...
                strcmpi(arduPilotType,'CopterSonde'))
                % Groundspeed plot
                plt1 = subplot(4,1,1);
                plot(t_GPS,GPS_mat(:,8),'b-',t_NKF,NKF2_mat(:,6),'g-')
                title('Groundspeed, and Windspeed vs Time')
                ylabel({'Groundspeed (blue)';'Windspeed (green)';'(m/s)'})
            else
                % Groundspeed plot
                plt1 = subplot(4,1,1);
                plot(t_GPS,GPS_mat(:,8),'b-',t_ctun,CTUN_mat(:,4),'r-',t_NKF,NKF2_mat(:,
                title('Groundspeed, Airspeed, and Windspeed vs Time')
                ylabel({'Groundspeed (blue)';'Airspeed ...
                    (red)';'Windspeed (green)';'(m/s)'})
            end

            % Throttle Output
            plt2 = subplot(4,1,2);
```

```
        plot(t_RCOU,RCOU_mat(:,6),'b')
        title('Throttle vs Time')
        ylabel({'Throttle';'(%)'})
        ylim([0 100])

        % For the dotted line along x-axis of pitch plot
        zero=int8(zeros(length(ATT_mat(:,5)),1));

        % Aircraft Pitch angle: Can change ylim to something more ...
            relevant.
        % TIV uses -20 to 50 to see high AoA landing
        plt3 = subplot(4,1,3);
        plot(t_ATT,ATT_mat(:,5),'b',t_ATT,zero,'r:')
        title('Aircraft Pitch Angle vs Time')
        ylabel({'Aircraft Pitch';'Angle (  )'})
        ylim([-10 40])


        % Altitude plot
        plt4 = subplot(4,1,4);
        plot(t_GPS,GPS_mat(:,7),'b')
        title('Altitude vs Time')
        ylabel({'Altitude AGL';'(m)'})
        xlabel('Time (seconds)')

        linkaxes([plt1, plt2, plt3, plt4],'x')
        xlim([t_low t_high])


    end

    if (strcmpi(indvToggle,'Yes'))
```

```
fig8=figure(8);
fig8.Name = 'Interactive Plot - Right click or press ...
    Return/Enter when finished';

% Groundspeed plot
subplot(4,1,1);
plot(t_GPS,GPS_mat(:,8),'b-',t_ctun,CTUN_mat(:,4),'r-',t_NKF,NKF2_mat(:,6),'
title('Groundspeed and Airspeed vs Time')
ylabel({'Groundspeed (blue)';'Airspeed (red)';'Windspeed ...
    (green)';'(m/s)'})
xlim([min(t_GPS) max(t_GPS)])

% Throttle Output
subplot(4,1,2);
plot(t_RCOU,RCOU_tabl(:,6),'b')
title('Throttle vs Time')
ylabel({'Throttle';'(%)'})
xlim([min(t_RCOU) max(t_RCOU)])
ylim([0 100])

% For the dotted line along x-axis of pitch plot
zero=int8(zeros(length(ATT_mat(:,5)),1));

% Aircraft Pitch angle: Can change ylim to something more ...
    relevant.
% TIV uses -20 to 50 to see high AoA landing
subplot(4,1,3);
plot(t_ATT,ATT_mat(:,5),'b',t_ATT,zero,'r:')
title('Aircraft Pitch Angle vs Time')
ylabel({'Aircraft Pitch';'Angle (  )'})
xlim([min(t_ctun) max(t_ctun)])
```

```
        ylim([-20 50])

        % Altitude plot
        subplot(4,1,4);
        plot(t_GPS,GPS_mat(:,7),'b')
        title('Altitude vs Time')
        xlim([min(t_GPS) max(t_GPS)])
        ylabel({'Altitude AGL';'(m)'})
        xlabel('Time (seconds)')
        n = 0;
        while true
            [horiz, vert, button] = ginput(1);
            if isempty(horiz) || button(1) == 3; break; end
            n = n+1;
            x_n(n) = horiz(1); % save all points you continue getting
            hold on
            y_n1(n)=GPS_mat(find(t_GPS≥x_n(n),1),8);      % Groundspeed
            y_n2(n)=CTUN_mat(find(t_ctun≥x_n(n),1),4);      % Airspeed
            y_n3(n)=NKF2_mat(find(t_NKF≥x_n(n),1),6);      % Windspeed
            y_n4(n)=RCOU_mat(find(t_ctun≥x_n(n),1),6);      % ...
                Throttle Percent
            y_n5(n)=ATT_mat(find(t_ctun≥x_n(n),1),5); % Aircraft Pitch
            y_n6(n)=GPS_mat(find(t_GPS≥x_n(n),1),7);     % Altitude
            y_n7(n)=RCOU_mat(find(t_ctun≥x_n(n),1),4);   % Pitch PWM

            % Groundspeed plot
            subplot(4,1,1);
            plot(t_GPS,GPS_mat(:,8),'b',t_ctun,CTUN_mat(:,4),'r',t_NKF,NKF2_mat(:,6)
            title('Groundspeed, Airspeed, and Windspeed vs Time')
            ylabel({'Groundspeed (blue)';'Airspeed ...
                (red)';'Windspeed (green)';'(m/s)'})
            xlim([min(t_GPS) max(t_GPS)])
```

```
            % Throttle Output
            subplot(4,1,5);
            plot(t_RCOU,RCOU_mat(:,6),'b',x_n, y_n4,'kx')
            title('Throttle vs Time')
            ylabel({'Throttle';'(%)'})
            xlim([min(t_RCOU) max(t_RCOU)])
            ylim([0 100])

            % For the dotted line along x-axis of pitch plot
            zero=int8(zeros(length(ATT_mat(:,5)),1));

            % Aircraft Pitch angle: Can change ylim to something ...
                more relevant.
            % TIV uses -20 to 50 to see high AoA landing
            subplot(4,1,3);
            plot(t_ATT,ATT_mat(:,5),'b',t_ATT,zero,'r:',x_n,y_n5,'kx');
            title('Aircraft Pitch Angle vs Time');
            ylabel({'Aircraft Pitch';'Angle (  )'});
            xlim([min(t_ctun) max(t_ctun)]);
            ylim([-20 50]);

            % Altitude plot
            subplot(4,1,4);
            plot(t_GPS,GPS_mat(:,7),'b',x_n,y_n6,'kx')
            title('Altitude vs Time')
            xlim([min(t_GPS) max(t_GPS)])
            ylabel({'Altitude AGL';'(m)'})
            xlabel('Time (seconds)')

            drawnow

```

```
        end

        if (strcmpi(throttleToggle,'Yes')) & ...
            (strcmpi(pitchToggle,'No'))

            thrT = interp1(throttleMap(:,1),throttleMap(:,2),y_n4);

            % Drag = Thrust
            C_D = (thrT.*2)./((y_n2.^2)*S*rho);
            % Lift = Weight
            C_L = (GTOW*2)./(S*rho*y_n2.^2);
            % Ratio of C_L/C_D
            L_D = C_L./C_D;

            % Flight Analysis
            baseFileName = sprintf('%s_metricsFull.csv', ...
                baseNameNoExtDFL);
            fullOutputMatFileName = fullfile(folder, baseFileName);
            % Save file with parsed data as the original filename ...
                plus the added portion
            % Create a table with the data and variable names
            T = table(round(y_n1.',1), round(y_n2.',1), ...
                round(y_n3.',1), round(y_n4.',1), round(y_n5.',1), ...
                round(y_n6.',1), round(y_n7.',1), round(thrT.',2), ...
                round(C_D.',6), round(C_L.',6), round(L_D.',4), ...
                'VariableNames', {'Groundspeed (m/s)','Airspeed ...
                (m/s)','Windspeed (m/s)','Throttle (%)','Aircraft ...
                Pitch (  )','Altitude (m, AGL)','Pitch PWM','Thrust ...
                Output (units of Throttle Curve input ...
                file)','C_D','C_L','CD_CL'} )
            % Write data to text file
            writetable(T, fullOutputMatFileName)
```

```
            fig9 = figure(9);
            fig9.Name = 'Lift vs Drag Coefficients';
            subplot(1,2,1)
            scatter(C_D.',C_L.')
            title('C_L vs C_D')
            ylabel('C_L')
            xlabel('C_D')

            subplot(1,2,2)
            scatter(y_n2.',L_D.')
            title('L_D Ratio vs Airspeed')
            ylabel('L/D')
            xlabel('Airspeed (m/s)')

        elseif (strcmpi(throttleToggle,'Yes')) & ...
            (strcmpi(pitchToggle,'Yes'))

            % Mapping throttle Percent from User Input to Thrust Output
            thrT = interp1(throttleMap(:,1),throttleMap(:,2),y_n4);
            % Mapping pitch PWM from Pixhawk to true angular deflection
            pitchT = interp1(pitchMap(:,1),pitchMap(:,2),y_n7);

            % Drag = Thrust*cos(angle)
            C_D = (thrT.*cos(pitchT/180*3.14)*2)./((y_n2.^2)*S*rho);
            % Lift = Weight + Drag*sin(angle)
            C_L = ...
                ((GTOW+(thrT.*sin(pitchT/180*3.14)))*2)./(S*rho*y_n2.^2);
            % Ratio of C_L/C_D
            L_D = C_L./C_D;

            % Flight Analysis
```

```
baseFileName = sprintf('%s_metricsFull.csv', ...
    baseNameNoExtDFL);
fullOutputMatFileName = fullfile(folder, baseFileName);
% Save file with parsed data as the original filename ...
    plus the added portion
% Create a table with the data and variable names
T = table(round(y_n1.',1), round(y_n2.',1), ...
    round(y_n3.',1), round(y_n4.',1), round(y_n5.',1), ...
    round(y_n6.',1), round(y_n7.',1), round(thrT.',2), ...
    round(pitchT.',1), round(C_D.',6), round(C_L.',6), ...
    round(L_D.',4), 'VariableNames', {'Groundspeed ...
    (m/s)','Airspeed (m/s)','Windspeed (m/s)','Throttle ...
    (%)','Aircraft Pitch (  )','Altitude (m, ...
    AGL)','Pitch PWM','Thrust Output (units of Throttle ...
    Curve input file)','Servo Angular Deflection ...
    (  )','C_D','C_L','CD_CL'} )
% Write data to text file
writetable(T, fullOutputMatFileName)

fig9 = figure(9);
fig9.Name = 'Lift vs Drag Coefficients';
subplot(1,2,1)
scatter(C_D.',C_L.')
title('C_L vs C_D')
ylabel('C_L')
xlabel('C_D')

subplot(1,2,2)
scatter(y_n2.',L_D.')
title('L_D Ratio vs Airspeed')
ylabel('L/D')
xlabel('Airspeed (m/s)')
```

```
            end

        end
    end
    %% Animation of Flight

    if(strcmpi(animateToggle, 'Yes'))
        % Get animation plot titles and output file names for animations
        if(strcmpi(animateToggle,'Yes') | strcmpi(animation,'Yes'))
            plotTitle = input('<strong>Enter Plot Title for animation ...
                function: </strong>','s');
            % Get the name of the intput.mat file and save as ...
                input_parsed.mat
            userFileName = input('<strong>Enter an output file name for ...
                the animation sequence to save as: </strong>','s');
            if(strcmpi(userFileName,''))
                userFileName = 'defaultAnimationOutput';
            end
            vidFileName = regexprep(userFileName, ' +', ' ');
            videoOutputFileName = fullfile(startingFolderDFL, vidFileName);

            animVid = VideoWriter(videoOutputFileName,'MPEG-4');
            animVid.FrameRate = animFrameRate;  %can adjust this, 5 - ...
                10 works well for me
            animVid.Quality = 100;
        end
        if(strcmpi(tripleAnimateTPH,'Yes'))
            %% Animation Setup
            animVid = VideoWriter('Kessler 8-4-2020, ...
                TriplePlot','MPEG-4'); %open video file
```

```
        animVid.FrameRate = 20;  %can adjust this, 5 - 10 works ...
            well for me
        animVid.Quality = 100;

        interval = 20;

        timer = table2array(BARO_table(1:2:8597-3,3));
        cAnim1 = [nan;table2array(TPH_table(1:2:end,6))];
        cAnim2 = [nan;table2array(TPH_table(1:2:end,9))];
        cAnim3 = [nan;table2array(BARO_table(1:2:8597-3,4))];
        xAnim=[nan;table2array(GPS_table(1:4297,6))];
        yAnim=[nan;table2array(GPS_table(1:4297,7))];
        zAnim=[nan;table2array(GPS_table(1:4297,9))];
        lx = length(xAnim);
        ly = length(yAnim);
        lz = length(zAnim);

        fig10 = figure(10);
        fig10.Position = [360 380 1180 400];
        %% Temperature Plot %%%%%%%%%%%%
        subplot(1,3,1);
        ax1 = gca;
        title('Temperature ( C )');
        xlim(ax1, [min(xAnim(2:lx)) max(xAnim(2:lx))]);
        ylim(ax1, [min(yAnim(2:ly)) max(yAnim(2:ly))]);
        zlim(ax1, [min(zAnim(2:lz)) max(zAnim(2:lz))]);
        view(ax1, 3)
        grid on
        zl = zlabel('Altitude (m, AGL)');
        xticks(min(xAnim):((max(xAnim)-min(xAnim))/4):max(xAnim));
        yticks(min(yAnim):((max(yAnim)-min(yAnim))/4):max(yAnim));
        zticks(min(zAnim):((max(zAnim)-min(zAnim))/5):max(zAnim));
```

```
        xtickformat('%.3f')
        ytickformat('%.3f')
        ztickformat('%.0f')
        set(ax1,'Color','k','xcolor','w','ycolor','w','zcolor','w','LineWidth',2)

        data_range1 = ceil(max(max(cAnim1(2:end)))) - ...
            floor(min(min(cAnim1(2:end)))) + 1;
        colormap(jet(data_range1*10));
        caxis([min(min(cAnim1(2:end))) max(max(cAnim1(2:end)))])
        cbh1 = colorbar();
        cbh1.Ticks = ...
            round(min(min(cAnim1(2:end))):((max(max(cAnim1(2:end)))-min(min(cAnim1(2
        ticks1 = strsplit(num2str(cbh1.Ticks));
        ax01 = axes('Position', cbh1.Position);
        edges1 = linspace(0,1,numel(ticks1)+1);
        centers1 = edges1(2:end)-((edges1(2)-edges1(1))/2);
        text(ones(size(centers1))*0.5, centers1, ticks1, ...
            'FontSize', ...
            cbh1.FontSize,'BackgroundColor','w','Margin',1,'HorizontalAlignment', ..
            'Center', 'VerticalAlignment', 'Middle');
        ax01.Visible = 'off';   %turn off new axes
        cbh1.Ticks = [];
        %% Humidity Plot %%%%%%%%%%%%
        subplot(1,3,2);
        ax2 = gca;
        title('Relative Humidity (%)');
        xlim(ax2, [min(xAnim(2:lx)) max(xAnim(2:lx))]);
        ylim(ax2, [min(yAnim(2:ly)) max(yAnim(2:ly))]);
        zlim(ax2, [min(zAnim(2:lz)) max(zAnim(2:lz))]);
        view(ax2, 3)
        grid on
        xticks(min(xAnim):((max(xAnim)-min(xAnim))/4):max(xAnim));
```

```
yticks(min(yAnim):((max(yAnim)-min(yAnim))/4):max(yAnim));
zticks(min(zAnim):((max(zAnim)-min(zAnim))/5):max(zAnim));
xtickformat('%.3f')
ytickformat('%.3f')
ztickformat('%.0f')
set(ax2,'Color','k','xcolor','w','ycolor','w','zcolor','w','LineWidth',2)

data_range2 = ceil(max(max(cAnim2(2:end)))) - ...
    floor(min(min(cAnim2(2:end)))) + 1;
colormap(jet(data_range2*10));
caxis([min(min(cAnim2(2:end))) max(max(cAnim2(2:end)))])
cbh2 = colorbar();
cbh2.Ticks = ...
    round(min(min(cAnim2(2:end))):((max(max(cAnim2(2:end)))-min(min(cAnim2(2
ticks2 = strsplit(num2str(cbh2.Ticks));
ax02 = axes('Position', cbh2.Position);
edges2 = linspace(0,1,numel(ticks2)+1);
centers2 = edges2(2:end)-((edges2(2)-edges2(1))/2);
text(ones(size(centers2))*0.5, centers2, ticks2, ...
    'FontSize', ...
    cbh2.FontSize,'BackgroundColor','w','Margin',1,'HorizontalAlignment', ..
    'Center', 'VerticalAlignment', 'Middle');
ax02.Visible = 'off';   %turn off new axes
cbh2.Ticks = [];
%% Pressure Plot %%%%%%%%%%%%
subplot(1,3,3);
ax3 = gca;
title('Pressure (mbar)');
xlim(ax3, [min(xAnim(2:lx)) max(xAnim(2:lx))]);
ylim(ax3, [min(yAnim(2:ly)) max(yAnim(2:ly))]);
zlim(ax3, [min(zAnim(2:lz)) max(zAnim(2:lz))]);
view(ax3, 3)
```

```
            grid on
            xticks(min(xAnim):((max(xAnim)-min(xAnim))/4):max(xAnim));
            yticks(min(yAnim):((max(yAnim)-min(yAnim))/4):max(yAnim));
            zticks(min(zAnim):((max(zAnim)-min(zAnim))/5):max(zAnim));
            xtickformat('%.3f')
            ytickformat('%.3f')
            ztickformat('%.0f')
            set(ax3,'Color','k','xcolor','w','ycolor','w','zcolor','w','LineWidth',2)

            data_range3 = ceil(max(max(cAnim3(2:end)))) - ...
                floor(min(min(cAnim3(2:end)))) + 1;
            colormap(jet(data_range3*10));
            caxis([min(min(cAnim3(2:end))) max(max(cAnim3(2:end)))])
            cbh3 = colorbar();
            cbh3.Ticks = ...
                round(min(cAnim3(2:end)):((max(cAnim3(2:end))-min(cAnim3(2:end)))/5):max
            ticks3 = strsplit(num2str(cbh3.Ticks));
            ax03 = axes('Position', cbh3.Position);
            edges3 = linspace(0,1,numel(ticks3)+1);
            centers3 = edges3(2:end)-((edges3(2)-edges3(1))/2);
            text(ones(size(centers3))*0.5, centers3, ticks3, ...
                'FontSize', ...
                cbh3.FontSize,'BackgroundColor','w','Margin',1,'HorizontalAlignment', ..
                'Center', 'VerticalAlignment', 'Middle');
            ax03.Visible = 'off';   %turn off new axes
            cbh3.Ticks = [];
            %% Position Control
            subfig = get(gcf,'children');
            set(subfig(1),'position',[.98 -.115 0 2.1]);  % Pressure ...
                color bar labels
            %set(subfig(1),'position',[0.98 0.19 0 .62]);
```

```
        set(subfig(2),'position',[.97 .2125 .02 .58]);     % ... Pressure color bar
        set(subfig(3),'position',[.7025 .06 .265 .88]);    % ... Pressure mian plotting area
        set(subfig(4),'position',[.6545 0.19 0 .62]);   % Humidity ... color bar labels
        set(subfig(5),'position',[.645 .2125 .02 .58]);      % ... Humidity color bar
        set(subfig(6),'position',[.3775 .06 .265 .88]);   % ... Humidity main plotting area
        set(subfig(7),'position',[.33 0.19 0 .62]);   % Temperature ... color bar labels
        set(subfig(8),'position',[.32 .2125 .02 .58]);      % ... Temperature color bar
        set(subfig(9),'position',[.053 .06 .265 .88]);    % ... Temperature main plotting area

        view(subfig(3),-45,20);
        view(subfig(6),-45,20);
        view(subfig(9),-45,20);

        AnimPos = [.025 .88 .09 .1];
        zl.Position = [34.9796 -97.51672 744.2306];
        %% Set Axis Colors
        ax1.ZTickLabel(1) = {'0'};
        ax2.ZTickLabel(1) = {'0'};
        ax3.ZTickLabel(1) = {'0'};

        for i=1:length(ax1.XTickLabel)
            ax1.XTickLabel{i} = ['\color[rgb]{0,0,0}' ...
                ax1.XTickLabel{i}];
```

```
            ax2.XTickLabel{i} = ['\color[rgb]{0,0,0}' ...
                ax2.XTickLabel{i}];
            ax3.XTickLabel{i} = ['\color[rgb]{0,0,0}' ...
                ax3.XTickLabel{i}];
        end
        for j=1:length(ax1.YTickLabel)
            ax1.YTickLabel{j} = ['\color[rgb]{0,0,0}' ...
                ax1.YTickLabel{j}];
            ax2.YTickLabel{j} = ['\color[rgb]{0,0,0}' ...
                ax2.YTickLabel{j}];
            ax3.YTickLabel{j} = ['\color[rgb]{0,0,0}' ...
                ax3.YTickLabel{j}];
        end
        for k=1:length(ax1.ZTickLabel)
            ax1.ZTickLabel{k} = ['\color[rgb]{0,0,0}' ...
                ax1.ZTickLabel{k}];
            ax2.ZTickLabel{k} = ['\color[rgb]{0,0,0}' ...
                ax2.ZTickLabel{k}];
            ax3.ZTickLabel{k} = ['\color[rgb]{0,0,0}' ...
                ax3.ZTickLabel{k}];
        end

        ax1.XTickLabel(i) = {' '};
        ax2.XTickLabel(i) = {' '};
        ax3.XTickLabel(i) = {' '};
        ax1.YTickLabel(j) = {' '};
        ax2.YTickLabel(j) = {' '};
        ax3.YTickLabel(j) = {' '};

        zl.Color = 'k';
        %% Animation Function
        open(animVid);
```

```
        for jj=1:interval:length(xAnim);
            try
                p1 = ...
                    patch(ax1,xAnim(1:jj),yAnim(1:jj),zAnim(1:jj),cAnim1(1:jj),'Edge
                    view(-45,45)
                p2 = ...
                    patch(ax2,xAnim(1:jj),yAnim(1:jj),zAnim(1:jj),cAnim2(1:jj),'Edge
                    view(-45,45)
                p3 = ...
                    patch(ax3,xAnim(1:jj),yAnim(1:jj),zAnim(1:jj),cAnim3(1:jj),'Edge
                    view(-45,45)

                delete(findall(fig10,'type','annotation'));
                Seconds = floor(mod(timer(jj,1),60));
                Minutes = fix(timer(jj,1)/60);

                if(Seconds<10)
                    String = {sprintf('Time ...
                        (min:sec)\n%.0f:0%.0f',Minutes,Seconds)};
                else
                    String = {sprintf('Time ...
                        (min:sec)\n%.0f:%.0f',Minutes,Seconds)};
                end
                annotation(fig10,'textbox',AnimPos,'String',String,'HorizontalAlignm


                pause(1/50); %Pause and grab frame
                frame = getframe(gcf); %get frame
                writeVideo(animVid, frame);
                cla(ax1);
                cla(ax2);
                cla(ax3);
```

```
            catch
                pause(1/50); %Pause and grab frame
                frame = getframe(gcf); %get frame
                writeVideo(animVid, frame);
                close(animVid);
                break

            end
        end
        close(animVid);
    elseif(strcmpi(isometric,'Yes'))
        %% Animation Setup

        if(strcmpi(plotTitle,''))
            plotTitle = 'Default';
        end
        WindSpacing = NKF2_table(1:5:end,:);

        if(height(GPS_table)<height(WindSpacing))
            WindSpacing = WindSpacing(1:height(GPS_table),:);
        else GPS_table = GPS_table(1:height(WindSpacing),:);
        end

        Lat = table2array(GPS_table(:,6));
        Long = table2array(GPS_table(:,7));
        Alt = table2array(GPS_table(:,9));
        minAlt = min(Alt);
        if(minAlt<0)
            Alt=Alt+abs(min(Alt));
        elseif(minAlt>0)
            Alt=Alt-min(Alt);
        end
```

```
        NW = table2array(WindSpacing(:,4));
        EW = table2array(WindSpacing(:,5));
        WindMag = round(table2array(WindSpacing(:,6)),1);

        limHigh = max(max(abs(EW),abs(NW)));
        limLow = -limHigh;

        fig10 = figure(10);
        fig10.Position=[130 130 800 600];
        fig10.Resize = 'off';

        xAnim1=[nan;Long(:,1)];
        yAnim1=[nan;Lat(:,1)];
        zAnim1=[nan;Alt(:,1)];

        lx = length(xAnim1);
        ly = length(yAnim1);
        lz = length(zAnim1);
        %% Wind Plot %%%%%%%%%%%%%
        subplot(1,2,1)
        % Wind Plotting Data
        ax1 = gca;
        % Modified compass figure with higher radial limit
        c1=compass(ax1,[0 0],[limHigh 0]);
        set(c1(1),'visible','off')
        set(ax1,'ydir','reverse')
        labels = findall(ax1,'type','text');
        % Altering the angular label
        set(findall(ax1, 'String', '0'),'String', 'N');
        set(findall(ax1, 'String', '90'),'String', 'E');
        set(findall(ax1, 'String', '180'),'String', 'S');
        set(findall(ax1, 'String', '270'),'String', 'W');
```

```
        set(labels(3:6),'visible','off');
        set(labels(9:12),'visible','off');

        view(ax1, -90,90)
        title({'Pixhawk-Estimated Wind Rose';''});
        %% GPS Plot %%%%%%%%%%%%
        subplot(1,2,2)
        % GPS Plotting data
        ax2 = gca;
        xlim(ax2, [min(Long) max(Long)]);
        ylim(ax2, [min(Lat) max(Lat)]);
        zlim(ax2, [min(Alt) max(Alt)]);
        xl = xlabel('Longitude');
        yl = ylabel('Lattitude');
        zl = zlabel('Alt (m, AGL)');

        if(strcmpi(plotTitle,'Default'))
            plotTitle = baseNameNoExtDFL;
        end

        t = title({plotTitle,' '},'Interpreter', 'none');
        t.FontSize = 16;

        view(ax2, 3)
        xticks(min(Long):((max(Long)-min(Long))/4):max(Long));
        yticks(min(Lat):((max(Lat)-min(Lat))/4):max(Lat));
        zticks(min(Alt):((max(Alt)-min(Alt))/4):max(Alt));
        xtickformat('%.3f')
        ytickformat('%.3f')
        ztickformat('%.0f')
        grid
        set(ax2,'Color','k','xcolor','w','ycolor','w','zcolor','w','LineWidth',2)
```

```
axis(ax2,[min(Long) max(Long) min(Lat) max(Lat) min(Alt) ...
    max(Alt)]);
%% Set Axis Colors
for i=1:length(ax2.XTickLabel)
    ax2.XTickLabel{i} = ['\color[rgb]{0,0,0}' ...
        ax2.XTickLabel{i}];
end
for j=1:length(ax2.YTickLabel)
    ax2.YTickLabel{j} = ['\color[rgb]{0,0,0}' ...
        ax2.YTickLabel{j}];
end
for k=1:length(ax2.ZTickLabel)
    ax2.ZTickLabel{k} = ['\color[rgb]{0,0,0}' ...
        ax2.ZTickLabel{k}];
end

%ax2.XTickLabel{i} = [];   ax2.YTickLabel{j} = [];

xl.Color = 'k';
yl.Color = 'k';
zl.Color = 'k';
%% Position Control
subfig1 = get(gcf,'children');
set(subfig1(1),'position',[.1 .075 .8 .8]);      % GPS ...
    animation space
set(subfig1(2),'position',[.725 .725 .225 .225]);   % Wind rose

view(subfig1(1),-45,45);

display('Beginning animation sequence');
%% Animation Function
```

```
            if(strcmpi(recordAnimation,'Yes'));
                open(animVid);

                for i=1:animateSpeed:length(EW);
                    c1=compass(ax1,[0 EW(i)],[limHigh NW(i)]);
                    set(c1(1),'visible','off');
                    labels = findall(ax1,'type','text');
                    view(ax1, -90,90);
                    set(ax1,'ydir','reverse');
                    set(labels(13),'visible','off');
                    set(labels(14),'visible','off');

                    set(c1(2),'LineWidth',1.5,'color','red');
                    %annotation('textbox',[.645 .55 .3 ...
                        .3],'String',sprintf('%s ...
                        %s',num2str(WindMag(i)),'m/s'),'FitBoxToText','on');

                    %removedthis
                    if(i≤(animateTailLength+1))
                        pH1 = ...
                            patch(ax2,'XData',xAnim1(1:i),'YData',yAnim1(1:i),'ZData',zA
                        pH2 = ...
                            patch(ax2,'XData',xAnim1(i),'YData',yAnim1(i),'ZData',zAnim1
                    else
                        pH1 = patch(ax2,'XData',[nan; ...
                            xAnim1((i-animateTailLength):i)],'YData',[nan; ...
                            yAnim1((i-animateTailLength):i)],'ZData',[nan; ...
                            zAnim1((i-animateTailLength):i)],'EdgeColor','r','FaceColor'
                        pH2 = ...
                            patch(ax2,'XData',xAnim1(i),'YData',yAnim1(i),'ZData',zAnim1
                    end

```

```
                pause(1/(50)) %Pause and grab frame
                frame = getframe(gcf); %get frame
                writeVideo(animVid, frame);
                delete(findall(fig10,'type','annotation'));
                cla(ax2);

            end

            close(animVid);
        else
            for i=1:animateSpeed:length(EW);

                c1=compass(ax1,[0 EW(i)],[limHigh NW(i)]);
                set(c1(1),'visible','off');
                labels = findall(ax1,'type','text');
                view(ax1, -90,90);
                set(ax1,'ydir','reverse');
                set(findall(ax1, 'String', '0'),'String', 'N');
                set(findall(ax1, 'String', '90'),'String', 'E');
                set(findall(ax1, 'String', '180'),'String', 'S');
                set(findall(ax1, 'String', '270'),'String', 'W');
                set(labels(3:6),'visible','off');
                set(labels(9:14),'visible','off');

                set(c1(2),'LineWidth',1.5,'color','red');
                annotation('textbox',[.645 .55 .3 ...
                    .3],'String',sprintf('%s ...
                    %s',num2str(WindMag(i)),'m/s'),'FitBoxToText','on');

                if(i≤(animateTailLength+1))
                    pH1 = ...
                        patch(ax2,'XData',xAnim1(1:i),'YData',yAnim1(1:i),'ZData',zA
```

```
                        pH2 = ...
                            patch(ax2,'XData',xAnim1(i),'YData',yAnim1(i),'ZData',zAnim1
                    else
                        pH1 = patch(ax2,'XData',[nan; ...
                            xAnim1((i-animateTailLength):i)],'YData',[nan; ...
                            yAnim1((i-animateTailLength):i)],'ZData',[nan; ...
                            zAnim1((i-animateTailLength):i)],'EdgeColor','r','FaceColor'
                        pH2 = ...
                            patch(ax2,'XData',xAnim1(i),'YData',yAnim1(i),'ZData',zAnim1
                    end

                    pause(1/(50)) %Pause and grab frame

                    delete(findall(fig10,'type','annotation'));
                    cla(ax2);

                end
            end
        else
            %% Animation Setup

            if(strcmpi(plotTitle,''))
                plotTitle = 'Default';
            end
            WindSpacing = NKF2_table(1:5:end,:);

            if(height(GPS_table)<height(WindSpacing))
                WindSpacing = WindSpacing(1:height(GPS_table),:);
            else GPS_table = GPS_table(1:height(WindSpacing),:);
            end

            Lat = table2array(GPS_table(:,6));
```

```
        Long = table2array(GPS_table(:,7));
        Alt = table2array(GPS_table(:,9));
        minAlt = min(Alt);
        if(minAlt<0)
            Alt=Alt+abs(min(Alt));
        elseif(minAlt>0)
            Alt=Alt-min(Alt);
        end
        NW = table2array(WindSpacing(:,4));
        EW = table2array(WindSpacing(:,5));
        WindMag = round(table2array(WindSpacing(:,6)),1);

        limHigh = max(max(abs(EW),abs(NW)));
        limLow = -limHigh;

        fig10 = figure(10);
        fig10.Position=[130 130 800 600];
        fig10.Resize = 'off';

        xAnim1=[nan;Long(:,1)];
        yAnim1=[nan;Lat(:,1)];
        zAnim1=[nan;Alt(:,1)];

        lx = length(xAnim1);
        ly = length(yAnim1);
        lz = length(zAnim1);
        %% USGS Mapping Data
baseURL = "https://basemap.nationalmap.gov/ArcGIS/rest/services";
usgsURL = baseURL + "/BASEMAP/MapServer/tile/${z}/${y}/${x}";
basemaps = ["USGSImageryOnly" "USGSImageryTopo" "USGSTopo" ...
    "USGSShadedReliefOnly" "USGSHydroCached"];
```

```
    displayNames = ["USGS Imagery" "USGS Topographic Imagery" "USGS ...
       Shaded Topographic Map" "USGS Shaded Relief" "USGS Hydrography"];
    attribution = 'Credit: U.S. Geological Survey';
            %% GPS Plot %%%%%%%%%%%%
            % GPS Plotting data
            ax2 = geoaxes();
            geobasemap('satellite')
            basemapx = basemaps(2);
            url = replace(usgsURL,"BASEMAP",basemapx);
            view(ax2,2)

            if(strcmpi(plotTitle,'Default'))
                plotTitle = baseNameNoExtDFL;
            end

            t = title({plotTitle,' '},'Interpreter', 'none');
            t.FontSize = 16;

            display('Beginning animation sequence');
            %% Animation Function
            if(strcmpi(recordAnimation,'Yes'));
                open(animVid);

                for i=1:animateSpeed:length(EW);

                    %annotation('textbox',[.645 .55 .3 ...
                       .3],'String',sprintf('%s ...
                       %s',num2str(WindMag(i)),'m/s'),'FitBoxToText','on');

                    if(i≤(animateTailLength+1))
                        hold on
```

```
                        pH1 = ...
                            geoplot(ax2,yAnim1(1:i),xAnim1(1:i),'r','LineWidth',animateTa
                        pH2 = ...
                            geoplot(ax2,yAnim1(i),xAnim1(i),'Marker','o','MarkerSize',an
                    else
                        hold on
                        pH1 = geoplot(ax2,[nan; ...
                            yAnim1((i-animateTailLength):i)],[nan; ...
                            xAnim1((i-animateTailLength):i)],'r','LineWidth',animateTail
                        pH2 = ...
                            geoplot(ax2,yAnim1(i),xAnim1(i),'Marker','o','MarkerSize',an
                    end

                    geobasemap('satellite')
                    geolimits([min(yAnim1)-.0005 ...
                       max(yAnim1)+.0005],[min(xAnim1)-.0005 ...
                       max(xAnim1)+.0005]);

                    pause(1/(50)) %Pause and grab frame
                    frame = getframe(gcf); %get frame
                    writeVideo(animVid, frame);
                    delete(findall(fig10,'type','annotation'));
                    cla(ax2);
                    hold off

                end

                close(animVid);
            else
                for i=1:animateSpeed:length(EW);

                    if(i≤(animateTailLength+1))
```

```
                    hold on
                    pH1 = ...
                        geoplot(ax2,yAnim1(1:i),xAnim1(1:i),'r','LineWidth',animateT
                    pH2 = ...
                        geoplot(ax2,yAnim1(i),xAnim1(i),'Marker','o','MarkerSize',an
                else
                    hold on
                    pH1 = geoplot(ax2,[nan; ...
                        yAnim1((i-animateTailLength):i)],[nan; ...
                        xAnim1((i-animateTailLength):i)],'r','LineWidth',animateTail
                    pH2 = ...
                        geoplot(ax2,yAnim1(i),xAnim1(i),'Marker','o','MarkerSize',an
                end

                geobasemap('satellite')
                geolimits([min(yAnim1)-.0005 ...
                    max(yAnim1)+.0005],[min(xAnim1)-.0005 ...
                    max(xAnim1)+.0005]);

                pause(1/(50)) %Pause and grab frame
                delete(findall(fig10,'type','annotation'));
                cla(ax2);
                hold off

            end
        end
    end
end
%% Hover/Profile Averaging
if(strcmpi(hoverProfile,'Hover') | strcmpi(hoverProfile,'Profile'))
    warning('off','MATLAB:ISMEMBER:RowsFlagIgnored');
    warning('off','MATLAB:table:RowsAddedExistingVars');
```

```
%% Merge Suite and GPS data
for i=1:height(TPH_table)
    currentTPH = table2cell(TPH_table(i,5));
    currentTPH = currentTPH{1};
    A(i,1) = cellstr(currentTPH(1:end-2));
end

for i=1:height(GPS_table)
    currentGPS = table2cell(GPS_table(i,5));
    currentGPS = currentGPS{1};
    B(i,1) = cellstr(datetime(currentGPS,'Format','HH:mm:ss.S'));
end

idx1 = ismember(A, B, 'rows');

count=1;
for i=1:length(idx1)
    if(idx1(i) == 1)
        iteration1(count,1) = i;
        tempVals(count,1) = A(i);
        count=count+1;
    end
end

GPS_time_alone = cellstr(GPS_full_time(:,1:end-2));
idx2 = ismember(GPS_time_alone,tempVals,'rows');

count=1;
for i=1:length(idx2)
    if(idx2(i) == 1)
        iteration2(count,1) = i;
        count=count+1;
```

```
            end
        end

        GPS_table_new = GPS_table(iteration2,:);

        count=1;
        for i=1:height(GPS_table_new)
            test = ...
                find(BARO(:,1)≥table2array(GPS_table_new(i,1)),1,'first');
            if (test ≥ 1)
                iteration3(count,1) = test;
                count=count+1;
            end
        end

        itLen = [length(iteration1) length(iteration2) length(iteration3)];

        finalTable = [GPS_table(iteration2(1:min(itLen)),4:10) ...
            BARO_table(iteration3(1:min(itLen)),4:5) ...
            TPH_table(iteration1(1:min(itLen)),6:11)];

        baseFileName = sprintf('%s_Parsed_Pix_Suite.csv', ...
            baseNameNoExtDFL);
        fullOutputMatFileName = fullfile(folderDFL, baseFileName);
        % Write data to text file
        writetable(finalTable, fullOutputMatFileName);
        %% Hover with 10 second averaging
        if(strcmpi(hoverProfile,'Hover'))
            count=1;
            for i=1:50:(height(finalTable)-50)
                tempArray1(count,1) = finalTable(i,1);
                tempArray1(count,2) = finalTable(i,2);
```

```
                tempArray1(count,3:7) = finalTable(i,3:7);
                tempArray1(count,8:15) = ...
                    array2table(mean(table2array(finalTable(i:(i+50),8:15))));
                count=count+1;
            end

            tempArray1.Properties.VariableNames = ...
                finalTable.Properties.VariableNames;
            baseFileName = sprintf('%s_Suite_10sec_Av.csv', ...
                baseNameNoExtDFL);
            fullOutputMatFileName = fullfile(folderDFL, baseFileName);
            % Write data to text file
            writetable(tempArray1, fullOutputMatFileName);
        end
        %% Profile with 10 meter averaging
        if(strcmpi(hoverProfile,'Profile'))
            count=1;
            iter=1;
            startAlt = table2array(finalTable(iter,6));
            for i=1:height(finalTable)
                curAlt = table2array(finalTable(i,6));

                if((abs(curAlt - startAlt)) ≥ 10)
                    tempArray2(count,1) = ...
                        array2table(mean(table2array(finalTable(iter:i,1))));
                    tempArray2(count,2) = ...
                        array2table(mean(table2array(finalTable(iter:i,2))));
                    tempArray2(count,3:15) = ...
                        array2table(mean(table2array(finalTable(iter:i,3:15))));
                    iter=i;
                    count=count+1;
                    startAlt = table2array(finalTable(iter,6));
```

```
                end
            end

            tempArray2.Properties.VariableNames = ...
                finalTable.Properties.VariableNames;
            baseFileName = sprintf('%s_Suite_10m_Av.csv', ...
                baseNameNoExtDFL);
            fullOutputMatFileName = fullfile(folderDFL, baseFileName);
            % Write data to text file
            writetable(tempArray2, fullOutputMatFileName);
        end
        warning('on','MATLAB:ISMEMBER:RowsFlagIgnored');
        warning('on','MATLAB:table:RowsAddedExistingVars');
    end
else

    %% USGS Mapping Data
    baseURL = "https://basemap.nationalmap.gov/ArcGIS/rest/services";
    usgsURL = baseURL + "/BASEMAP/MapServer/tile/${z}/${y}/${x}";
    basemaps = ["USGSImageryOnly" "USGSImageryTopo" "USGSTopo" ...
        "USGSShadedReliefOnly" "USGSHydroCached"];
    displayNames = ["USGS Imagery" "USGS Topographic Imagery" "USGS ...
        Shaded Topographic Map" "USGS Shaded Relief" "USGS Hydrography"];
    attribution = 'Credit: U.S. Geological Survey';
    %% Load All GPS Files

    [baseFileName, folder]=uigetfile('*.mat','Select the INPUT DATA ...
        FILE(s)','MultiSelect','on');
    disp('Parsing and merging Pixhawk files.')

    if(strcmpi(simultaneous,'Yes'))
        for i=1:length(baseFileName)
```

```
            fullInputMatFileName(i) = fullfile(folder, baseFileName(i));
            load(char(fullInputMatFileName(i)),'GPS');

            Time{:,i} = GPS(2:end,4);
            Speed{:,i} = GPS(2:end,11);
            Lat{:,i} = GPS(2:end,8);
            Long{:,i} = GPS(2:end,9);
            Alt{:,i} = GPS(2:end,10);
            minTime(i) = min(GPS(2:end,4));
            maxTime(i) = max(GPS(2:end,4));
            clear GPS

        end
    else
        for i=1:length(baseFileName)
            fullInputMatFileName(i) = fullfile(folder, baseFileName(i));
            load(char(fullInputMatFileName(i)),'GPS');

            rawLat{:,i} = GPS(2:end,8);
            rawLong{:,i} = GPS(2:end,9);
            rawAlt(:,i) = GPS(2:end,10);
            rawSpeed{:,i} = GPS(2:end,11);
            clear GPS

        end

        for k=1:i
            Speed = [rawSpeed{k}];
            tempLat = [rawLat{k}];
            tempLong = [rawLong{k}];
            tempAlt = [rawAlt{k}];
            Lat{k} = tempLat(Speed>.5);
```

```
            Long{k} = tempLong(Speed>.5);
            Alt{k} = tempAlt(Speed>.5);
            clear tempLat tempLong tempAlt Speed
        end
    end


    disp('Completed Pixhawk file manipulation.');
    %% Data Merging
    % Combine all datasets into arrays of same size
    % find all takeoff times
    % find first aircraft to takeoff
    % convert times to basic plotting number (incriment of 1)
    % keep lat/long of all grounded aircraft, but pad with plotting number
    % plot all aircraft according to plotting number, with lat/long being
    %   the only differntiating factor

    if(strcmpi(simultaneous,'Yes'))

        minT = min(minTime);
        maxT = max(maxTime);
        translate(:,1) = minT:200:maxT;
        translate(:,2) = 1:length(minT:200:maxT);
        startLat = cellfun(@(v)v(1),Lat);
        startLong = cellfun(@(v)v(1),Long);
        startAlt = cellfun(@(v)v(1),Alt);
        endLat = cellfun(@(v)v(end),Lat);
        endLong = cellfun(@(v)v(end),Long);
        endAlt = cellfun(@(v)v(end),Alt);

        finLat(1:length(translate(:,1)),1:i)=0;
        finLong(1:length(translate(:,1)),1:i)=0;
```

```
        finAlt(1:length(translate(:,1)),1:i)=0;
        finTime(:,1) = translate(:,2);
        finSpeed(1:length(translate(:,1)),1:i) = 0;

        for(j = 1:i)
            curTime = round(cell2mat(Time(j)),-2);
            curLat = cell2mat(Lat(j));
            curLong = cell2mat(Long(j));
            curAlt = cell2mat(Alt(j));
            curSpeed = cell2mat(Speed(j));
            count=0;
            for k = 1:length(curTime);
                Catch = find(curTime(k,1)==translate(:,1),1,'first');
                if(Catch≥0)
                    count=count+1;
                    if(count ≠ 1)
                        finLat(Catch,j) = curLat(k,1);
                        finLong(Catch,j) = curLong(k,1);
                        finAlt(Catch,j) = curAlt(k,1);
                        finSpeed(Catch,j) = curSpeed(k,1);
                    else
                        finLat(1:Catch,j)=curLat(k,1);
                        finLong(1:Catch,j)=curLong(k,1);
                        finAlt(1:Catch,j)=curAlt(k,1);
                        finSpeed(1:Catch,j) = curSpeed(k,1);
                    end
                end
            end
            finLat(Catch:end,j) = curLat(end,1);
            finLong(Catch:end,j) = curLong(end,1);
            finAlt(Catch:end,j) = curAlt(end,1);
            finSpeed(Catch:end,j) = curSpeed(end,1);
```

```
            clear curTime curLat curLong curAlt
        end

        for col = 1:size(finLat,2)
            for row = 1:size(finLat,1)
                if finLat(row,col) == 0
                    finLat(row,col) = finLat(row+1,col);
                end
            end
        end

        for col = 1:size(finLong,2)
            for row = 1:size(finLong,1)
                if finLong(row,col) == 0
                    finLong(row,col) = finLong(row+1,col);
                end
            end
        end

        for col = 1:size(finAlt,2)
            for row = 1:size(finAlt,1)
                if finAlt(row,col) == 0
                    finAlt(row,col) = finAlt(row-1,col);
                end
            end
        end

        for(k = 1:i)
            try startSpeed(k) = find(finSpeed(:,k) ≥1,1,'first');
                endSpeed(k) = find(finSpeed(:,k) ≥1,1,'last');

            catch startSpeed(k) = -1;
```

```
                endSpeed(k) = -1;
            end
        end

        firstSpeed = min(startSpeed(startSpeed > 0));
        lastSpeed = max(endSpeed(endSpeed > 0));

        startSpeed(startSpeed<0) = firstSpeed;
        endSpeed(endSpeed<0) = lastSpeed;

        trimStart = min(startSpeed);
        trimEnd = max(endSpeed);

        finLat = finLat(trimStart:trimEnd,:);
        finLong = finLong(trimStart:trimEnd,:);
        finAlt = finAlt(trimStart:trimEnd,:);
        finTime = finTime(trimStart:trimEnd,:);
    end
    %% Animation Function

    % Color Setup
    ColOrd = [1 0 0; 0 1 0; 0 0 1; 1 1 1; 0 0 0; 1 1 0; 1 0 1; 0 1 1];
    [m,n] = size(ColOrd);

    % Animation of the Pixhawk Log from recorded GPS coordinates
    if(strcmpi(simultaneous,'Yes'))
        minLat = min(min(finLat));
        maxLat = max(max(finLat));
        minLong = min(min(finLong));
        maxLong = max(max(finLong));
        minAlt = min(min(finAlt));
        maxAlt = max(max(finAlt));
```

```
        if(strcmpi(animation,'No'))

            disp('Beginning plot setup.');

            fig1=figure(1);
            fig1.Position = [10 50 560 725];
            fig1.Resize = 'off';

            ax = geoaxes();
            geobasemap('satellite')
            basemapx = basemaps(2);
            url = replace(usgsURL,"BASEMAP",basemapx);
            view(ax,2)

            disp('Starting the plotting process.');

            for(j =1:i)
                ColRow = rem(j,m);
                if ColRow == 0
                    ColRow = m;
                end
                Col = ColOrd(ColRow,:);

                hold on

                geoplot(ax,finLat(:,j),finLong(:,j),'Color',Col,'LineWidth',.8);
                geoplot(ax,finLat(end,j),finLong(end,j),'LineStyle','none','Marker','o',

                geobasemap('satellite')
                geolimits([minLat-.0005 maxLat+.0005],[minLong-.0005 ...
                    maxLong+.0005]);
```

```
            end

            title(plotTitle);

            baseFileNamePic = sprintf('MergedData_Image10');
            fullParsedMatFileNamePic = fullfile(folder, baseFileNamePic);

            saveas(fig,fullParsedMatFileNamePic,'png')


            disp('Successfully completed plot.');

        elseif (strcmpi(animation,'Yes'))

            disp('Beginning animation setup.');

            animVid = VideoWriter('Kessler 8-4-2020, ...
                TriplePlot','MPEG-4'); %open video file
            animVid.FrameRate = 20;  %can adjust this, 5 - 10 works ...
                well for me
            animVid.Quality = 100;

            tot = ceil(length(finLat(:,1))/iterationSkip);

            fig1 = figure(1);
            fig1.Position = [10 50 560 725];
            fig1.Resize = 'off';

            if(strcmpi(isometric,'No'))

                ax = geoaxes();
                geobasemap('satellite')
```

```
            basemapx = basemaps(2);
            url = replace(usgsURL,"BASEMAP",basemapx);
            view(ax,2)
            title(plotTitle);

            disp('Starting the animation process.');

            open(animVid);

            ptot = 0;

            for jj=1:iterationSkip:length(finLat(:,1))
                hold on
                for k = 1:i
                    ColRow = rem(k,m);
                    if ColRow == 0
                        ColRow = m;
                    end
                    Col = ColOrd(ColRow,:);
                    hold on

                    geoplot(ax,finLat(1:jj,k),finLong(1:jj,k),'Color',Col,'LineWidth
                    geoplot(ax,finLat(jj,k),finLong(jj,k),'LineStyle','none','Marker

                    geobasemap('satellite')
                    geolimits([minLat-.0005 ...
                        maxLat+.0005],[minLong-.0005 maxLong+.0005]);
                end
                pause(.001);
                frame = getframe(gcf);
                writeVideo(animVid,frame);
                ptot = ptot+1;
```

```
                display = sprintf('Iteration: %d of %d, %s%% ...
                    complete',ptot,tot,num2str(round(ptot/tot*100,1)));
                disp(display);
                cla(ax);
                hold off
            end

            close(animVid);

            disp('Successfully completed animation.');
        else
            ax = gca();
            axis([minLat maxLat minLong maxLong minAlt maxAlt]);
            view(ax, 3)
            xticks(minLat:((maxLat-minLat)/4):maxLat);
            yticks(minLong:((maxLong-minLong)/4):maxLong);
            zticks(minAlt:((maxAlt-minAlt)/5):maxAlt);
            xtickformat('%.1f')
            ytickformat('%.1f')
            ztickformat('%.0f')
            title(plotTitle);
            grid

            view(ax,[azimuthAngle elevationAngle]);

            disp('Starting the animation process.');

            open(animVid);

            ptot = 0;

            for jj=1:iterationSkip:length(finLat(:,1))
```

```
                        hold on
                        for k = 1:i
                            ColRow = rem(k,m);
                            if ColRow == 0
                                ColRow = m;
                            end
                            Col = ColOrd(ColRow,:);
                            hold on

                            plot3(ax,finLat(1:jj,k),finLong(1:jj,k),finAlt(1:jj,k),'Color',C
                            plot3(ax,finLat(jj,k),finLong(jj,k),finAlt(jj,k),'LineStyle','no

                        end
                        pause(.001);
                        frame = getframe(gcf);
                        writeVideo(animVid,frame);
                        ptot = ptot+1;
                        display = sprintf('Iteration: %d of %d, %s%% ...
                           complete',ptot,tot,num2str(round(ptot/tot*100,1)));
                        disp(display);
                        cla(ax);
                        hold off
                    end

                    close(animVid);

                    disp('Successfully completed animation.');
                end
            end

        else
            % Animation of the Pixhawk Log from recorded GPS coordinates
```

```
        minLat = min(min(cell2mat(Lat(:))));
        maxLat = max(max(cell2mat(Lat(:))));
        minLong = min(min(cell2mat(Long(:))));
        maxLong = max(max(cell2mat(Long(:))));
        minAlt = min(min(cell2mat(Alt(:))));
        maxAlt = max(max(cell2mat(Alt(:))));

        if(strcmpi(animation,'No'))

            disp('Beginning plot setup.');

            fig1=figure(1);
            fig1.Position = [10 50 750 600];
            fig1.Resize = 'off';

            ax = geoaxes();
            geobasemap('satellite')
            basemapx = basemaps(2);
            url = replace(usgsURL,"BASEMAP",basemapx);
            view(ax,2)

            disp('Starting the plotting process.');

            for(j =1:i)
                ColRow = rem(j,m);
                if ColRow == 0
                    ColRow = m;
                end
                Col = ColOrd(ColRow,:);

                hold on

```

```
                geoplot(ax,cell2mat(Lat(j)),cell2mat(Long(j)),'Color',Col);

                geobasemap('satellite')
                geolimits([minLat-.0005 maxLat+.0005],[minLong-.0005 ...
                    maxLong+.0005]);
            end
            title(plotTitle);

            baseFileNamePic = sprintf('MergedData_Image10');
            fullParsedMatFileNamePic = fullfile(folder, baseFileNamePic);

            saveas(fig1,fullParsedMatFileNamePic,'png')


            disp('Successfully completed plot.');

        else

            disp('Beginning animation setup.');

            % Get the name of the intput.mat file and save as ...
                input_parsed.mat
            baseFileName = sprintf('MergedData_Animation1');
            fullParsedMatFileName = fullfile(folder, baseFileName);

            % Initialize total iteration count
            tot=0;
            for l=1:i
                tot = tot+numel(Lat{l});
            end

            tot=ceil(tot/iterationSkip)+1;
```

```
        animVid = VideoWriter(fullParsedMatFileName,'MPEG-4');
        animVid.FrameRate = 20;  %can adjust this, 5 - 10 works ... well for me
        animVid.Quality = 100;

        fig1 = figure(1);
        fig1.Position = [10 50 750 600];
        fig1.Resize = 'off';

        ax = geoaxes();
        geobasemap('satellite')
        basemapx = basemaps(2);
        url = replace(usgsURL,"BASEMAP",basemapx);
        view(ax,2)
        title(plotTitle);

        disp('Starting the animation process.');

        open(animVid);

        ptot = 0;
        for k=1:i
            hold on
            tLat = cell2mat(Lat(k));
            tLong = cell2mat(Long(k));
            for jj=1:iterationSkip:length(tLat)
                ColRow = rem(k,m);
                if ColRow == 0
                    ColRow = m;
                end
                Col = ColOrd(ColRow,:);
```

```
        hold on


        geoplot(ax,tLat(1:jj),tLong(1:jj),'Color',Col);
        geoplot(ax,tLat(jj),tLong(jj),'LineStyle','none','Marker','o','Marke

        if(k>1)
            for x = 1:(k-1)
                ColRow = rem(x,m);
                if ColRow == 0
                    ColRow = m;
                end
                Col = ColOrd(ColRow,:);
                geoplot(ax,cell2mat(Lat(x)),cell2mat(Long(x)),'Color',Col);
            end
        end


        geobasemap('satellite')
        %geolimits([minLat-.0005 ...
           maxLat+.0005],[minLong-.0005 maxLong+.0005]);

        pause(.001);
        frame = getframe(gcf);
        writeVideo(animVid,frame);
        ptot = ptot+1;
        display = sprintf('Iteration: %d of %s, %s%% ...
           complete',ptot,num2str(tot),num2str(round(ptot/tot*100,1)));
        disp(display);
        cla(ax);
        hold off
    end
```

```
                end


                close(animVid);

                disp('Successfully completed animation.');

            end
        end
end
toc
disp('All operations completed.');

%% Functions

% make a spectral analysis function

function [P1, f] = freq_this(X,Fs, L)

T = 1/Fs;               % Sampling period
t = (0:L-1)*L;          % Time vector

Y = fft(X);

P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);

f = Fs*(0:(L/2))/L;

end

```

```
% Implementation of the MHPA as displayed in Rautenberg2018 this is an
% itteration on Axford1968


%%%%%%[VN, VE, VD ...
    (m/s)][Angle(Degrees)][U(m/s)]%%%%%%%%%%%%%%[U,alpha,beta][VN,VE,VD][phi,theta,psi]
function [windVELOCITY, windDIRECTION, windSPEED] = ...
    MHPA_Rautenberg2018(aDATA, gDATA, bDATA)

% tTEMP is singular does not need to be defined

% airdata
u_a=aDATA(:,1);
alpha=aDATA(:,2);
beta= aDATA(:,3);

% ground data
VN=gDATA(:,1);
VE=gDATA(:,2);
VD=gDATA(:,3);

%body data
phi=bDATA(:,1);
theta=bDATA(:,2);
psi=bDATA(:,3);

R=287; %J kg^-1 K^-1
cp=1004; % J kg^-1 K^-1

k=R/cp; % poisson number

for i=1:length(bDATA(:,1))
```

```
    RM(:,i) = ...
        [cosd(psi(i))*cosd(theta(i))+tand(alpha(i))*(sind(phi(i))*sind(psi(i))+ ...
        ...
    cosd(phi(i))*cosd(psi(i))*sind(theta(i)))+tand(beta(i))*(cosd(psi(i))*sind(phi(i))*s
    cosd(phi(i))*sind(psi(i)));
    cosd(theta(i))*sind(psi(i))+tand(alpha(i))*(cosd(phi(i))*sind(psi(i))*sind(theta(i))
        ...
    tand(beta(i))*(cosd(phi(i))*cosd(psi(i))+sind(phi(i))*sind(psi(i))*sind(theta(i)));
    -sind(theta(i))+cosd(phi(i))*cosd(theta(i))*tand(alpha(i))+cosd(theta(i))*sind(phi(i

    D(i,1) = sqrt(1+tand(beta(i))^2+tand(alpha(i))^2);

    bodyU(i,1)=abs(u_a(i))/D(i);
end

%D = sqrt(1+tan(beta)^2+tan(alpha)^2);

groundMHP = bodyU.*RM';

windGROUND = abs(gDATA) - abs(groundMHP);

windVELOCITY = windGROUND;
for i=1:length(windGROUND)
    windDIRECTION(i) = atan2d(windVELOCITY(i,1),windVELOCITY(i,2))+90;
end
%windDIRECTION=rad2deg(windDIRECTION);
windSPEED=sqrt(windVELOCITY(:,1).^2+windVELOCITY(:,2).^2+windVELOCITY(:,3).^2);
end

% Implementation of the MHPA as derived in Lenschow1989 (incorporates
% angular rates)

```

```
%%%%%%[VN, VE, VD ...
    (m/s)][Angle(Degrees)][U(m/s)]%%%%%%%%%%%%%%[U,alpha,beta][VN,VE,VD][phi,theta,psi]
function [windVELOCITY, windDIRECTION, windSPEED] = MHPA_wRATES(aDATA, ...
    gDATA, bDATA, rDATA, L)

% tTEMP is singular does not need to be defined

% airdata
u_a=aDATA(:,1);
alpha=aDATA(:,2);
beta= aDATA(:,3);

% ground data
VN=gDATA(:,1);
VE=gDATA(:,2);
VD=gDATA(:,3);

%body data
phi=bDATA(:,1);
theta=bDATA(:,2);
psi=bDATA(:,3);

%rate data

p=rDATA(:,1);
q=rDATA(:,2);
r=rDATA(:,3);

R=287; %J kg^-1 K^-1
cp=1004; % J kg^-1 K^-1

```

```
k=R/cp; % poisson number

for i=1:length(bDATA(:,1))
    RM_gw(:,i) = ...
        [cosd(psi(i))*cosd(theta(i))+tand(alpha(i))*(sind(phi(i))*sind(psi(i))+ ...
        ...
    cosd(phi(i))*cosd(psi(i))*sind(theta(i)))+tand(beta(i))*(cosd(psi(i))*sind(phi(i))*s
    cosd(phi(i))*sind(psi(i)));
    cosd(theta(i))*sind(psi(i))+tand(alpha(i))*(cosd(phi(i))*sind(psi(i))*sind(theta(i))
        ...
    tand(beta(i))*(cosd(phi(i))*cosd(psi(i))+sind(phi(i))*sind(psi(i))*sind(theta(i)));
    -sind(theta(i))+cosd(phi(i))*cosd(theta(i))*tand(alpha(i))+cosd(theta(i))*sind(phi(i

%     RM_gb(:,i) = [cosd(psi(i))*cosd(theta(i)) ...
    -(sind(phi(i))*sind(psi(i))+cosd(psi(i))*sind(phi(i))*sind(theta(i))) ...
    sind(psi(i))*sind(theta(i))+cosd(psi(i))*sind(theta(i))*cosd(phi(i));
%         sind(psi(i))*cosd(theta(i)) ...
    cosd(psi(i))*cosd(phi(i))+sind(psi(i))*sind(theta(i))*sind(phi(i)) ...
    sind(psi(i))*sind(theta(i))*cosd(phi(i))-cosd(psi(i))*sind(phi(i));
%         -sind(theta(i)) cosd(theta(i))*sind(phi(i)) ...
    cosd(theta(i))*cosd(phi(i))];

rates(i,1) ...
    =L*(deg2rad(q(i))*sind(theta(i))*cosd(psi(i))-r(i)*sind(psi(i)));
rates(i,2) ...
    =L*(deg2rad(r(i))*cosd(psi(i))*cosd(theta(i))-q(i)*sind(psi(i))*sind(theta(i)));
rates(i,3) =-L*(deg2rad(q(i))*cosd(theta(i)));

    D(i,1) = sqrt(1+tand(beta(i))^2+tand(alpha(i))^2);

    bodyU(i,1)=abs(u_a(i))/D(i);
end
```

```
%D = sqrt(1+tan(beta)^2+tan(alpha)^2);

groundMHP = bodyU.*RM_gw';

windGROUND = abs(gDATA) - abs(groundMHP) - rates;

windVELOCITY = windGROUND;
for i=1:length(windGROUND)
    windDIRECTION(i) = atan2d(windVELOCITY(i,1),windVELOCITY(i,2))+90;
end
%windDIRECTION=rad2deg(windDIRECTION);
windSPEED=sqrt(windVELOCITY(:,1).^2+windVELOCITY(:,2).^2+windVELOCITY(:,3).^2);
end
```

VITA

Kyle Hickman

Candidate for the Degree of

Master of Science

Thesis: THIS BLOWS?: EVALUATION OF ADDITIVE MANUFACTURED FIVE HOLE PROBES FOR UNMANNED SYSTEM BASED WIND MEASUREMENTS

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, 2021.

Completed the requirements for the Bachelor of Science in Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in 2019.

Completed the requirements for the Bachelor of Science in Mechanical Engineering at Oklahoma State University, Stillwater, Oklahoma in 2019.

Professional Memberships:

American Model Association, American Institute of Aeronautics and Astronautics, American Physical Society