

UNDERSTANDING AVIAN SOARING TO EXTEND  
UAV MISSION ENDURANCE THROUGH REMOTE  
DETECTION OF THERMAL UPDRAFTS

By

CODY WAYNE PINKERMAN

Bachelor of Science in  
Aerospace & Mechanical Engineering  
Oklahoma State University  
Stillwater, Oklahoma  
2008

Master of Science in  
Mechanical & Aerospace Engineering  
Oklahoma State University  
Stillwater, Oklahoma  
2010

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
DOCTOR OF PHILOSOPHY  
May, 2016

UNDERSTANDING AVIAN SOARING TO EXTEND  
UAV MISSION ENDURANCE THROUGH REMOTE  
DETECTION OF THERMAL UPDRAFTS

Dissertation Approved:

Dr. Andrew S. Arena Jr.

Dissertation Adviser

Dr. Jamey D. Jacob

Dr. Joseph P. Conner

Dr. Timothy J. O'Connell

## ACKNOWLEDGEMENTS

I would like to say a quick thank you to those who are responsible for aiding and encouraging me while on this journey of completing my dissertation.

Thank you to my close friends Ben, Alicia, Brit, and Stacey. You four have been my inner circle. You were there for the best times and through the hardships that occurred during this phase in life. Your encouragement and friendship is truly important to me. I thank God every day for having you in my life. Thank you Nic for being a close colleague and friend during my graduate career. Even after finishing your run, you have inspired me to learn more and to have the determination needed to finish. Thank you to my family for your patience and love through this final degree. Thank you Tanner and Kelly for your friendship these past few years. To the Weaver and Lester families; thank you for your generosity, friendship, and encouragement. It has been a joy to get to know you and to feel welcomed into your homes and families.

Thank you to my church family for your support and prayers. To the men of my Wednesday lunch crew; I thank you for your encouragement and prayers and for keeping me accountable. To the group at Roadhouse, I say thanks for being a consistent part of my life, giving me stability, an avenue to vent, and for something to look forward to on Monday evenings.

Thank you to my advisor and committee for giving me the opportunity to learn and explore through this dissertation process. Thank you for your council and patience through the stumbling points of this research. A special thank you to Dr. Fisher for your enthusiasm and for giving me a wonderful opportunity while at OSU to explore teaching. It was eye opening and enjoyable experience.

I would like to thank Stillwater R/C Flyers and the Soaring League of North Texas for offering flying tips and additional insight into the RC soaring community. Would also like to thank the SLNT for giving me the opportunity to observe the 2015 F3J Soaring team selections to gather additional information RC thermal soaring.

A special thank you to Kay Porter and Nathan Marquez for taking time to help with edits and formatting of this dissertation.

Name: CODY WAYNE PINKERMAN

Date of Degree: May, 2016

Title of Study: UNDERSTANDING AVIAN SOARING TO EXTEND UAV MISSION  
ENDURANCE THROUGH REMOTE DETECTION OF THERMAL  
UPDRAFTS

Major Field: MECHANICAL AND AEROSPACE ENGINEERING:

Abstract: Thermal updrafts can be utilized to increase an aircraft's flight endurance and range. However, much of the discussion involving thermal updrafts comprise of flight techniques within a known thermal updraft and very little is discussed over the detection of thermals without the influence of the upward or downward velocity component. This paper discusses possible thermal locating methods discovered from looking at thermal detection primarily from an avian viewpoint. An avian study was conducted to gather information on how avian soaring species locate thermal updrafts, in which no definitive answer was found. Of the possible theories of avian thermal location methods, combined with methods used from RC aircraft, a single method was chosen for further study through simulation and experimentation to construct possible applications to unmanned aerial systems. By quantifying wind shifts in the local area, the direction and location of local thermal updrafts was theorized to be able to be calculated. However, due to the high magnitudes of uncertainty found during the experimental approach and then portrayed in additional simulations with added uncertainty, this method is shown to be unfavorable for use in remotely detecting thermal updrafts.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.....	1
1.1 GOAL .....	2
1.2 RATIONAL .....	2
1.3 HYPOTHESIS .....	2
1.4 OBJECTIVES .....	4
II. LITERATURE REVIEW.....	6
2.1 THERMAL UPDRAFTS.....	7
2.2 AVIAN STUDIES .....	10
2.3 MANNED FLIGHT.....	13
2.3.1 GLIDERS.....	13
2.3.2 PATENTED TECHNOLOGY.....	15
2.3.3 RADIO CONTROL.....	17
2.4 UNMANNED FLIGHT .....	20
III. SIMULATIONS .....	23
3.1 SINK PROFILE .....	24
3.2 EXPERIMENTAL SIMULATION .....	33
3.2.1 THIRD VECTOR APPROACH.....	35
3.2.2 THIRD VECTOR SIMULATION .....	37
3.2.3 AVERAGING METHODS .....	38
IV. EXPERIMENTAL SETUP .....	46
4.1 SENSOR DESIGN.....	47
4.1.1 MICROCONTROLLER.....	47
4.1.2 WIRELESS TRANSMISSION .....	48
4.1.3 WIND SPEED .....	52

Chapter	Page
4.1.4 WIND DIRECTION .....	56
4.1.5 TEMPERATURE & HUMIDITY .....	58
4.1.6 BAROMETRIC PRESSURE.....	62
4.1.7 ARRANGEMENT .....	65
4.1.8 COST .....	67
4.2 TESTING .....	69
4.3 SENSOR VISUALIZATIONS .....	78
4.3.1 MATLAB SERIAL.....	79
4.3.2 COOLTERM.....	80
4.3.3 PROCESSING .....	82
4.4 GUI DESIGN.....	86
4.4.1 READ XBEE DATA .....	87
4.4.2 STORE XBEE DATA .....	91
4.4.3 VISUALIZE XBEE DATA .....	93
V. RESULTS .....	99
5.1 EXPERIMENTAL APPROACH.....	100
5.2 BURN RESULTS .....	101
5.3 FLIGHT DATA .....	115
5.3.1 JUNE 1 <sup>ST</sup> , 2015 DATA .....	115
5.3.2 SEPTEMBER 10 <sup>TH</sup> , 2015 DATA.....	124
5.3.3 ADDITIONAL DATA.....	132
5.3 UNCERTAINTY .....	133
5.3.1 UNCERTAINTY EQUATIONS .....	134
5.3.2 CALCULATED UNCERTAINTY .....	137
5.3.3 UNCERTAINTY WITHIN DATA .....	139
5.3.4 STATISTICAL WIND DATA .....	141
5.3.5 ADDITIONAL SIMULATIONS.....	147
5.3.6 COMBINED UNCERTAINTY ANALYSIS .....	162
V. CONCLUSION & RECOMMENDATIONS .....	166
6.1 CONCLUSIONS.....	166
6.2 RECOMMENDATIONS .....	168

Chapter	Page
6.2.1 ADDITIONAL EXPERIMENTATION .....	169
6.2.2 NEW EXPERIMENTATION.....	169
6.2.3 SENSOR UPGRADES .....	171
REFERENCES .....	172
APPENDICES .....	175
ARDUINO CODE .....	176
GRAPHICAL USER INTERFACE .....	180

## LIST OF TABLES

Table	Page
Table 4.1: Individual sensor cost .....	68
Table 4.2: Computer hub cost.....	68
Table 4.3: Total Cost for sensor package.....	69
Table 5.1: Wind averages from different sources .....	107
Table 5.2: Component accuracy .....	133
Table 5.3: Uncertainty ranges for June 1st, 2015 .....	163
Table 5.4: Range of uncertainty for September 10th, 2015 .....	164

## LIST OF FIGURES

Figure	Page
Figure 2.1: Column thermal (left), bubble thermal (right).....	8
Figure 2.2: Thermal updraft Gaussian distribution.....	9
Figure 2.3: Thermal detection concept from found patents.....	16
Figure 2.4: Thermal updraft with ground inflow vertical cross section (left), horizontal cross section (right).....	18
Figure 2.5: Third vector.....	19
Figure 2.6: Uniform and sink profile.....	19
Figure 3.1: Sink flow.....	24
Figure 3.2: Stream lines of uniform with sink flow.....	27
Figure 3.3: Simulation of test area with sink flow only.....	28
Figure 3.4: Simulation of test area with uniform and sink flow.....	28
Figure 3.5: Thermal column cylindrical control volume.....	31
Figure 3.6: Sink strength based on thermal factors.....	32
Figure 3.7: Simulation of experimental setup.....	34
Figure 3.8: Uniform plus sink profile.....	35
Figure 3.9: Vector approach of for sensors.....	35
Figure 3.10: Simulation vector approach.....	38



Figure	Page
Figure 4.15: RHT03 switch to read values .....	61
Figure 4.16: BMP180 barometric pressure sensor .....	62
Figure 4.17: BMP180 setup .....	63
Figure 4.18: Obtaining pressure .....	64
Figure 4.19: BMP180 error code .....	64
Figure 4.20: Storing, converting, and displaying pressure .....	65
Figure 4.21: Sensor wire diagram .....	66
Figure 4.22: Sensor layout .....	67
Figure 4.23: Simple shading (left); Constructed solar radiation shielding (right) .....	71
Figure 4.24: Sensor temperature comparisons .....	72
Figure 4.25: Initial wind speed values .....	73
Figure 4.26: Wind tunnel testing .....	74
Figure 4.27: Anemometer initial testing results .....	75
Figure 4.28: Initial anemometer oscilloscope readings .....	76
Figure 4.29: Oscilloscope readings with capacitor .....	77
Figure 4.30: Anemometer results with added capacitor .....	78
Figure 4.31: Arduino monitor hexadecimal display .....	79
Figure 4.32: CoolTerm display .....	81
Figure 4.33: Example of CoolTerm output using temperature sensor .....	82
Figure 4.34: Sample Processing sketch for testing .....	85
Figure 4.35: Initial GUI concept for ground sensor interface .....	86
Figure 4.36: GUI XBee library import and initialization .....	87

Figure	Page
Figure 4.37: GUI check for incoming data.....	88
Figure 4.38: GUI code to read sensor address.....	89
Figure 4.39: GUI code to read and convert packet (only temperature shown).....	90
Figure 4.40: GUI error exceptions.....	91
Figure 4.41: GUI code to initialize saving of data.....	92
Figure 4.42: GUI code of saving data within columns.....	93
Figure 4.43: Primary background image (1000x800 pixels).....	94
Figure 4.44: GUI overlay.....	95
Figure 4.45: Code for text.....	96
Figure 4.46: Code for wind vector.....	96
Figure 4.47: Code used to draw thermal indicators.....	97
Figure 4.48: Final version of GUI at the time of this document.....	98
Figure 5.1: April 20th control burn location and approximate sensor placement.....	101
Figure 5.2: Mesonet station, Marena, near controlled burn.....	103
Figure 5.3: Directional rose plots for April 20th burn.....	104
Figure 5.4: Wind rose plots for controlled burn.....	105
Figure 5.5: Wind direction for controlled burn.....	106
Figure 5.6: Directional rose plot with average wind direction (red) (Top), Speed weighted rose plot with weighted average (red) (Bottom).....	108
Figure 5.7: Pressure data for controlled burn.....	109
Figure 5.8: Temperature data for controlled burn.....	110
Figure 5.9: Relative humidity data for controlled burn.....	111

Figure	Page
Figure 5.10: Alpha wind shifts sample rate comparisons.....	113
Figure 5.11: Bravo shift comparisons.....	114
Figure 5.12: OSU Unmanned Aircraft Flight Station with sensor locations.....	116
Figure 5.13: Wind rose of each sensor package during data collection.....	117
Figure 5.14: Rose plots of wind direction (blue) and average direction (red) during data collection.....	118
Figure 5.15: Pressure data for June 1st, 2015.....	119
Figure 5.16: Temperature results during June 1st, 2015.....	120
Figure 5.17: Relative humidity levels during June 1st, 2015.....	120
Figure 5.18: Noticed temperature drops for June 1st, 2015.....	121
Figure 5.19: Noticed humidity spikes for June 1st, 2015.....	122
Figure 5.20: Thermal wind rose for June 1st, 2015 (11:00 - 11:15 C.S.T.).....	123
Figure 5.21: June 1st, 2015 thermal event.....	124
Figure 5.22: Pressure data for September 10th, 2015.....	125
Figure 5.23: Temperature data for September 10th, 2015.....	126
Figure 5.24: Humidity data for September 10th, 2015.....	127
Figure 5.25: Compass plots for September 10th, 2015.....	128
Figure 5.26: Wind rose plots for September 10th, 2015.....	129
Figure 5.27: September 10th thermal location found at 15:55 C.S.T.....	130
Figure 5.28: September 10th thermal pull.....	131
Figure 5.29: September 10th thermal indications.....	132

Figure	Page
Figure 5.30: Uncertainty levels across a simulated test case using an average vector of 5 mph at 30° .....	137
Figure 5.31: Uncertainty with differences between the instantaneous and average vectors .....	138
Figure 5.32: Sept 10th uncertainty using raw data .....	139
Figure 5.33: Sept 10th uncertainty using raw data close-up view .....	140
Figure 5.34: June 1st uncertainty using raw data, close-up view .....	141
Figure 5.35: Sept 10th standard deviation in measured speed .....	143
Figure 5.36: Sept 10th standard deviation in measured angle .....	144
Figure 5.37: Standard deviation of wind speed during thermal event .....	145
Figure 5.38: deviation of wind direction during thermal event .....	146
Figure 5.39: Sept 10th difference in standard deviation of single sensor (Charlie) using different average lengths .....	147
Figure 5.40: Vector components of simulation .....	148
Figure 5.41: Code of uncertainty being added in simulation .....	149
Figure 5.42: Simulation with added uncertainty .....	150
Figure 5.43: Simulation with added uncertainty with triangulation of thermal event .....	151
Figure 5.44: Histogram of initial detections vs sink strength .....	152
Figure 5.45: Optimal simulation setup .....	153
Figure 5.46: Detection vs sink strength with 5 mph uniform wind and sensor uncertainty values .....	154

Figure	Page
Figure 5.47: Detection vs sink strength with 2 mph uniform wind and sensor uncertainty values .....	155
Figure 5.48: Detection vs sink strength with 10 mph uniform wind and sensor uncertainty values .....	156
Figure 5.49: Detection vs sink strength with 5 mph uniform wind and gust uncertainty values .....	157
Figure 5.50: Detection vs sink strength comparisons of 5 mph uniform wind with sensor uncertainty (top) and gust uncertainty (bottom) .....	158
Figure 5.51: Simulation setup with thermal components .....	159
Figure 5.52: Detection vs $r/R_t$ ratio for multiple thermal sizes .....	160
Figure 5.53: Indicators due to uncertainty with no thermal component .....	161
Figure 5.54: Total uncertainty for June 1st, 2015 thermal event .....	162
Figure 5.55: June 1st, 2015 thermal event uncertainty levels .....	163
Figure 5.56: September 10th, 2015 uncertainty values .....	164
Figure 5.57: Uncertainty around September 10th, 2015 thermal event .....	165

## CHAPTER I

### 1. INTRODUCTION

A pressing matter with new aircraft, including unmanned aerial vehicles (UAVs), is increasing overall flight endurance. Many techniques for increasing endurance have focused on increasing engine performance and fuel consumption, decreasing drag profiles, or maximizing payload to allow for excess fuel capacity. Another technique to increase flight endurance is by decreasing engine requirements through the use of naturally occurring convection updrafts, more commonly known as thermal updrafts or thermals. An issue affecting the use of thermals is the ability to find them. To better determine a method of locating thermals, an avian viewpoint was explored to better understand the method birds use to find thermals. However, a definitive answer to how birds detect thermals has not been found. Through piecing together thoughts from the aerospace and avian literature, different hypotheses were produced on the nature of thermal discovery. One method was then selected for further studied to determine viability for use in UAV flight.

## 1.1 GOAL

The purpose of this research endeavor was to determine the method used by avian species to detect and find thermal updrafts and to attempt to adapt this method for use with unmanned aerial vehicles. The focus was on how these species find the updrafts, not on how these updrafts are utilized or optimized, as a tremendous amount of research already exists on this topic. In the case that multiple methods could be utilized, one method of thermal detection would be experimentally tested.

GOAL: Determine validity of a single, useful method of identifying thermal updrafts, based on avian soaring techniques, for specific implementation on UAVs in order to increase the total flight endurance and overall efficiency.

## 1.2 RATIONAL

The project's goal was initially set with specific reasoning behind it. Thermal updrafts can vastly increase an aircraft's flight endurance. However, a reliable method for detecting thermal updrafts has yet to be determined. Using a bird's natural ability to find these updrafts as a starting point, a detection method could be discovered. Yet, even in the avian literature, it is unclear as to the actual method birds use to local updrafts.

RATIONALE: Thermal updrafts can vastly increase an aircraft's flight endurance.

Understanding and applying a bird's natural ability to find these updrafts could lead to a viable detection method for aircraft, both manned and unmanned, to locate local updrafts and increase mission endurance.

## 1.3 HYPOTHESIS

To begin working towards the proposed goal, information was initially examined on current theories and thoughts on avian detection of thermal updrafts. No definitive answer to how birds

detect thermals was found. However, from the information found from these avian studies, which will be discussed in detail in Chapter 2, multiple hypothesis were discovered and examined.

#### HYPOTHESES:

- Memory
  - Recalling where updrafts have previously been
- Feeling
  - Feeling shifts in local air properties to give indication of thermal updraft locations
- Sight
  - Seeing objects within the thermal updraft
- Search
  - Optimizing flight pattern to find areas of lift
- Extrasensory
  - Avian species may have a yet unknown method of seeing, feeling, detecting updrafts that has yet been discovered

These five hypotheses were found from multiple sources with regards to just avian species. Additional information was found from aerospace studies, both manned and unmanned flight, in order to further focus the information towards a single method as stipulated in the goal. By using additional information from non-avian sources, the prospect of a single hypothesis having more viability over another could be inferred. From information found from avian and aerospace studies, which will be discussed in detail in Chapter 2, detection of thermal updrafts, theoretically, could be possible through detecting changes in the surrounding air properties, primarily through shifts in air direction and speed. This comes from the avian theory of feeling the changes while in flight as well as support from anecdotal evidence from manned flight. Detecting shifts in the surrounding air, a bird or an aircraft could be capable of determining the direction of a nearby thermal updraft. Changes in fluid properties was one of the five possible ways of detecting thermal updrafts inferred from avian studies as well as a known method from aerospace studies.

#### HYPOTHESIS TO TEST:

- Feeling
  - Feeling shifts in local air properties to give indication of thermal updraft locations
- Additional anecdotal evidence found from aerospace studies
  - Manned and unmanned flight

#### RATIONALE FOR SINGLE TEST:

- Aerospace studies evidence
- Anecdotal evidence from
  - Glider studies
  - Radio controlled aircraft studies, interviews, and personal flight experience

### 1.4 OBJECTIVES

The goal of the research is to test the feasibility of using surrounding air properties to identify nearby thermal updrafts. The goal was approached through a series of objectives involving computer simulation and experimental data collection to determine the viability to locate local thermals for UAVs using measurable changes in surrounding air properties.

To accomplish the goal of this dissertation, a series of objectives were generated. These objectives were outlined to be contingent on the information of the prior objective and to give a process for which the goal could be accomplished. The first phase would be to gather any information regarding thermal updraft detection. Based on the found information, simulations would be conducted in order to glean initial feasibility and obtain the necessary calculations for the method in question. Once a plausible method and necessary approach was found, data collection and experimentation could begin in an area with known thermal activity. With data collection, it would be beneficial to collect and analyze data at a ground level, and then later collect data while in flight to

allow for comparison and evaluation of the collected data. Based on the chosen method of detection, via shifts in the local wind vector, the initial objectives for the goal were generated as follows:

OBJECTIVE 1: Conduct literature review on avian and manned thermal soaring with emphasis on thermal updraft identification.

OBJECTIVE 2: Develop computer simulations to show the possible effects thermal updrafts have on surrounding air masses.

OBJECTIVE 3: Design and test ground based sensor network for ground based experiments at OSU UAV Flight Field.

OBJECTIVE 4: Simulate experimental approach to determine method in order to subtract dominant/uniform wind vector to determine thermal updraft component.

OBJECTIVE 5: Implement UAV with recording apparatus to experimentally obtain air data during thermal soaring in combination with ground sensor network to confirm existence of thermal updrafts.

OBJECTIVE 6: Use combination of simulated and experimental data to confirm or deny the feasibility to use wind shifts to detect nearby thermal updrafts.

The subsequent chapters will go into greater detail on the procedures and completion of each of the objectives.

## CHAPTER II

### 2 LITERATURE REVIEW

Information regarding avian thermal detection was the focus of the information sought after during the initial phases of this research endeavor. This information was gathered from avian literature and scholarly sources, aerospace sources, as well as interviews with avian experts: individuals who have more scholarly knowledge of avian anatomy and social behavior than the engineers conducting this study.

Along with avian thermal detection examined, information from manned flight was also examined in order to determine additional types of techniques that have been or could be utilized. Thermal use from manned flight was found through glider and radio controlled (RC) flight experiences. Previous research involving UAVs and thermals with regards to detection and utilization as well as techniques to increase mission endurance was also conducted. Most of the reviewed literature however contained little to no information regarding the detection of thermals, but more on the application, simulation, and coding algorithms during flight within a thermal updraft. This information obtained from manned and unmanned flight would help to narrow down the multiple hypotheses found in avian studies in order to inspect a single hypothesis.

The background information gathered was used to complete the first objective outlined in Chapter 1. The following sections go into details of the important information, or lack thereof, found within multiple sources pertaining to thermal updraft detection

## 2.1 THERMAL UPDRAFTS

Before gleaming information from the avian and human flight experiences, definitions and knowledge of the structure of thermals should be defined. Thermals are defined as large columns of rising air in the convective mixed layer<sup>1</sup>. Thermals are buoyant air masses that ascend due to warmer ground temperatures heating the surrounding air making it less dense<sup>2,3</sup>. Thermals are formed from ground transferring heat by convection to the air mass above its surface. As ground conditions become warmer, the air above becomes less dense and begins to rise. These conditions promote the formation of a thermal updraft. This updraft will continue to form until the ground air begins to cool. These thermals can have vertical air velocities ranging from weaker 1 to 2 m/s updrafts to typical 5 to 10 m/s; velocities of 20 m/s have been recorded<sup>2,4</sup>. The vertical velocity component or lift region of the thermal can range from 200 to 600 m in diameter at altitudes of 1,000 to 2,000 ft and can become even wider at higher altitudes<sup>8</sup>.

Once a thermal is created, it can become one of two main types: column and bubble<sup>2,5,6</sup>. A column thermal is as the name suggests; a constant column of rising air that is being fed by the heat from the ground source. Column thermals tend to form in regions with greater temperature differences. Under the right soil and ground condition and with hot weather conditions, the ground stays warmer longer, thus continuously “feeding” the thermal with less dense, buoyant air (Figure 2.1 left).

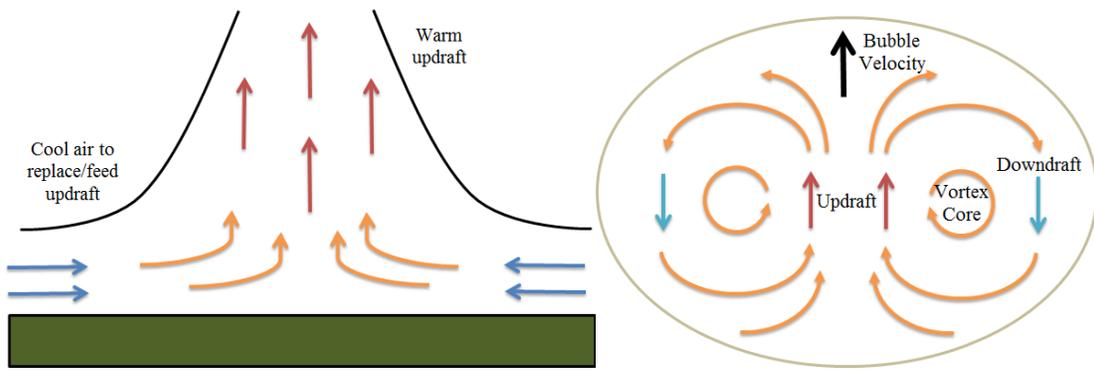


Figure 2.1: Column thermal (left), bubble thermal (right)

Bubble thermals are formed when the created thermal breaks away from the ground source. This can be either due to ground cooling, obstructions, or from wind effects which can cause a quick burst of cooler air to replace a portion of warm air thus pinching off the column<sup>4,7</sup>. The thermal breaks away from the surface to form a bubble or vortex ring. The vortex ring consists of a core with upward velocity surrounded by a shell of downward velocity air (Figure 2.1 right). The vertical velocity of these thermal bubbles is strongest in the center of the core. The velocity within the core starts at a minimum at the bottom and increases till it reaches the center and will begin to decrease in magnitude until it begins to develop into the down draft of the outer ring<sup>7</sup>. As the bubble thermal rises from the ground in the atmospheric boundary layer, the size of the thermal increase<sup>2,7,8</sup>.

It should also be said that the location of the thermal is subject to the surrounding wind conditions. As both a column and bubble thermal rises, the path can and will shift with the direction of the prevailing winds. This means that the thermal will be moving parallel to the surface as well as vertically. In some cases, if multiple thermals are present, thermal streets can be created<sup>9,10</sup>. Thermal streets occur when the prevailing winds will elongate thermals rows or streets which are evident by gathered cumulus clouds.

Most thermal models have been based on column thermals. Part of this seems to be for simplicity in modeling only the upward velocity components of the thermal. Most of the modeling found within engineering research uses different types of Gaussian distributions or “top hat” distributions to model the upward velocity in the thermal where the strength of the velocity updraft is based on the location within the radius of the thermal (Eq. 2.1)<sup>11-15</sup>.

$$V_t(r) = V_{\max} \left[ 1 - \left( \frac{r}{R_t} \right)^2 \right] e^{-\left( \frac{r}{R_t} \right)^2} \quad \text{Eq. 2.1}$$

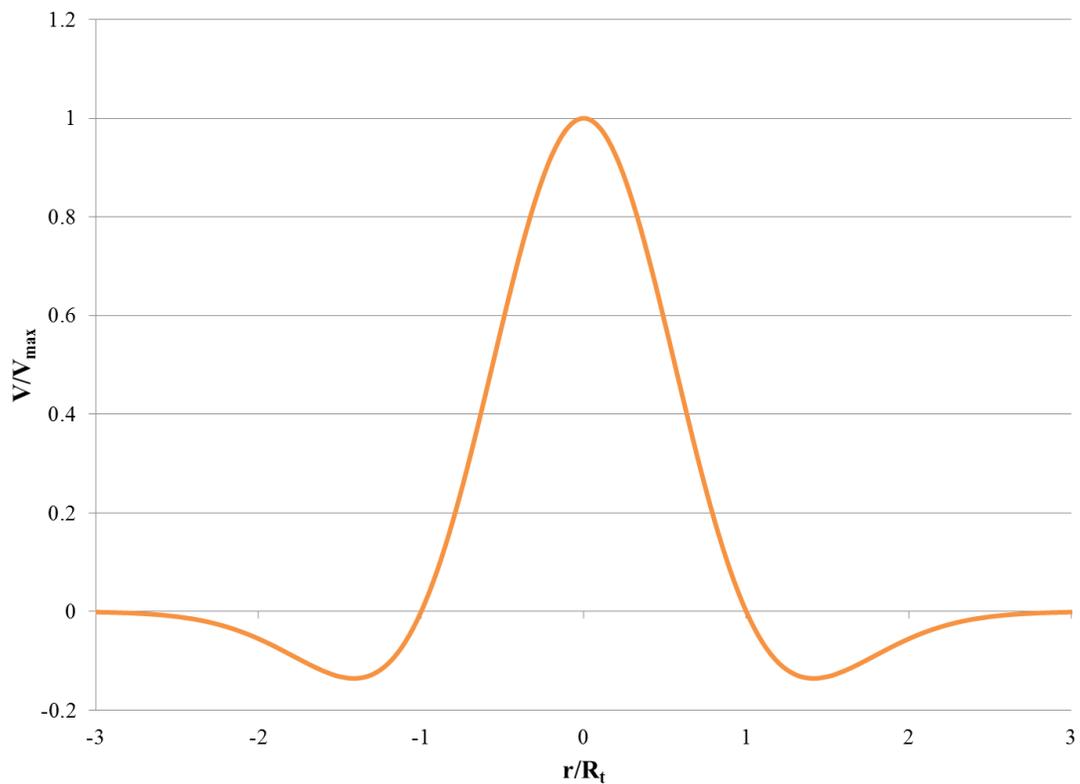


Figure 2.2: Thermal updraft Gaussian distribution

Where in Eq. 2.1  $V_{max}$  is the maximum thermal updraft velocity,  $r$  is the radius from the center of the thermal, while  $R_t$  is the radius of the thermal updraft. Figure 2.2 shows the modeled updraft distribution using Eq. 2.1. In some models, only the positive vertical velocity was modeled. Downdrafts, lateral velocity, and vortex effects were not modeled. Most of these models were used to help in the development of optimal circling radius within the thermal to optimize the altitude and energy gain.

## 2.2 AVIAN STUDIES

An intense literature review on avian use and detection of thermal updrafts was undertaken. The purpose of the literature review was to attempt and discover the method employed by avian soaring species to detect thermal updrafts. From much of the literature on thermal soaring with birds, the focus has been more on the strategies of using thermals for flight path optimization, the effects on social structure, and comparisons and observations with bird-thermal interactions<sup>2, 15-20</sup>. Few addressed the technique of a bird's natural ability to detect these thermal updrafts. A prevailing theme in this literature is that birds are able to reliably detect thermals but that the specific mechanism of detection is unknown. For example, van Loon stated; "little is known about distances at which birds can sense thermals yet we expect thermals with birds in them to be detectable at farther distances than thermals without birds"<sup>18</sup>. However, from avian studies, a few hypotheses about how soaring birds find thermals were formed.

It has been seen that many birds will find thermals based on their ability to observe other birds or even gliders that are soaring in a particular thermal<sup>2, 4, 17, 18</sup>. This gives rise to one hypothesis for a bird's ability to find a thermal is by sight. Glider pilots will find thermal using the visual cues of spotting birds in thermals. This phenomenon has also been shown in reverse, in which birds will head towards gliders using thermals<sup>2</sup>. There have been many cases in which glider pilots will begin to fly within a thermal and notice birds changing directions to join them<sup>19</sup>. Bird observation studies have also shown that birds will get cues of thermals from other birds. One such observation detailed the event of a flock of European Starlings (*Sturnus vulgaris*) using

an Osprey (*Pandion haliaetus*, a large bird of prey) as a reference for flying in a thermal<sup>17</sup>. Cone observed the Starlings thermalling over the Osprey and matching the Osprey's flying for their own benefit. The starlings had not been sighted thermal soaring solo before the presence of the Osprey. Many glider reference books illustrate observations of soaring birds traveling miles to join gliders and/or other soaring birds that are in strong thermal updrafts<sup>2</sup>. The visual cue of observing other birds and gliders (that are likely perceived as large soaring birds), plays an important role in the ability of birds to find thermals, yet this does not account for the first bird's ability to find the thermal.

Much of the research addressing the use of thermals by soaring birds was the application of the thermals. Van Loon simulated the social interaction of migrating soaring birds in using thermal updrafts<sup>18</sup>. Migrating flocks of White Storks (*Ciconia ciconia*) were modeled to simulate migration patterns to better understand flight behavior and thermal selection. In this simulation, the birds were able to detect thermals in two specific ways at two different radii. Sight of another bird using a thermal had the largest radius of detection for the simulated bird (indirect detection). The second method in finding a thermal was that the bird was simply able to see or detect the thermal itself (direct detection). No actual information was given except that the bird could detect a thermal within a much smaller radius as that compared to visually spotting another bird in a current thermal. This was based on other studies that observed storks regularly flying to thermals that were located kilometers away, and flying to farther thermals using visual cues such as other birds<sup>2, 21</sup>.

Qualitative analysis from tracked birds has shown the presence and magnitudes of thermals in specific regions<sup>15</sup>. In some cases, soaring birds were outfitted with GPS tracking equipment and a positive increase of altitude, especially in spiral patterns, indicated thermal updrafts. Other studies have used powered and unpowered aircraft to observe soaring birds using thermal updrafts<sup>19</sup>. One such study examined the maximum flight altitudes of four species of birds from radar samples and related this information to predicted thermal depth and intensity of the area

which was then compared to simulated thermals and estimated altitude gains for each of the species. Once again, the focus of this information was used in regard to migration patterns and efficiency, and not the methods exploited to find updrafts.

Much of the literature conducted in the research has shown that birds detect thermals visually through seeing objects within the thermal. Another possibility is that birds rely on memory to find thermals<sup>20</sup>. Birds occupying territories over months and years develop intimate knowledge of the landscapes they inhabit, including propensity for thermal formation over variable land cover. Memory could also account for migratory birds using thermal updrafts. Migratory birds would know the flight path and know where thermals have been found in migrations past, and this knowledge could be passed down from generation to generation. A bird could detect thermal updrafts through ultimate or proximate behavior. The ultimate behavior would account for a bird predicting thermal formations through instinct or experience, while proximate behavior would account for detection of existing thermals.

Other studies have suggested that birds could be capable of “feeling” thermals; that is, the bird can feel changes in local air properties (pressure, wind direction, and/or temperature) and deduce a direction for a thermal<sup>12, 18, 21, 22</sup>. A bird’s inner ear is highly sensitive to changes in pressure, much more so than that of humans<sup>20</sup>. This gives birds an advantage to possibly feel pressure drops in the air during flight. As well as being sensitive to pressure changes, birds are more sensitive to changes in wind velocity and sudden changes in lift by feeling these changes along their wings. In a study by researchers at Kansas State University, the mechanoreceptors on a feather follicle on a bird’s was theorized to help a bird determine airflow along it’s wings<sup>23</sup>. Discharge frequency of these receptors were analyzed and showed an increase in discharge frequency with change in airflow velocity. This study concluded that the receptors in a bird’s wing are capable of detecting stall and separation point on the wing, as well as the capability to measure airspeed due to the secondary flight feathers.

If this is the case, then perhaps thermals can be detected from much larger range from knowing local atmospheric conditions and not just detected by sight or memory. For example, soaring birds will often wait for a specific time of day to seek out thermals<sup>12</sup>. Raptor migratory flights or foraging bouts on breeding territories usually begin in late morning when the ground has been warmed and wind velocity is generally greater than that of early morning. In addition to the noticeable increase in temperature that might alert birds to conditions favorable for thermal formation, soaring birds might be able to detect and follow air currents that move toward rising columns of air, thereby hitting a thermal in the absence of strictly visual cues or the reliance on memory.

Even with the ideas of birds using sight, memory, pressure, and wind shifts there is also the possibility of extrasensory technique used by birds to detect thermals<sup>20</sup>. There could be some yet unknown construct that is used by soaring birds that is neither sight, nor memory, nor sensing of atmospheric conditions. Or even perhaps a bird must actively search for the updrafts. Perhaps a soaring bird only knows of thermal updrafts once it has felt the vertical component. In either case, an overall consensus of a bird's ability to detect and find thermal updrafts was not discovered during the avian literature search during this research endeavor.

## 2.3 MANNED FLIGHT

Resources in manned aviation were also examined regarding previous and current methods of determining thermal updrafts. Thermal soaring with regards to full scale glider, small scale radio controlled aircraft, as well as current methods utilized or theorized by unmanned aircraft were examined.

### 2.3.1 GLIDERS

The sport of unpowered gliding is relatively young, since thermal soaring was not examined until about the 1930s<sup>5,6</sup>. Since then, gliders have been able to use thermals to get incredible energy benefits. Some gliders have sustained flights lasting longer than 9 hours and have traveled over 1,200 miles without stopping<sup>14</sup>. Gliders have their own methods of finding

thermals. Glider pilots use visual cues (e.g. birds, other gliders, rising dust and dirt particles, etc.) to determine the most probable locations for thermal updrafts. Visual cues can be found at heights of 2,000 and 3,000 ft above the surface<sup>2,5</sup>. Even dust and debris that is caught in an updraft can be indications of thermals. Not only will glider pilots notice other objects using thermals, but the pilot can also notice changes in the ground wind direction. When a thermal is forming or still being fed by the warm ground temperature, air will travel toward the thermal to replace the air that is being caught in the updraft. Thus, by placing flags or wind vanes, pilots can notice shifts in ground wind direction and determine possible thermal locations.

Another visual cue glider pilots use is the formation of cumulus clouds. Cumulus clouds will form due to the thermal feeding moist, warm air to the cloud layer. Sometimes these can be seen in the form of a milky patch of haze, which is caused by a thermal not containing enough water vapor to create a full cumulus cloud. Polarized sunglasses can often help spot these types of hazed areas. When watching the clouds, pilots have to be aware of the difference between a cloud that is forming and one that is in a state of decay. When a cloud is forming, the cloud is usually being fed moisture from a thermal updraft. Thus a forming cloud would be a good indication of a thermal location. However, if the cloud is decaying, then there will be strong downdrafts because of the cooling air. Decaying clouds can be identified by noticing the structure of the cloud base. If the base is jagged, then it is in a state of decay<sup>2,6</sup>. Glider pilots will also be on the lookout for cloud streets. Cloud streets are rows of cumulus clouds that are arranged in fairly straight lines along the direction of the wind and are usually spaced out about twice the cloud base height<sup>5</sup>.

Most gliders are also equipped with a variometer. A variometer is an instrument in the glider that is used to measure changes in vertical speed<sup>5,6</sup>. By noticing a change in vertical speed, the pilot can determine if he or she is in a state of sink or upward movement. If the glider is in an upward movement, then it is possibly within a thermal updraft. The variometer is useful to not only determine the location of the thermal, but also the strength and size of the thermal which can

be used to determine the best approach. When flying in or through bubble thermals however, the variometer can give false readings showing stronger thermals away from the core. This can occur when flying around the cap and base regions where the updrafts are lower in magnitude and changing direction.

### 2.3.2 PATENTED TECHNOLOGY

While examining manned soaring methods, multiple patents, both domestic and foreign, were found claiming to aid in the detection of thermal updrafts and to help pilots when soaring. These patents were searched based on claims to detect thermal updrafts.

First mention of a “thermal finder” was found as early as 1955, where a recent method for detecting local updrafts was being mentioned as new and upcoming instrumentation that may aid in soaring flight<sup>24</sup>. This “thermal finder” was described as having 2 temperature sensors, being placed on the tips of each wing, where the change of temperature could be quantified across the wingspan. The idea was to detect possible thermal updrafts by turning in the direction of higher temperature as the updraft would be comprised of warmer air. Multiple patents were found based on this idea, where 2 or more temperature sensors would be placed along the wingspan and/or sailplane, to give indication of nearby thermals by following the areas of warmer air flow<sup>25-27</sup>. One patent increased the number of sensors to not only analyze temperature on the tips of the wings, but also by adding rear and/or forward sensors to give more direction and identification of thermals<sup>27</sup>.

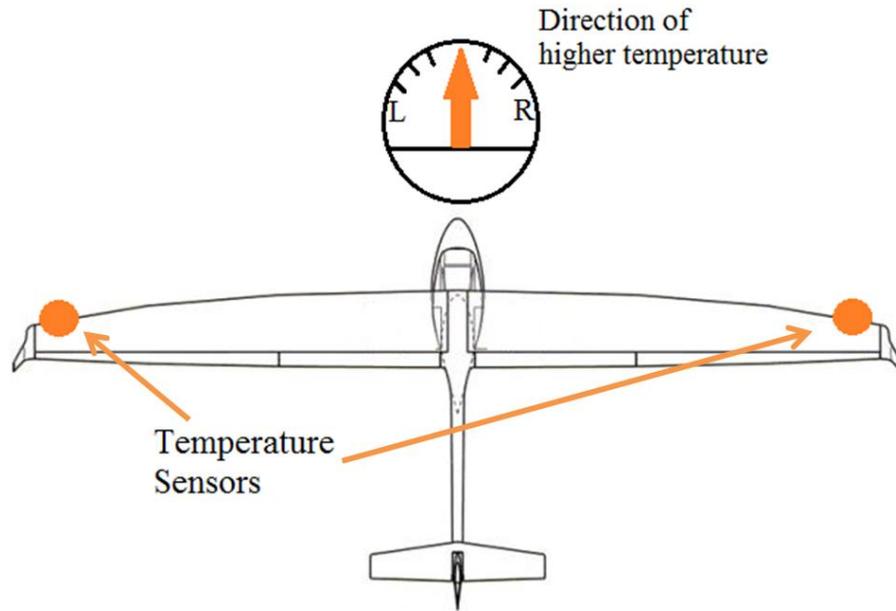


Figure 2.3: Thermal detection concept from found patents

Based on the claims of the patents, and from other patents that reference them, it seems that these methods for following areas of higher temperature only work in close proximity to the thermal and would be used primarily to help determine direction of turning and centering. When used in conjunction with a variometer, for example, if indications show a signs of downdraft or updraft, knowing the temperature difference would give indication of whether to turn left or right (depending on direction of highest temperature) which would allow the sailplane to enter and center once within the updraft.

Other patents were found in which either humidity sensors<sup>25, 28</sup> or structural sensors on the wings were used to detect changes in humidity or the bending moments, respectively, on the wings<sup>29</sup>. The premise of these patents were similar to idea with change in temperature, but would use areas of higher humidity or highest wind bending to determine the thermal direction. Thermals are comprised of moist rising air, which would have a higher humidity level than

surrounding air along with cause a bending moment on a glider's wing which was within the updraft. Again, these ideas would be viable in close proximity to an updraft.

### 2.3.3 RADIO CONTROL

While looking into manned thermal flight, background information was also conducted on radio controlled (RC) thermal flight. RC thermal soaring differs from glider soaring, as the pilot is now an observer on the ground. Yet, these RC pilots are still able to achieve increased flight times by locating and utilizing local thermal updrafts. Multiple techniques were found to be utilized by RC pilots in order to find areas of thermal activity. Information was found through literature review as well as through onsite interviews. These interviews were conducted of multiple RC thermal soaring pilots at both local and national level soaring events. From these interviews, additional insight was gathered on techniques used by RC pilots.

Some of these techniques used by RC pilots mirror those used by full scale pilots: visual cues with birds or objects within the thermal, variometer instrumentation, and ground flags. A common practice amongst RC pilots use is noticing changes in the local wind speed and direction. RC pilots will either feel this change on their person, or notice these changes in their surroundings by utilizing flags, streamer, trees, tall grass, bushes, and bubbles<sup>30-32</sup>. This method comes from the knowledge that thermal updrafts pull in warm air from the ground level to feed into the vertical component (Figure 2.1). This inflow aspect is similar to that of a sink profile.

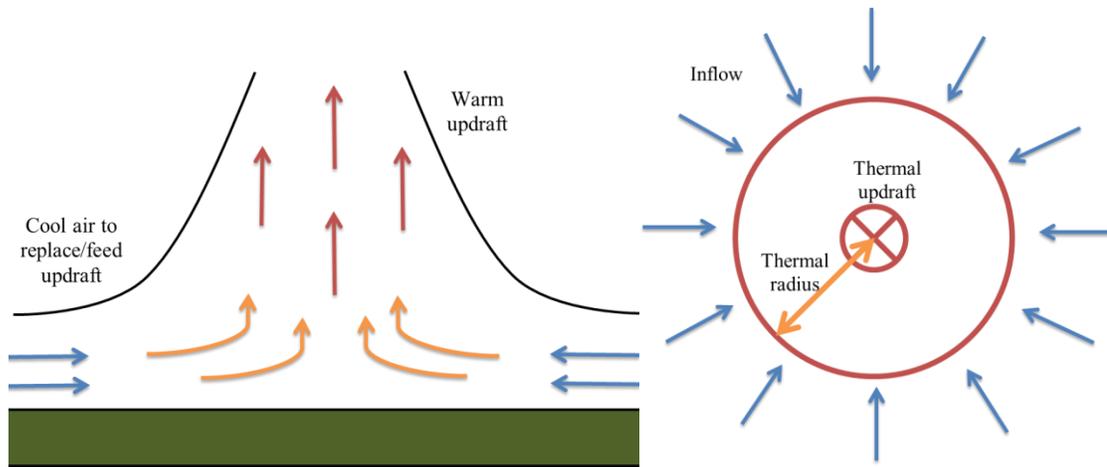


Figure 2.4: Thermal updraft with ground inflow vertical cross section (left), horizontal cross section (right)

The horizontal inflow from local thermals will affect the local wind conditions around an RC pilot. By noticing these changes, and mentally determining the direction of the inflow, RC pilots can determine areas of thermal activity. RC thermal soaring pilot and soaring world champion Joe Wurts illustrated finding thermal updrafts by mentally determining the wind's "third vector"<sup>32</sup>. The idea of the "third vector" is that the felt wind vector is a product of the combination of the uniform wind along with the inflow caused by the thermal updraft (Figure 2.5), which is similar to that of combining uniform flow field and a sink profile (Figure 2.6).

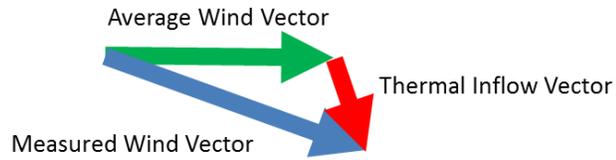


Figure 2.5: Third vector

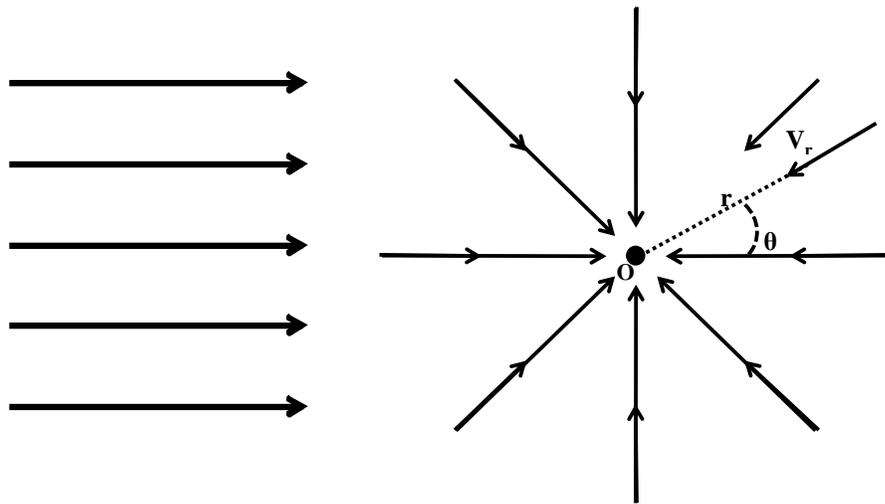


Figure 2.6: Uniform and sink profile

This summation that results in the “third vector” is similar to the idea of determining shifts in the local wind. These shifts are caused by the thermal vector and account for the major changes in the felt uniform wind component.

An additional method utilized by RC is watching birds: either large soaring birds or small finches<sup>30,31</sup>. By watching for birds utilizing thermals, RC pilots will know where updrafts are located and where to focus flight patterns. Large soaring species can be found utilizing formed updrafts. Smaller birds, such as finches, can be seen feeding on small insects that are caught in

the updraft. By watching for either of these types of birds, RC pilots can find updrafts to extend flight endurance.

## 2.4 UNMANNED FLIGHT

Previous research involving UAV mission endurance through detection and use of thermals was also conducted. Most of the published literature addresses application, simulation, and coding algorithms during flight within a thermal updraft. There was little to no information regarding thermal detection without being in the vertical updraft component.

Many studies used multiple UAVs to search for thermals and share thermal information with other UAVs<sup>11,33</sup>. With regard to a single UAV being able to detect thermal updrafts, most research utilized an exploration process or with known information about thermal updraft location<sup>13,22,34,35</sup>. For the exploration process, a UAV would be programmed to use thermals that were found along a given flight plan. One such study used energy calculations for optimal flight trajectories of known wind fields in order to optimize flight performance<sup>34</sup>. In this study, the flight was initially scanned by the simulated glider to determine current wind fields in the flight area. These wind values were then used to optimize a flight pattern within the flight area to optimize endurance. Other examples of exploration used a randomized flight pattern to search for thermals<sup>11</sup>. Other studies focused on optimizing the flight pattern to determine thermal locations<sup>13</sup>. During this research, the goal was to determine an optimal path for a single UAV to fly in order to identify thermals in a specific area. By having the UAV fly in an optimal spiral pattern, the UAV would be detecting thermals in the flight area to create a type of thermal map for the region. The UAV would have no knowledge of the thermals and would determine the location of a thermal based on the vertical air disturbances. The primary focus for this project was to minimize the mission time for the UAV to identify thermals in an area. The UAV would accomplish this by sensing strong vertical airspeeds using on-board sensors once inside the thermal, and not outside of the thermal.

An exploratory approach can benefit a UAV only when the thermal is in the path of the UAV. For other studies, the UAV was programmed with known information of probable thermal locations<sup>36</sup>. One such study used an altitude gain model to estimate the gain from a thermal of known size, strength, and location<sup>37</sup>. One known successful attempt to use thermal updrafts was with the NASA Dryden Research Center's UAV, Cloud Swift<sup>4, 34, 35</sup>. The Cloud Swift used real-time energy algorithms to determine the strength of the current thermal and determine an optimal circling radius. With these additional algorithms, the Cloud Swift was able to use 23 thermals, with an altitude gain of 173 meters from a single thermal, and flew over 60 minutes without using the motor to maintain flight. The Cloud Swift thermal detection was based on the algorithm's ability to interpret past thermal locations and assess if a current thermal is present. Data were collected for the area to indicate past presence of thermals for use in the thermal seeking process. In this case, the Cloud Swift was actively seeking thermals by using past experience to know where thermals were located and energy balance to recognize when thermals were found.

The strategy of using known thermal locations allows for quick simulation for flight optimization. Also, some of the prevailing research even states that thermal detection is unknown at this time and thus not taken into account. Kagabo states that "the strategies in place so far cannot predict with certainty the presence of an atmospheric thermal at a given location"<sup>38</sup>. With regards to mimicking soaring birds, as stated previously, memory is one possibility. Thus, the ability for a UAV to remember past thermal locations could help improve flight optimization for UAVs.

Another method UAV researchers are exploring to optimize thermal flying is through the use of flocks of UAVs. Research was found utilizing multiple UAVs for thermal detection where a group of UAVs communicate with one another while searching and using thermals during missions. In such research, simulations were made in which a group of UAVs would fly along a defined course while collecting data about possible thermals. Once a thermal was discovered by one UAV, data would be sent to the other UAVs in the group giving information of the size and

strength of the thermal. The UAVs could then change course to the found thermal, use the available energy, and then return to the original course until another thermal was found by another UAV. Again, this is an explorative approach to discovering thermal locations. These studies and simulations focused on the ability for multiple UAVs to be used to optimize flight paths for using thermal updrafts.

One simulation used a group of UAVs to detect thermal updrafts while using a Simultaneous Perturbation Stochastic Approximation (SPSA) method to determine thermal center and maximum vertical velocity<sup>11</sup>. Each UAV would fly in the direction of a specified waypoint until it detected a positive vertical airspeed. The UAVs were spaced at a distance less than the predetermined thermal diameter. Once an updraft is detected by a single UAV, coordinates and estimated updraft properties are sent to the other UAVs. Using energy analysis of the updraft, the individual UAVs would then decide whether to change course and use the thermal or continue on the current flight path based on the energy lost or gained using the updraft. For this simulation, twenty minute lifespan thermal models were used from previous atmospheric data from another project<sup>16, 35</sup>. Again, the UAVs relied on a collected data field to assess thermal patterns and did not have the capability to detect thermals from a distance.

## CHAPTER III

### 3 SIMULATIONS

From analyzing the background information from avian resources, five unique hypotheses were found regarding an avian species primary ability to determine thermal updraft locations: memory, feel, sight, search, extrasensory. By also looking through studies in manned, RC, and unmanned flight, the “feel” aspect showed the most promise with additional evidence to support the feasibility to locate updrafts and was therefore chosen for further exploration.

Simulations were created to test the functionality of ideal inflows from thermal updrafts. These simulations would also mirror the future experimental sensor packages in order simulate techniques to solve for the thermal inflow, or “third vector.” These simulations were to be used to complete Objective 2, as well as Objective 4 on the path of accomplishing this research endeavor’s goal. Simulations created include: simulating sink profiles to estimate ground air inflow caused by thermal updrafts, simulate different techniques in back solving the uniform flow vector with simulated wind conditions, and simulate the proposed graphical interface to be created with the sensor packages. These simulations were also to be used later on to make improvements on the experimental approach once the sensors were fully functional (Objective 4).

The following chapter will cover the initial simulations with information into what was being simulated, how it was simulated, along with pertinent information gleaned from the simulations.

### 3.1 SINK PROFILE

A sink flow is composed of streamlines that are directed towards the origin. The flow lines are radial lines where flow velocity varies with distance from the origin of the sink<sup>39</sup>.

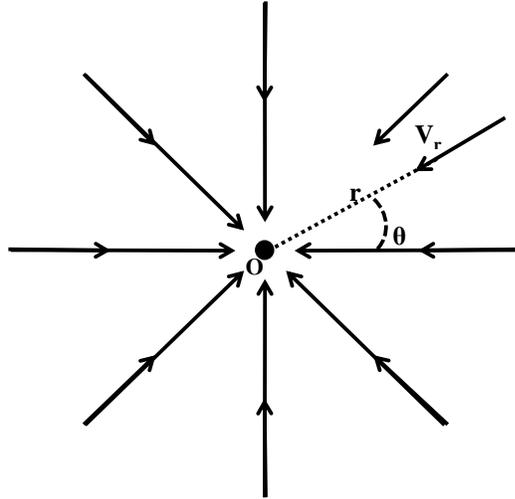


Figure 3.1: Sink flow

As mentioned in Chapter 2, the ground portion of column thermals are commonly thought to be similar to sink profiles, in that the thermals are fed warm air from the surrounding areas. The air mass is then changed from a horizontal velocity to a vertical velocity which is the useful component used in thermal flight.

By treating thermal updrafts as sink profiles, simple computational modeling was developed to study the effects and changes in wind shifts around modeled thermals caused by the ground inflow to the thermal. Wind vectors and thermal vectors would be calculated at varying distances around a simulated thermal to determine the combined vector condition. These models were used in conjunction with constructed air sensors, to experimentally measure air shifts in an area with possible thermal updrafts. The information recorded from the sensors and the

simulations would be helpful in determining an algorithm that could be used to triangulate a thermal updraft.

Stream and potential functions were used to model the shifts in winds due to thermals in a test area. The thermal components were modeled as 2D sink profiles where the streamlines were directed toward the sink or simulated thermal origin (Eq. 3.1 and Eq. 3.2).

$$\psi_{thermal} = \frac{-\Lambda}{2\pi} \theta \quad \text{Eq. 3.1}$$

$$\phi_{thermal} = \frac{-\Lambda}{2\pi} \ln r \quad \text{Eq. 3.2}$$

In Eq. 3.1 and Eq. 3.2,  $\Psi$  and  $\Phi$  are the stream function and potential function respectively,  $\Lambda$  represents the sink strength (ft<sup>2</sup>/s),  $r$  is the distance from the sink center, and  $\theta$  is the angle. To model the area around a thermal, the uniform wind component would also need to be modeled. Having a simulation with only the sink component would be representative of an optimal test case where only a thermal updraft was present. Addition of wind was treated as a uniform velocity component over the entire modeled area.

$$\psi_{wind} = V_{\infty} r \sin \theta \quad \text{Eq. 3.3}$$

$$\phi_{wind} = V_{\infty} r \cos \theta \quad \text{Eq. 3.4}$$

In Eq. 3.3 and Eq. 3.4,  $V_\infty$  represents the magnitude of the wind velocity. The two stream, or potential equations, were combined to model test areas that contained both uniform wind and thermal wind components.

$$\psi = \psi_{thermal} + \psi_{wind} = \frac{-\Lambda}{2\pi} \theta + V_\infty r \sin \theta \quad \text{Eq. 3.5}$$

$$\phi = \phi_{thermal} + \phi_{wind} = \frac{-\Lambda}{2\pi} \ln r + V_\infty r \cos \theta \quad \text{Eq. 3.6}$$

To find the individual velocity components ( $u$  and  $v$ ) at a given location, radial components were converted to Cartesian coordinates ( $x$  and  $y$ ) using the derivatives of the combined sink and uniform flow stream and potential functions.

$$u = \frac{\partial \phi}{\partial x} = \frac{\partial \psi}{\partial y} = V_x - \frac{\Lambda}{8\pi} \frac{(x - x_s)}{(x - x_s)^2 + (y - y_s)^2} \quad \text{Eq. 3.7}$$

$$v = \frac{\partial \phi}{\partial y} = \frac{-\partial \psi}{\partial x} = V_y - \frac{\Lambda}{8\pi} \frac{(y - y_s)}{(x - x_s)^2 + (y - y_s)^2} \quad \text{Eq. 3.8}$$

In Eq. 3.7 and Eq. 3.8,  $x_s$  and  $y_s$  represents the Cartesian location of the sink. Using a uniform flow and sink component, the stream lines of an area would be equivalent to Figure 3.2.

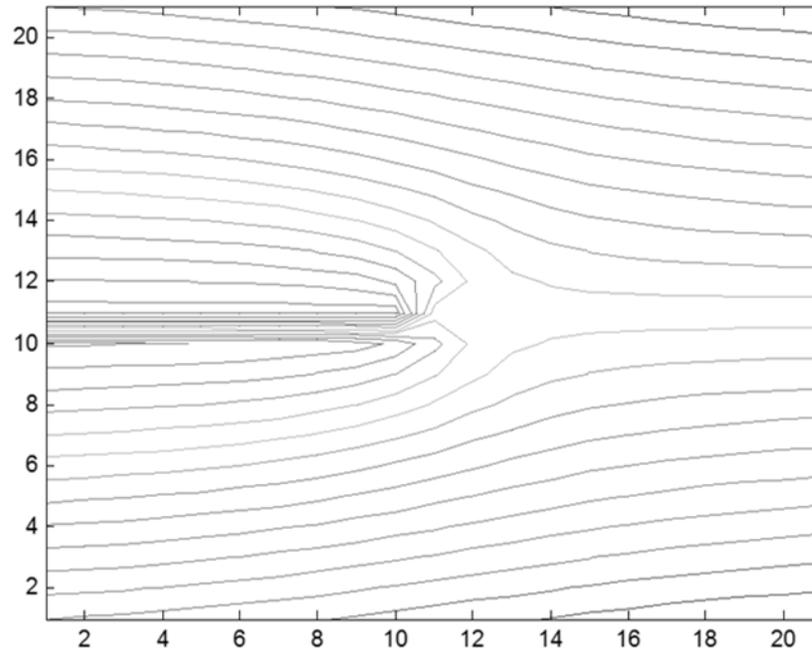


Figure 3.2: Stream lines of uniform with sink flow

Seeding the planned testing area with a constant wind profile and added thermal wind components, Figure 3.3 and Figure 3.4 show initial simulations with stream functions being calculated over the prescribed testing area for future data collection.



Figure 3.3: Simulation of test area with sink flow only



Figure 3.4: Simulation of test area with uniform and sink flow

These figures show the initial frame work for later simulations. Figure 3.3 and Figure 3.4 incorporate both uniform and thermal wind components at different location across the field and allow for simulation of thermal locating to be tested.

During some of the initial simulations, the simulated thermal component was chosen to be equivalent to a small factor of the overall wind speed thus looking at small disturbances within the test field. In later simulations, the sink strength was either specified or was calculated based on thermal size and updraft velocity. The sink strength is defined as volume flow rate per unit depth into the surface<sup>39</sup>. Where the strength is defined by Eq. 3.9.

$$\Lambda = \frac{\dot{m}}{\rho l} = 2\pi r V_r \quad \text{Eq. 3.9}$$

In Eq. 3.9,  $\dot{m}$  is the mass flow rate,  $\rho$  is the air density,  $l$  represents the length of the sink, and  $V_r$  is the radial velocity. Thus the radial velocity due do the sink or simulated thermal could be equated to Eq. 3.10.

$$V_r = \frac{\Lambda}{2\pi r} \quad \text{Eq. 3.10}$$

For many of the simulations, the sink strength was entered as a given or specified component within the code. However, it was also desired to try and equate the sink strength to thermal updraft parameters. Looking back at Eq. 3.9 for sink strength definition, a thermal strength could be related back to the mass flow rate of the thermal updraft ( $\dot{m}_{updraft}$ ).

$$\dot{m}_{updraft} = \rho A_t V_t = \rho \pi R_t^2 V_t \quad \text{Eq. 3.11}$$

In Eq. 3.11,  $A_t$ ,  $R_t$  and  $V_t$  represent the thermal area, radius, and thermal updraft velocity respectively. Substituting Eq. 3.11 into Eq. 3.9, as well as defining the thermal updraft velocity as a Gaussian distribution (Eq. 2.1), the sink strength could be modeled as a function of a thermal updraft with known radius and maximum updraft velocity while using an assumption of a height value of 1.

$$\Lambda = \frac{\dot{m}}{\rho l} = \pi R_t \int_0^{R_t} V_t(r) dr \quad \text{Eq. 3.12}$$

A conservation of mass approach was also used to try and better model the thermal updraft in simulations. Assuming all of the warm air from the ground inflow is transitioned to the vertical component of the thermal, the mass coming into the control volume would be equal to that leaving the control volume (Figure 3.5 and Eq. 3.13).

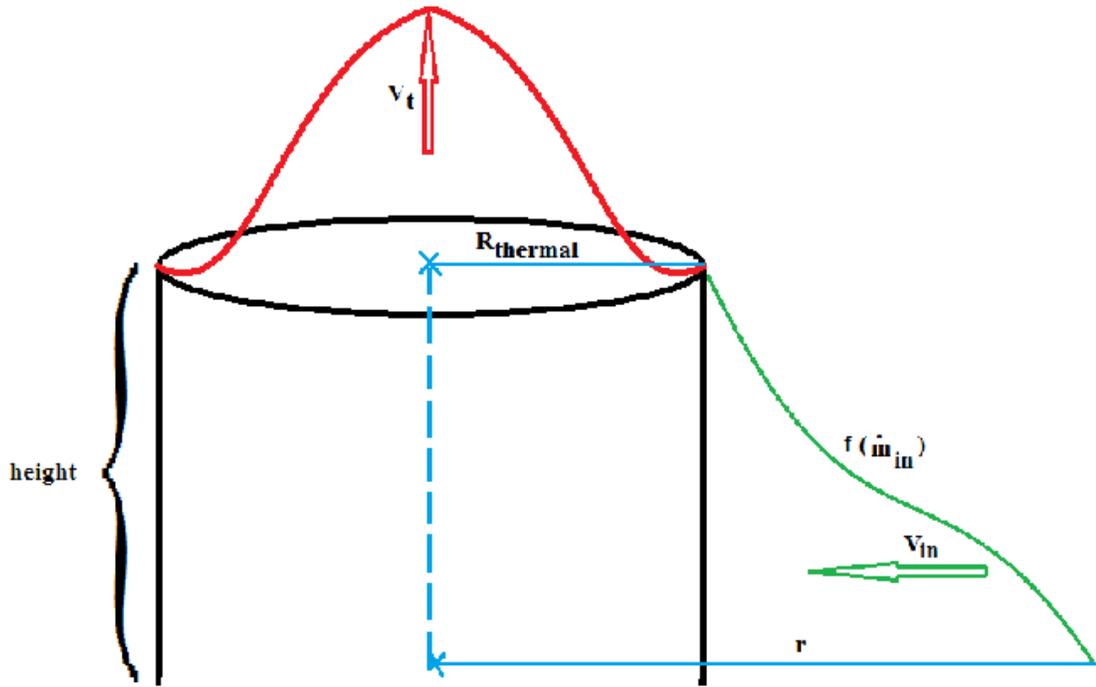


Figure 3.5: Thermal column cylindrical control volume

$$\dot{m}_{updraft} = \dot{m}_{radial} \quad \text{Eq. 3.13}$$

Where the updraft component would be defined in terms of the thermal updraft velocity and radius (Eq. 3.11). And the ground inflow mass flow rate ( $\dot{m}_{radial}$ ) being equated to the velocity ( $V_r$ ) at a radial distance ( $r$ ) from the thermal center with a specified height ( $h$ ) component.

$$\dot{m}_{radial} = \rho A V_r = \rho 2\pi r h V_r \quad \text{Eq. 3.14}$$

Combining Eq. 3.10, Eq. 3.11, and Eq. 3.14, the sink strength could be estimated as a function of the thermal size components (Eq. 3.15).

$$\Lambda = \frac{\pi R_t^2 V_t}{h} \quad \text{Eq. 3.15}$$

For an averaged updraft velocity, Eq. 3.15 is visualized by Figure 3.6 where different types of thermal updrafts could be representative of similar sink strength profiles.

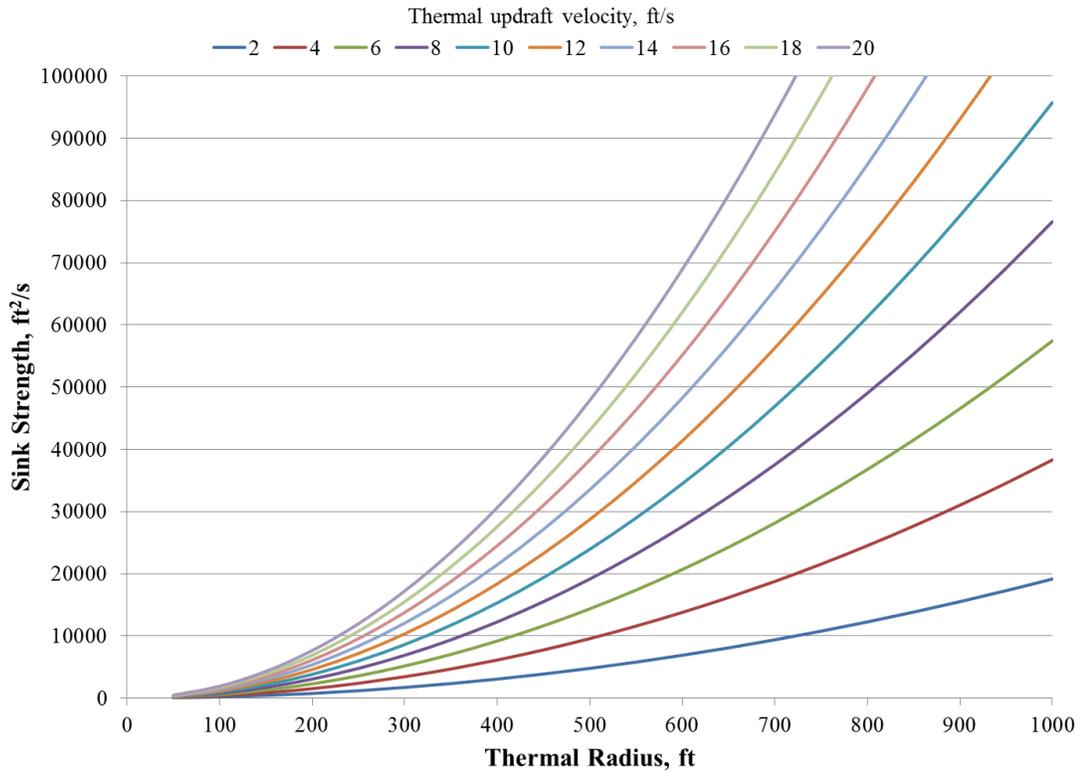


Figure 3.6: Sink strength based on thermal factors

The height value was estimated to be the height of the surface layer, which is estimated to be 10% of the convective boundary layer<sup>1</sup>. The mixing layer height varies throughout the day, with values ranging from 1 km in the early mornings to upwards of 2 km for the afternoons. For the height value in these initial calculations were based on about 320 ft, which was based on the lower 10% of a 1 km thick boundary layer. Figure 3.6 shows that for a constant theoretical sink strength, numerous idealized thermal updrafts could exist.

### 3.2 EXPERIMENTAL SIMULATION

Instead of looking at the entire simulation area, only specific points were calculated and modeled to represent the proposed sensor placements for future experiments. With the evidence found from the literature surveys as well as the initial simulations previously shown, simulations were updated to simulation detection using a few, localized points of known wind profiles. Once the thermal and wind components were calculated at each location, a new total velocity vector would be found and treated as the theoretical measured velocity profile from a ground based sensor. These simulations were created to mirror the experimental sensor packages in order to test and make improvements on the experimental approach. As with the idea of the “third vector” of using a feel aspect for finding thermal updrafts, these simulations would simulate finding the thermal inflow vector based on simulated surrounding ground air flow involving wind speed and direction at different locations in the testing area. These simulations were also to be used to improve on the methods of subtracting the “average” wind speed to determine the thermal inflow component of the measured wind.

The simulation of the experiment was setup with 3 sensors placed around a given field (Figure 3.7).



Figure 3.7: Simulation of experimental setup

Each node was given a uniform wind component (green line), which was constant for each node, along with a thermal component (red line) that was a function of the distance from the simulated thermal location (red dot). The simulation would then combine the thermal and uniform velocity vectors to determine a unique, simulated vector for each node (black).

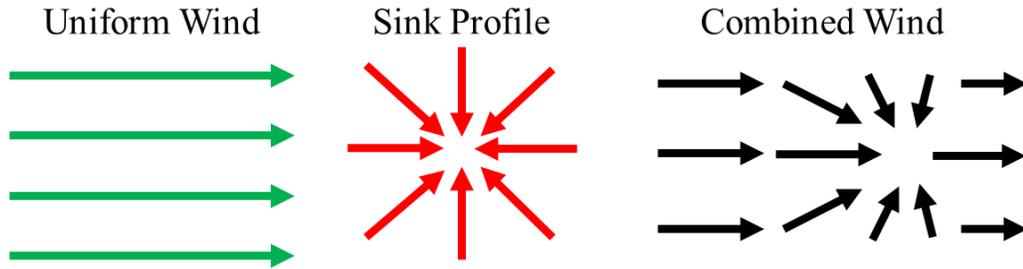


Figure 3.8: Uniform plus sink profile

### 3.2.1 THIRD VECTOR APPROACH

With the information found in the literature survey, the “third vector” method was used as the focus of testing it’s feasibility for remotely detecting thermal updrafts. In this subsection, the idealized “third vector” approach will be discussed, where the basic equations and organization will be shown.

To find the shifts in the wind, in an attempt to calculate the thermal vector based on the “third vector” approach, an average wind vector would need to be subtracted from the immediate wind vector to determine the direction of the shift (Figure 3.9).

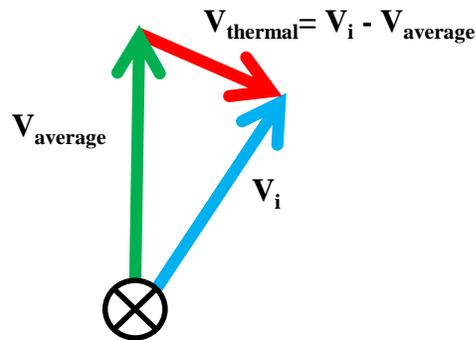


Figure 3.9: Vector approach of for sensors

By keeping track of a time span of wind measurements, the average wind would serve as the uniform wind component from the uniform and sink profile. To solve for the thermal vector ( $\vec{V}_{thermal}$ ), the uniform or average vector ( $\vec{V}_{average}$ ), would be subtracted from the instant or current wind measurement ( $\vec{V}_i$ ).

$$\vec{V}_{thermal} = \vec{V}_i - \vec{V}_{average} \quad \text{Eq. 3.16}$$

In order to subtract the vectors within the computer code, either the experimental interface or the simulations, each of the wind vectors are decomposed into  $x$  and  $y$  components using Eq. 3.17 through Eq. 3.20<sup>40, 41</sup>.

$$V_x = speed_j \cdot \cos(direction_j) \quad \text{Eq. 3.17}$$

$$V_y = speed_j \cdot \sin(direction_j) \quad \text{Eq. 3.18}$$

Where  $speed_j$  represents the wind speed, and  $direction_j$  is the wind direction during a single data point. The uniform wind vector components ( $Avg_x$  and  $Avg_y$ ) are based on the average of previously collected  $N$  data points.

$$Avg_x = \frac{1}{N} \sum_{j=1}^N speed_j \cdot \cos(direction_j) \quad \text{Eq. 3.19}$$

$$Avg_y = \frac{1}{N} \sum_{j=1}^N speed_j \cdot \sin(direction_j) \quad \text{Eq. 3.20}$$

Once the vector components are each calculated, the angle of the thermal vector ( $\theta_{thermal}$ ) is calculated using Eq. 3.21.

$$\theta_{thermal} = \tan^{-1} \left( \frac{V_y - Avg_y}{V_x - Avg_x} \right) \quad \text{Eq. 3.21}$$

Only the angle is calculated and currently used to triangulate the position of the air inflow. The thermal angle is used to indicate the direction of possible thermal inflow. In conjunction with other sensors, the thermal vector would be used to triangulate areas of thermal activity.

### 3.2.2 THIRD VECTOR SIMULATION

To simulate the “third vector” approach, the same concepts and calculations mentioned in the previous section (Eq. 3.16 through Eq. 3.21) would be used. In this section, the changes from the idealized “third vector” approach and the simulation will be discussed.

First, each simulated sensor node would need to be given a wind condition involving a uniform and thermal component. In order to do this, a constant, uniform wind speed and direction ( $\mathbf{V}_{uniform}$ ) is specified for the testing area along with a thermal component ( $\mathbf{V}_{thermal}$ ) as a function of radius from thermal center being added at predetermined times. Each node would combine these 2 components to simulate the felt wind condition (Figure 3.10).

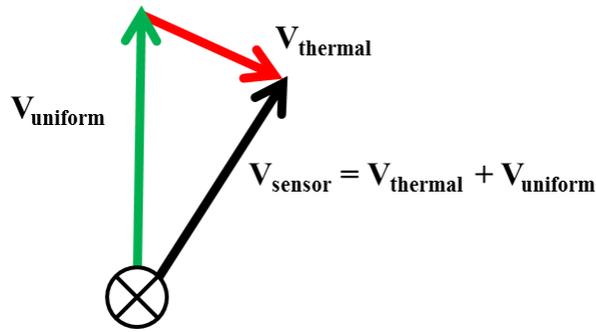


Figure 3.10: Simulation vector approach

The sensor vector ( $\mathbf{V}_{\text{sensor}}$ ) is used as the instantaneous vector for the thermal direction calculation. The average component is then created from previous data points of the calculated  $\mathbf{V}_{\text{sensor}}$ . The thermal direction is then calculated using Eq. 3.21.

Simulations were ran with only the uniform wind being present for the initial data points to allow the average vector to be calculated and stored in the arrays created for the average vector. At a specified time, a thermal would appear at a predetermined location with a specified thermal or sink strength using the methods discussed in Section 3.2. The simulation would then use Eq. 3.21 to calculate the thermal direction for each node to simulate the feasibility of finding the shift due to the thermal component.

### 3.2.3 AVERAGING METHODS

Different averaging and wind elimination techniques were used to algebraically remove the wind component from the combined velocity vector to back solve for the thermal direction. The following are some of the methods that were attempted to solve for the average or uniform wind component used in initial simulations and was eventually used in experimentation.

Originally it was thought to average multiple or all sensors or locations as a basis for the local wind component. However, initial results showed a large range of averages when compared to the specified simulated wind component. With a constant thermal location, thermal strength,

and constant uniform wind speed programmed, the wind direction was changed between 0° and 360° from a northern direction (vector direction). During some simulations, the averaged wind value of 3 different nodes varied up to values as high as 17% of the simulated wind speed (Figure 3.11).

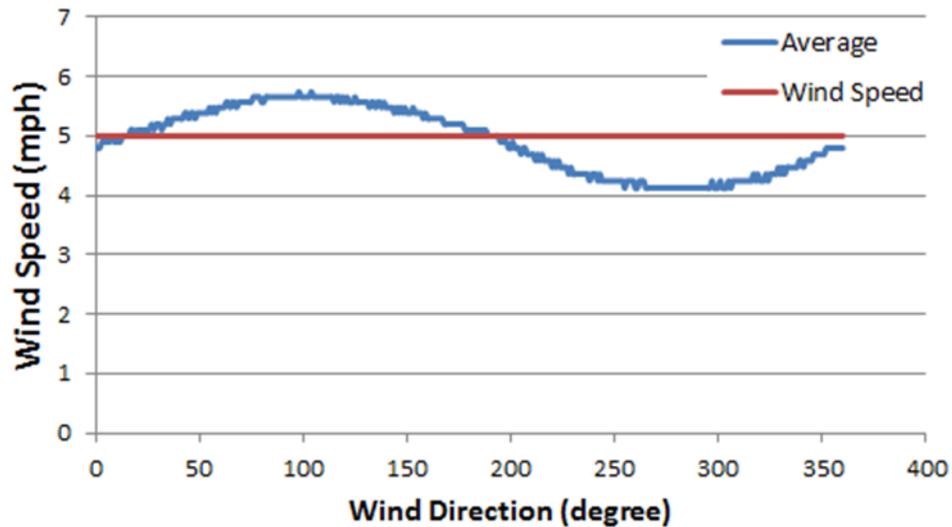


Figure 3.11: Changes in global average with direction during constant thermal input

The values in Figure 3.11 were from a simulation consisting of a constant thermal location, with a constant 5 mph wind speed which was rotated between 0° north clockwise through 360°. The “global” average wind speed was found by averaging the combined wind vector (thermal component plus wind component) from all 3 simulated sensor locations. The simulated average would drift even higher for higher wind speed values.

This method of subtracting the average velocity vector of all three locations was simulated to look at the effect this change would have on the calculated thermal direction. For this simulation, the calculated average from the 3 locations were then subtracted from each

sensor's velocity vector. Once the average was subtracted, the direction was plotted as a straight line (orange) from the sensor to represent the direction of the felt shift (Figure 3.12).



Figure 3.12: Simulated shift using global average

From Figure 3.12, the simulated shift directions are shown to be unreliable when determining thermal direction. Each sensor's direction does give indication of thermal activity to the Southwest of each sensor, while the thermal is only southwest of 2 of the 3 sensors.

Averaging between multiple sensor locations was showing to be an inefficient method to find shifts due to possible thermal activity. Instead of relying on multiple sensors, and individual

averaging approach was to be simulated. If each node was to keep a moving average of the measured airspeed and direction, then each sensor would be responsible for detecting local shifts. If a newly formed thermal was to be detected and make a sudden shift in wind states, by subtracting the current average from the current measured wind state would reveal the possible thermal direction or direction of the newly added vector component. Once the thermal is felt and added to the measured vector, the moving average would begin to be affected and the direction would start to slowly shift until the average was equivalent to the measured vector.

Once the shift occurs, the moving average would start to incorporate the new elements, and thus the shift's direction would slowly start to veer away from the initial value. The shift direction would only be visible for a short time, but could be lengthen if a larger average for the uniform was taken, assuming the average contained a low standard deviation (Figure 3.13).

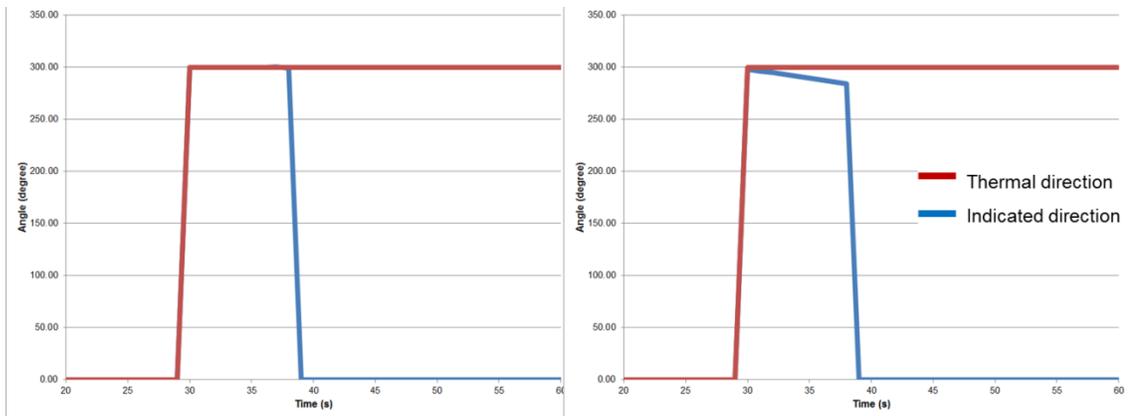


Figure 3.13: Simulated thermal direction from using single node averaging

Assuming no initial thermal activity, Figure 3.13 shows that given enough time for the sensor to compute an accurate average, the calculated shift would be in the direction of the theoretical thermal direction for a short time. Once the shift occurred, and if maintained, the

average would then begin to incorporate the new vector and thus the calculated shift direction would return to 0.

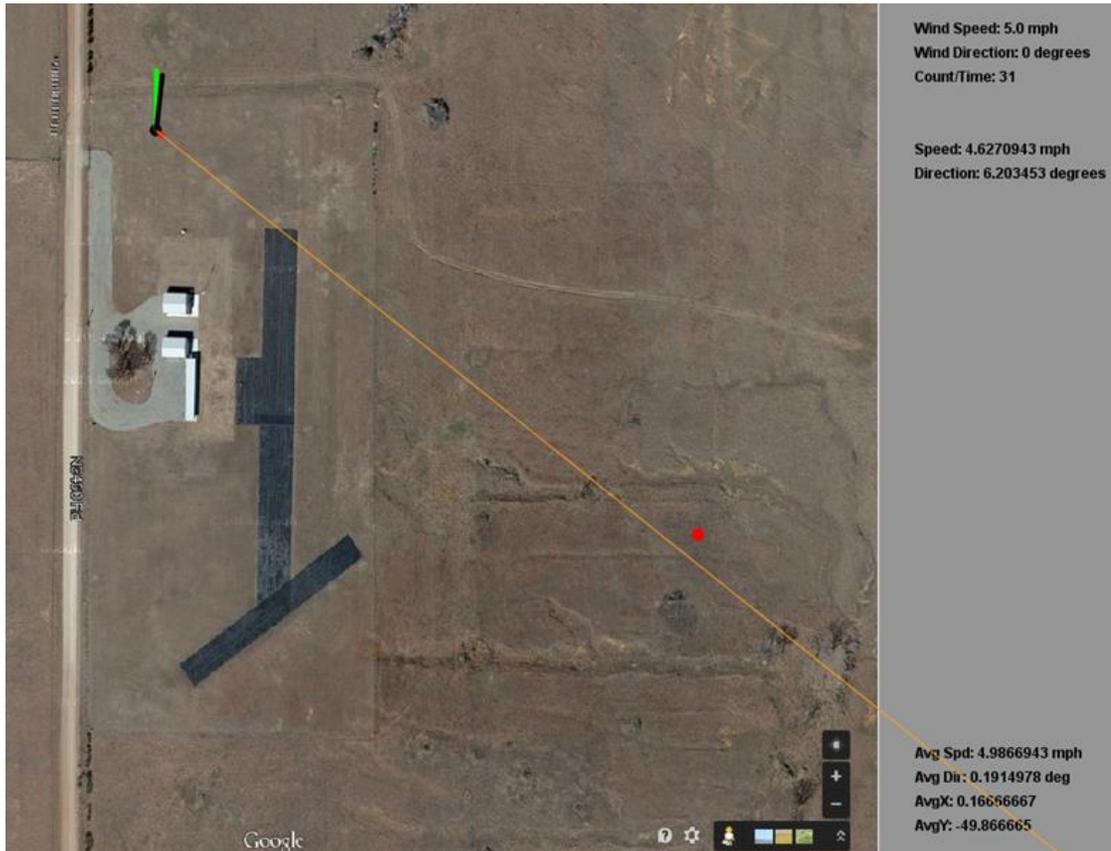


Figure 3.14: Example of single averages showing thermal convergence

The idea, by using multiple nodes or sensor locations, a thermal area could still be triangulated by looking at multiple shift convergences (Figure 3.15).

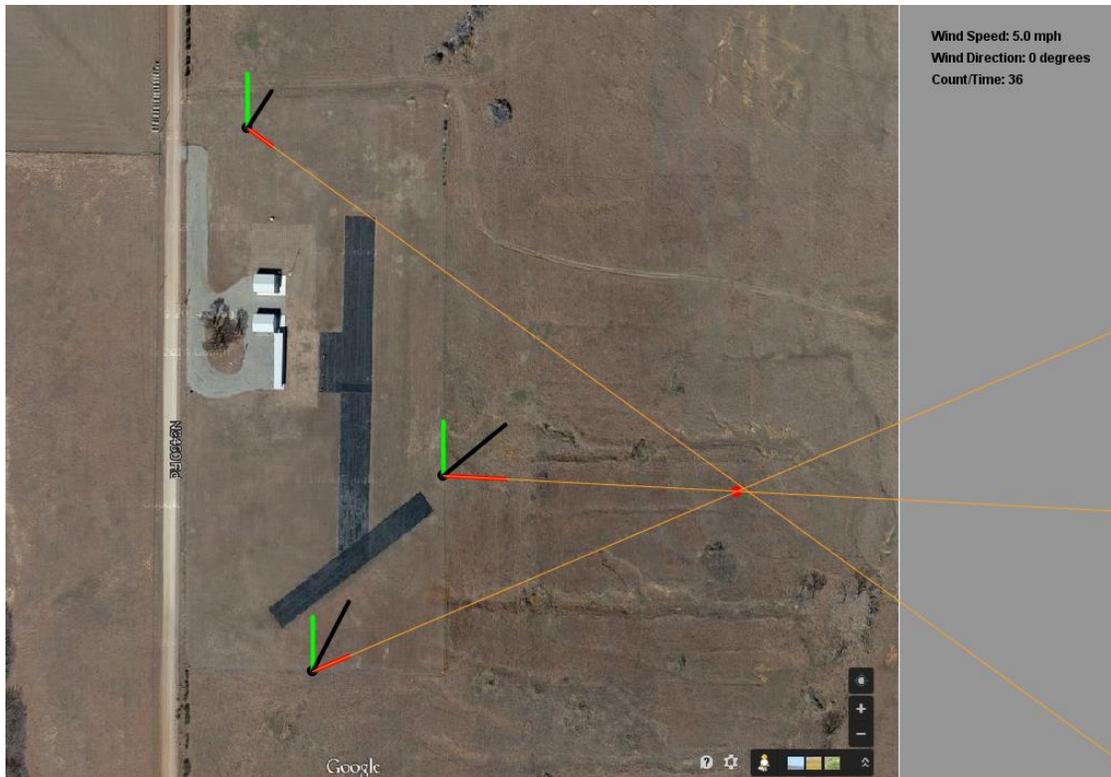


Figure 3.15: Example of single averages showing thermal triangulation

While using an individual averaging method where each sensor would keep an independent moving average of wind data points, additional techniques were examined to further aid in determining thermal location. The initial concept that was used in Figure 3.14 and Figure 3.15, was to use a single data point to subtract a moving average. However while doing initial sensor testing, it was seen that the directional sensor could still be in motion while tracking to a new position. To help eliminate this movement, it was determined that the instantaneous vector could benefit from taking moving average of the past few data points (Eq. 3.22 and Eq. 3.23).

$$V_x = \frac{1}{F} \sum_{j=1}^F speed_j \cdot \cos(direction_j) \quad \text{Eq. 3.22}$$

$$V_y = \frac{1}{F} \sum_{j=1}^F speed_j \cdot \sin(direction_j) \quad \text{Eq. 3.23}$$

In these equations, the  $F$  value would be small compared to  $N$  used for the uniform wind components. This second average was to help as the wind vane moved with changes in wind direction and minimize uncertainty with the inertial and movement used when calculating  $V_i$ . This double moving average could be utilized with different techniques based on the separate, if any, of the 2 averages. The data points used in both averages could either share data points or be independent of one another and contain no overlap (Figure 3.16).

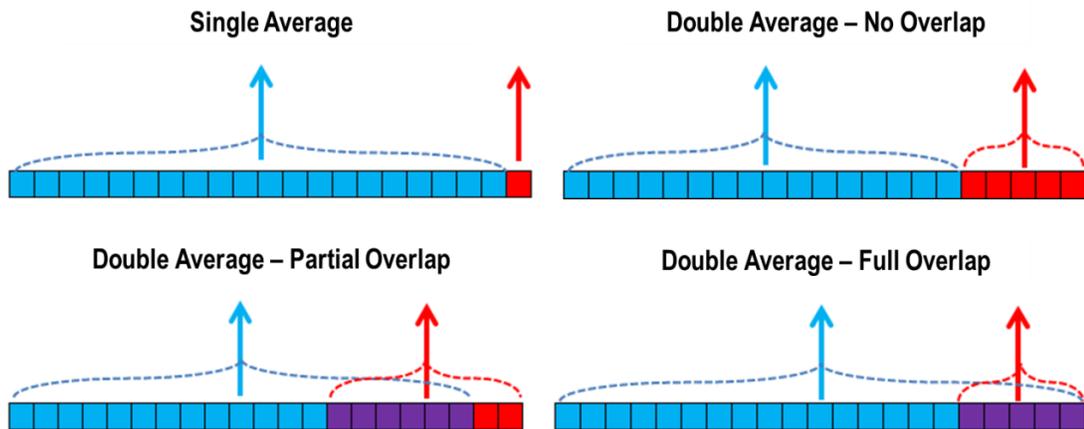


Figure 3.16: Averaging methods

In Figure 3.16, the blue vector would be the larger averaged vector used to represent the uniform wind component, while the red vector would be used for the instantaneous vector. Each of these methods were assessed to look at more efficient ways of calculating the thermal component.

It was only after analyzing experimental data that it was determined that both the average and the instantaneous vectors would each be composed of a moving average that would not overlap. Additional methods for averaging, such as full overlap and partial overlapping, were also examined but were found to be less adequate than the non-overlap. Additional information on the data that was collected and analyzed for comparing the averaging methods will be discussed on Section 5.1.

## CHAPTER IV

### 4 EXPERIMENTAL SETUP

To experimentally determine the feasibility with locating thermals from changes in air properties, a ground based sensor network was designed to measure local air properties. Using commercially available electronic components, sensors were constructed to measure air properties around a flight field in order to measure the changes that would occur around forming thermal updrafts. These sensors were designed to measure wind speed and direction, temperature, barometric pressure, and relative humidity. The information was collected on a 2 second interval then wirelessly transmitted to a local computer which displayed the values in real time via a graphical user interface (GUI) as well as store the data for future use. The sensors discussed in this Chapter, fulfill Objective 3 towards the course of completing the tasked goal.

The following sections go into greater detail on the specifications on the individual components and software used in creating the sensor packages. The first section will examine the development of the sensor packages with information regarding the individual components and coding involved with each respective piece. Afterwards, the visual interface will be examined, showing the major steps and primary functions of the interface.

## 4.1 SENSOR DESIGN

The following section will dwell on the hardware and software systems that went into creating the sensors used to collect experimental data sets. First the individual components that comprised the data sensor will be described. Each component will be briefly described based on the manufactures reports that were available along with observations made during the utilization of the component. During these sections, specifics pertaining to the components specific libraries and code strategies used in the creation of the Arduino sketches will also be explained. Overall Arduino Sketch coding libraries and strategies will not be described, only the information relevant to the creation of the sensor packages. After which, the final sections will illustrate the final construction, total cost evaluation, and testing of the sensor package will be described

### 4.1.1 MICROCONTROLLER

The over the counter Arduino Uno microcontroller was chosen to read and transmit air properties.



Figure 4.1: Arduino Uno board

The Arduino Uno is based on the ATmega328 microprocessor. It has 14 digital input/output pins of which 6 provide Pulse Width Module (PWM) outputs, 6 analog inputs, 32k flash memory, 16 Mhz clock speed, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. The Arduino Uno can be powered via a 2.1 mm jack with a power source between 7 and 20 V and supply components with either 3.3 or 5 V. The Arduino would be powered via a 9 V battery. The Arduino Uno would serve as the core of the sensor. It would supply power sensor components and programmed to read data from over the counter sensors, perform necessary conversions and/or computations, and then relay that information to a central computer for storage and analysis.

The Arduino would be programmed through an open-source Arduino Software IDE (integrated development environment). The Arduino IDE makes it possible to write code (called sketches) and then upload it to the Arduino board. The Arduino IDE is written in JAVA and based on open-source software, yet code written in the sketches is very similar to C++ programming.

#### 4.1.2 WIRELESS TRANSMISSION

XBee Pro 63 mW PCB Antenna - Series 2B (ZigBee Mesh) was used in conjunction with the Arduino Uno to relay data from the sensor back to a local computer for visualization and storage.

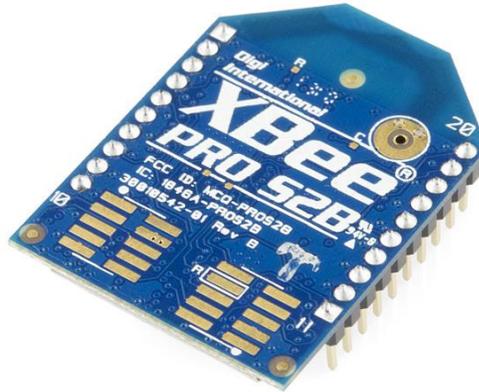


Figure 4.2: XBee Pro 63 mW PCB Antenna - Series 2B (ZigBee Mesh)

The XBee Pro 63 mW PCB Antenna runs off of 3.3 V at 295 mA. The Xbee Pro has a built-in antenna, 250kbps max data rate with a 63mW output (+17 dBm) and is rated for a 1 mile (1600 m) range. The XBee has 6 10-bit ADC input pins, 8 digital IO pins with a 128-bit encryption.

To integrate the XBee antenna, an XBee Explorer and XBee shield were purchased.

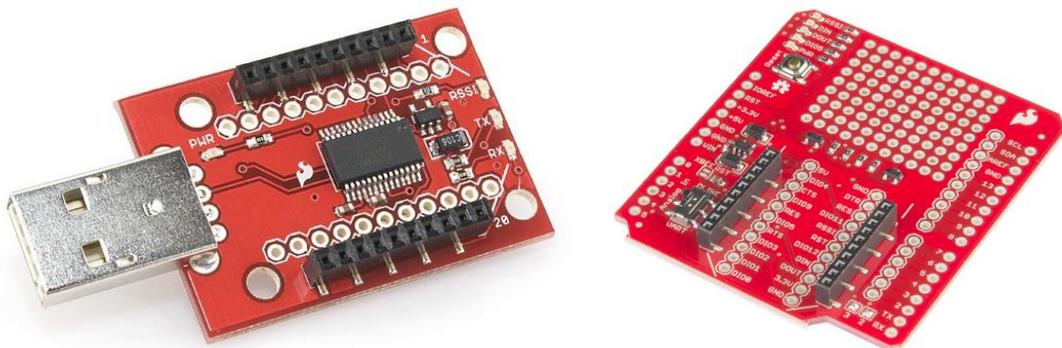


Figure 4.3: XBee Explorer & Arduino XBee Shield

The XBee Shield allows for an XBee unit to be directly attached to an Arduino unit (such as the Arduino UNO) without having to individually connect the input/output pins of the XBee to corresponding pins on an Arduino unit. The XBee Explorer allows for an XBee to be connected via a USB port directly to a computer console allowing for data to be streamed directly to a computer console without the need for a secondary Arduino to read the data as well as updating firmware.

The coding of the XBee within the Arduino is minimal. The XBee however does have to be uploaded with its own firmware and initialized through secondary software programs and not with the Arduino IDE. The X-CTU program is the official configuration program used for XBee radios and was used for this project<sup>42</sup>. Using the XBee Explorer, an XBee radio can be plugged into a Windows computer that contains the X-CTU software and configured for use with different systems. The first step when using the XBee is to update the firmware for the specific XBee module that one is using. The XBee radios can also be configured for their specific roll or class (such as end device, router, or coordinator) along with AT or API functions. The Coordinator function for an XBee radio is responsible for defining and maintaining the network<sup>42</sup>. There is only one coordinator within a network. A Router is capable of sending, receiving, and routing information within the network. There can be multiple router radios within the network. An end device is a lower level router. It is capable of joining a network while sending and receiving information, but is unable to relay information as a router device. This allows the end device to function at a lower power level and can go into a nonresponsive sleep mode.

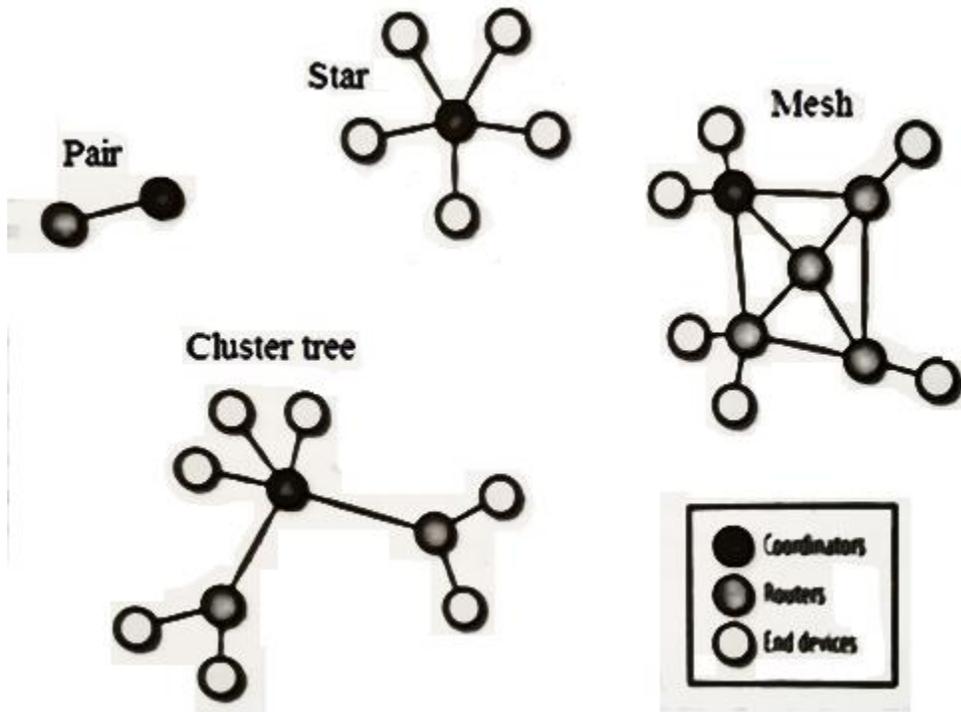


Figure 4.4: Wireless topology<sup>42</sup>

For the sensor packages, 3 of the XBee radios were configured as routers, while the 4<sup>th</sup> was set as coordinator.

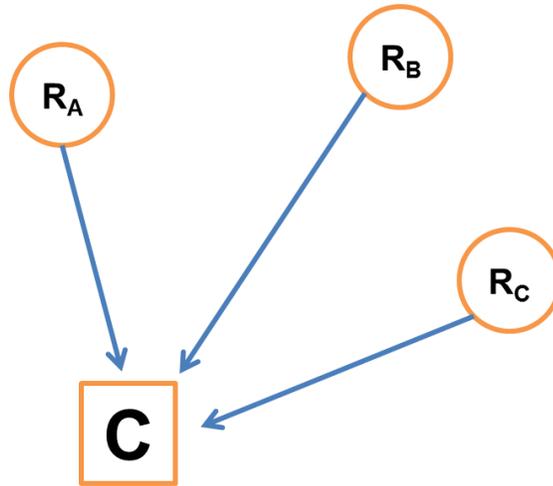


Figure 4.5: Sensor wireless configuration ( $R_A$   $R_B$   $R_C$  = Routers,  $C$  = Coordinator)

The coordinator would be connected to the primary computer for receiving sensor data, while the routers would be transmitting the collected sensor data back to the coordinator. The routers were configured to only send data to the coordinator and not have any interaction with each other.

#### 4.1.3 WIND SPEED

To measure wind speed, the Inspeed Vortex Wind Sensor was used. This anemometer is a 3-cup rotor connected to a magnet and reed switch allowing for one pulse per rotation.



Figure 4.6: Inspeed Vortex Wind Sensor

The sensor package was designed to count the number of pulses from the Vortex Wind Sensor and convert to wind speed using the following conversion:

$$1\text{Hz} = 1\text{pulse/second} = 2.5\text{ mph} \quad \text{Eq. 4.1}$$

Two wires provide the output for the anemometer: one connected to ground while the other is connected to one of the Arduino's PWM input slots (Pin 2). The Vortex was wired with a 100 nF capacitor and programmed with an internal 5 V pull-up resistor in order to account for debouncing with the Vortex's internal reed switch (explained in more detail in Section 4.2). The Arduino was coded with an interrupt, which would count the number of pulses during a 2 second time frame in order to calculate the wind speed during that interval.

When coding an Arduino sketch for the Inspeed Vortex, the first steps necessary were to define and initialize the PWM pin to be used to measure the wind speed. In the initial setup of

the sketch, the pin was to be defined, the behavior of the pin is to be specified (INPUT or OUTPUT, in this case it is INPUT), and the interrupt service routine is to be called.

```
#define Pin_Speed 2
...
void setup()
{
  ...
  pinMode(Pin_Speed,INPUT);
  digitalWrite(Pin_Speed, HIGH);
  attachInterrupt(0, countAnemometer, FALLING);
  ...
}
```

Figure 4.7: Arduino sketch setup regarding Inspeed Vortex

Once the pin and interrupt are declared in the beginning of the sketch, the interrupt itself could be created. The purpose of the interrupt was to keep count of the number of pulses or rotations that occurred during a specific time frame. For this sensor, the interrupt was simply a counter: every time the interrupt occurred, that is when a full rotation of the Inspeed Vortex takes places, add one to the counter.

```
void countAnemometer()
{
  numRevsAnemometer++;
}
```

Figure 4.8: Arduino sketch Inspeed Vortex interrupt

In order to use the counter that was being incremented by the interrupt, the main loop of the Arduino Sketch would call on a separate subroutine to calculate the wind speed from the number of measured pulses utilizing Eq. 4.1. This subroutine would be called in the main loop every 2 seconds.

```
float calcSpeed()
{
  int iSpeed;
  float xx;
  long spd = 25000;
  spd *= numRevsAnemometer;
  spd /= speed_calc;
  iSpeed = spd;
  xx = iSpeed/10.0;
  numRevsAnemometer = 0;
  return xx;
}
```

Figure 4.9: Arduino sketch Inspeed Vortex subroutine to calculate wind speed

In the subroutine (Figure 4.9) the number of pulses or rotations was multiplied by a constant 25000. The 25000 was found by taking Eq. 4.1 and converting the time to milliseconds instead of seconds as well as multiplying by 10 to be used to account for decimal formatting within the subroutine (Eq. 4.2).

$$spd = \frac{2.5 \text{ mph}}{1 \text{ Hz}} \cdot 1000 \frac{\text{mili seconds}}{\text{seconds}} \cdot 10 \quad \text{Eq. 4.2}$$

Once the number of rotations is multiplied by the *spd* constant, the new value is then divided by the total time that has occurred during the counting of the rotations: which in this case is 2 seconds. This value then gives the speed in mph multiplied by 10. The subroutine then divides the value by 10 to store in a float variable to give the final speed in mph with a single decimal value available. The value is then returned by the subroutine back to the main loop where it is then printed to the screen or transmitted through the XBee signal.

```
void loop()
{
  ...
  spd = calcSpeed();
  ...
  Serial.print(spd,2);
  ...
}
```

Figure 4.10: Arduino sketch Inspeed Vortex main loop

#### 4.1.4 WIND DIRECTION

To measure wind direction, the Inspeed E-Vane was chosen.



Figure 4.11: Inspeed E-Vane

The E-Vane comes equipped with three wires: ground, power, and signal. The E-Vane is to be supplied between 2.7 and 5.5 V at a 12 mA current, has an output range of 5% to 95% of the input voltage, and is rated at an accuracy of  $\pm 6^\circ$ . It is recommended that the E-Vane be powered with 5 V to provide an analog output range of 0.25 to 4.75 V. This analog output range can then be converted to a degree range ( $0^\circ = 0.25$  V and  $360^\circ = 4.75$  V)

When programming the Arduino sketch to determine the wind direction, the Arduino would need to be capable of reading in the analog signal from the E-Vane and transcribing it to a usable angle measurement. The Arduino is capable of reading in the analog signal from the input pin (Pin A3). Once the value was read, it was necessary to perform a conversion from analog (0 to 1024) to voltage (0 to 5 V) and then from the output range voltage (5% to 95% of input) to the angle (0 to  $360^\circ$ ). The E-Vane was powered through the 5 V port of the Arduino, thus giving an output range between 0.25 V and 4.75 V to be used in the conversion of the measured angle.

$$\text{Degrees} = 80 \cdot \text{Voltage} - 20$$

$$\text{Degrees} = \text{Analog} \cdot \frac{400}{1024} - 20$$

Eq. 4.3

All of these conversions were accomplished through a single line of code after the analog signal was read and stored (Figure 4.12).

```
double ang; //angle variable
....
void loop()
{
  ...
  angT = analogRead(direPin);
  ang = (angT * 0.390625) - 20;
  ...
  Serial.print(ang,1);
  ...
}
```

Figure 4.12: Arduino sketch for direction

Once this simple conversion was accomplished, the value could then be transmitted or outputted for later analysis.

#### 4.1.5 TEMPERATURE & HUMIDITY

To be capable of measuring the surrounding air temperature and relative humidity, the RHT03 Humidity and Temperature Sensor was chosen.

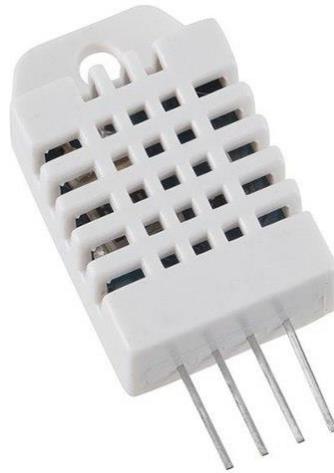


Figure 4.13: RHT03 humidity and temperature sensor

The RHT03 (also known by DHT-22) is a low cost humidity and temperature sensor with a single signal interface which comes already calibrated. This sensor has a 3.3-6 V input range rated at 1-1.5 mA current level. The RHT03 has a temperature range of -40 to 80 °C (-40 to 176 °F) and a rated accuracy of +/- 2% for relative humidity and +/- 0.5 °C for temperature. A 1 kOhm pull-up resistor was added to the signal pin. The pull-up resistor was wired externally for this component, but it could be possible to add an internal pull-up resistor on the Arduino. It is also advised to add a 100 nF capacitor between the power supply and ground for wave filtering.

When coding the Arduino to read the values from the RHT03, the DHT22 library was incorporated. The DHT22 library was created for the DHT22 sensor, which is also known as the RHT03. This library allowed for temperature and humidity data to be readily accessible by the Arduino. The library consisted of error functions as well as call functions for data to be gathered easily with fewer lines of code.

Utilizing the DHT22 library consisted of first adding the library into the sketch and then defining the pin on the Arduino to read the RHT03.

```
#include <DHT22.h>
...
#define DHT22_PIN 11
...
DHT22 myDHT22(DHT22_PIN);
...
double faren;
double humid;
```

Figure 4.14: RHT03 initialization

The DHT22 library was downloaded and added to the Arduino library path for access. Along with declaring the data pin on the Arduino Uno (pin 11), 2 double precision variables were created to store the values of temperature and humidity. In order to obtain and convert the data being returned by the RHT03, a switch was used to determine if data was being sent and was readable.

```

errorCode = myDHT22.readData();
...
switch(errorCode)
{
  case DHT_ERROR_NONE:
    faren = myDHT22.getTemperatureC()*1.8 + 32;
    humid = myDHT22.getHumidity();
    break;
  case DHT_ERROR_CHECKSUM:
    faren = myDHT22.getTemperatureC()*1.8 + 32;
    humid = myDHT22.getHumidity();
    break;
  default:
    faren = 11.11;
    humid = 11.11;
    break;
}
...
Serial.print(faren);
Serial.print(humid);

```

Figure 4.15: RHT03 switch to read values

The RHT03 sends data in the form of 5 sets of 8 bit values. The first 16 bits correspond to relative humidity, the next 16 bits correspond with temperature values, while the final 8 bits are the checksum of the sensor. The switch in Figure 4.15 looks at the data read using the DHT22 library. In the case where all 40 bits are read correctly, the humidity and temperature values are stored in the respective variable. If an error is to occur, normally a message would be displayed to explain the error. In this sketch, to simplify the user interface when displaying the values, the temperature and humidity values are fed a false value of the same significant figures of 11.11. This value would be displayed if there was an error with the sensor, thus still allowing the user to know an error occurred. This simplified the coding of the visual interface by having only one type of data being read instead of the need for multiple data types and multiple conversions.

In the case of the Arduino sketch, the temperature value was initially read as Celsius and then converted in the sketch. The default unit in the DHT22 library is in Celsius but there is also a Fahrenheit conversion built in. This built in function performs the same conversion while utilizing the library. For the current sketch, the conversion was kept in the primary sketch and the *'getFahrenheit'* function was not used.

#### 4.1.6 BAROMETRIC PRESSURE

Barometric Pressure Sensor BMP180 was purchased to read the barometric pressure for each sensor package.

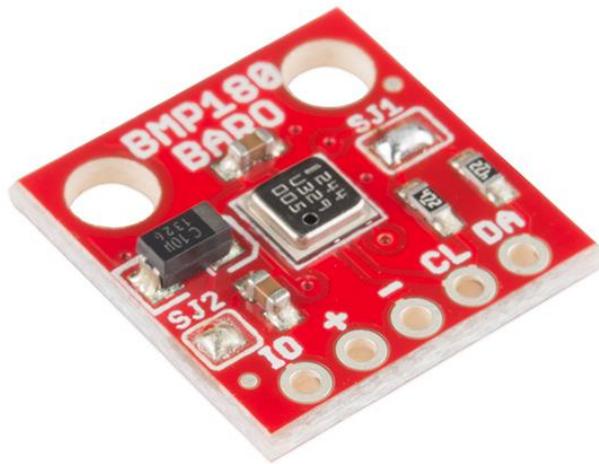


Figure 4.16: BMP180 barometric pressure sensor

The BMP180 is a high-precision, low power digital barometer with a range of 300 to 1100 hPa and an advertised accuracy of 0.02 hPa. The BMP180 can be powered using 1.8 to 3.6 V. For the sensor package, the BMP180 was powered using 3.3 V from the Arduino unit.

Similar to the RHT03, the BMP180 also has a library system ready to integrate with Arduino projects. The library allows for easy access and debugging of the BMP180. When using the BMP180, the library is called by the sketch and sensor initialized during setup (Figure 4.17).

```
#include <SFE_BMP180.h>
...
void setup()
{
  Serial.begin(9600);
  pressure.begin();
}
```

Figure 4.17: BMP180 setup

While in the main loop of the sketch, the status of the signal from the BMP180 is checked as it goes through the data from the sensor: such as initializing and getting the temperature, and initializing and obtaining the pressure.

```

void loop()
{
  status =
  pressure.startTemperature();
  if (status != 0)
  {
    delay(status);
    status =
  pressure.getTemperature(T);
  if (status != 0)
  {
    status =
  pressure.startPressure(3);
  if (status != 0)
  ...

```

Figure 4.18: Obtaining pressure

If any of these cases are not met, an error statement is normally displayed depending on when the sensor failed to obtain information.

```

...
}
else pressu = 11.11;
}
else pressu = 11.11;
...

```

Figure 4.19: BMP180 error code

As seen in Figure 4.19, if an error occurs, a pressure value of 11.11 is stored and later displayed. This is similar to error cases with the RHT03 to help in later displaying the data. When testing,

the pressure value would be given a string statement to display where the error occurred. For example, if the sensor failed to obtain a temperature value, a statement such as "error retrieving temperature measurement" would be displayed. If no errors occurred, the measured pressure is stored, converted from mb to inches Hg, and later displayed.

```
...
delay(status);
status = pressure.getPressure(P,T);
if (status != 0)
{
  pressu = P*0.0295333727;
}
...
Serial.print(pressu);
```

Figure 4.20: Storing, converting, and displaying pressure

#### 4.1.7 ARRANGEMENT

Once the parts were purchased, it was necessary to solder extension wires to specific pins to allow for connectivity between the Arduino microprocessor and the components. Most of the sensors would require a power and ground along with one or two data ports. The pressure sensor would require 3.3 V from the Arduino while being obtaining data from 2 data ports. The Humidity/Temperature sensor and Inspeed E-vane would both be powered by a 5 V port with each sensor using one data port each. The Inspeed Anemometer would only require a ground and PWM pin with a capacitor to account for bouncing signal.

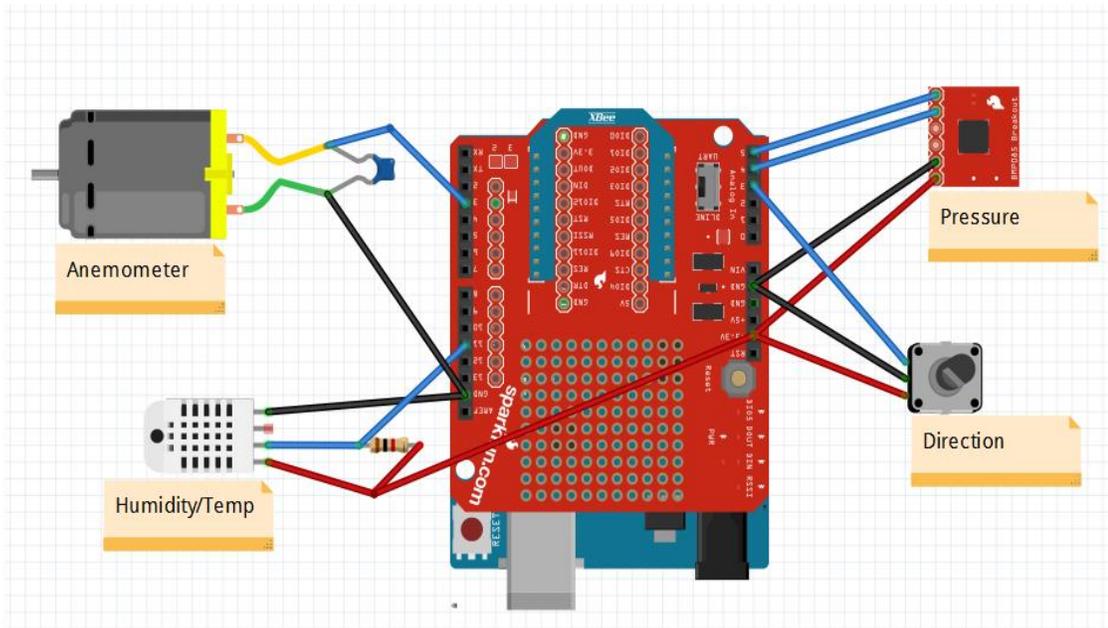


Figure 4.21: Sensor wire diagram

Once each sensor was individually wired, the humidity/temperature sensor, E-vane, and anemometer were each soldered to a JR female connector. This would allow for the sensor package to be more modular allowing for components to be easily exchanged or replaced. A wire mesh was constructed that consisted of pins for connectivity to the Arduino and then JR male connectors to serve as connection to the sensors to allow for multiple sensors to share the limited ground and power pins on the Arduino.

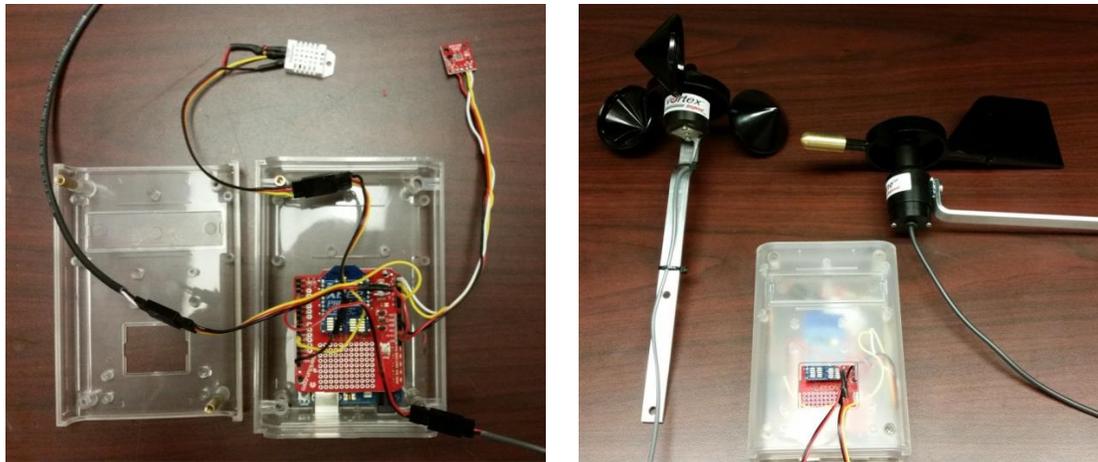


Figure 4.22: Sensor layout

To keep better track of the 3 constructed sensors, each package was labeled. Alpha, Bravo, and Charlie were the chosen designations for each of the sensors. Not only were these designations used to label the physical sensor packages, but the coding of the XBee radios and visual interface also mirrored the labeling. The coding of the visual interface discussed later will go into additional detail of the labeling of the sensors and data collection.

#### 4.1.8 COST

At the time of the sensor development, each of the mentioned components could easily be purchased from a number of online vendors. Other components, such as an Arduino enclosure and pin headers for the XBee Shield, were also purchased for each of the sensors. The total cost for an individual sensor was approximately \$279.15 (US) (Table 4.1).

Table 4.1: Individual sensor cost

Item	Quantity	Cost	Total
Inspeed E-Vane	1	\$99.00	\$99.00
Inspeed Vortex Anemometer	1	\$55.00	\$55.00
Barometric Pressure Sensor	1	\$9.95	\$9.95
Humidity/Temperature	1	\$9.95	\$9.95
Arduino Uno Board	1	\$24.95	\$24.95
Arduino Enclosure	1	\$7.95	\$7.95
Arduino Enclosure Extender	1	\$4.95	\$4.95
XBee S2 63mw	1	\$40.95	\$40.95
Arduino XBee S2 Shield	1	\$24.95	\$24.95
Arduino Header Kit	1	\$1.50	\$1.50
		<b>Total</b>	<b>\$279.15</b>

This cost includes a single sensor with all the necessary components minus wires, connector pins, and capacitors, to construct a single sensor unit.

In order to wirelessly read in the data from the sensors on a personal computer, one would need to buy an additional XBee radio with an XBee Explorer. This adds an additional \$65.90.

Table 4.2: Computer hub cost

Item	Quantity	Cost	Total
XBee S2 63mw	1	\$40.95	\$40.95
XBee S2 Explorer	1	\$24.95	\$24.95
		<b>Total</b>	<b>\$65.90</b>

All of the software used for programming the Arduino and creating the GUI for display and storage is freeware that can be downloaded and used. For this project, 3 sensors were constructed along

with a single computer hub. The total cost to purchase this was \$903.35 without taking into account shipping and handling fees.

Table 4.3: Total Cost for sensor package

Item	Quantity	Cost	Total
Inspeed E-Vane	3	\$99.00	\$297.00
Inspeed Vortex Anemometer	3	\$55.00	\$165.00
Barometric Pressure Sensor	3	\$9.95	\$29.85
Humidity/Temperature	3	\$9.95	\$29.85
Arduino Uno Board	3	\$24.95	\$74.85
Arduino Enclosure	3	\$7.95	\$23.85
Arduino Enclosure Extender	3	\$4.95	\$14.85
XBee S2 63mw	4	\$40.95	\$163.80
Arduino XBee S2 Shield	3	\$24.95	\$74.85
Arduino Header Kit	3	\$1.50	\$4.50
XBee S2 Explorer	1	\$24.95	\$24.95
		<b>Total</b>	<b>\$903.35</b>

## 4.2 TESTING

During the construction of the sensor package, testing was performed on each of the components independently during installation. Each component was added to an Arduino Sketch consecutively to allow each component and respective library to be tested and formatted. During this initial testing and building phase, the components were tested indoors and compared with basic, similar instruments for measuring the same phenomenon. Each sensor was independently connected to the Arduino Uno with a simple uploaded sketch to test the functionality of the component. For example, each of the three purchased RHT03 temperature and humidity sensors were sequentially connected to an Arduino Uno with a simple sketch that was designed to read the current room temperature and relative humidity level and display the values with relevant

units on the Arduino monitor on the connected personal computer. This was completed with all three RHT03 sensors to determine the validity of the library and sketch, along with the accuracy and precision of the three sensors. During this initial testing phase, the temperature readings were simply compared with an indoor digital thermometer. The other sensors were tested in a similar fashion, with a simple sketch being created or modified from the previous sensor test, to incorporate and test the necessary libraries and functions to call, convert, and output the sensor readings.

The RHT03 sensors are stated to be shipped already calibrated, yet these values were compared with indoor and outdoor temperature apparatus such as an indoor mercury thermometer, temperature sensor located at OSU flight field, and compared with the national weather's current outdoor temperature reading for the area: to help serve as a quick baseline. The BMP-150 pressure transducer was compare with a mercury barometer located in the OSU wind tunnel room.

To test the Xbee data transmission, the Arduino Uno was at first uploaded with a simple sketch to display random, sample data similar to that of the actual sensor. The output was visualized with both the Arduino monitor and other display software packages (described in more detail in future sections). This allowed to look for formatting options and to begin looking for optimal ways to view the transmitted data packet.

Once the sensors were individually tested and the final sensor package assembled, the sensors were taken to the OSU flight field for a complete field test. During initial field testing, the sensors were uploaded with the current version of the sensor Arduino sketch and placed around the flight field. The sensor packages were placed at different distances around the field, and were initially secured to fence posts or test benches. The purposes of these tests were to look for abnormalities in the data transmission and to note enhancements that should be made to the sensor code and the data interface.

During the initial field test, a few problems were found with multiple subsystems. First, there was found to be a line of sight issue with the Xbee transmitters. Even though the prescribed range of the sensor is 1 mile, the sensor could easily be obscured by a hill or building.

Second noticeable issue with the sensor packages, was an increase in the atmospheric temperature over time. There was also found to be discrepancy with the RHT03 sensors. When initially tested, the sensors were within a few degrees of the controlled temperature reading and would maintain this value. However, when placed outside, the temperature reading began to increase steadily and be off by 15 °F or more. This was found to be due to the components sensitivity with sunlight. During this testing, the sensor cases were mounted on local posts, with no protective covering from the sun. This caused a steady increase in the temperature reading. Once noticed, a single covering was placed on a single sensor to observe the effects of indirect sunlight on the sensor. The effect of this simple shade was shown to allow the temperature value to slowly return to an acceptable level. Based on this finding, it was necessary to construct a solar radiation shield for the sensors to allow for protection from the sun, while also allowing for ample airflow through the sensor (Figure 4.23).



Figure 4.23: Simple shading (left); Constructed solar radiation shielding (right)

The sensors were then tested to view temperature affects from having the sensor protected or not. All 3 sensors were used, each having a different shading technique. Charlie was set as the control with no additional protection from the sun, Bravo was setup with a single covering, while Alpha utilized the fully constructed radiation shield.

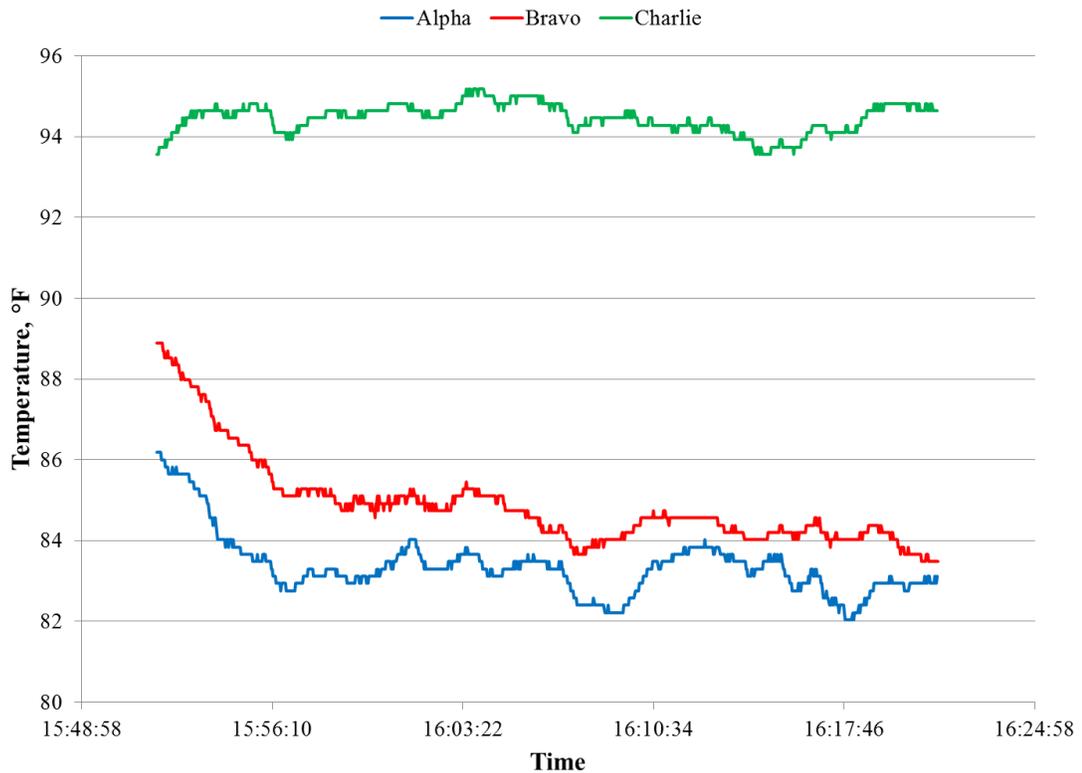


Figure 4.24: Sensor temperature comparisons

As can be seen in Figure 4.24, Charlie showed the highest temperature with no radiation protection. Alpha and Bravo showed lower temperature ranges closer to 83 °F and 85 °F respectively. A local temperature gage averaged about 84 °F during the 30 minute test, while online weather predicted a temperature around 78 °F for the area.

Finally, it was noticed that the wind speed was not consistent and would spike to unreasonable values. These values would be on the order between 2 to 600 mph, as can be seen in Figure 4.25.

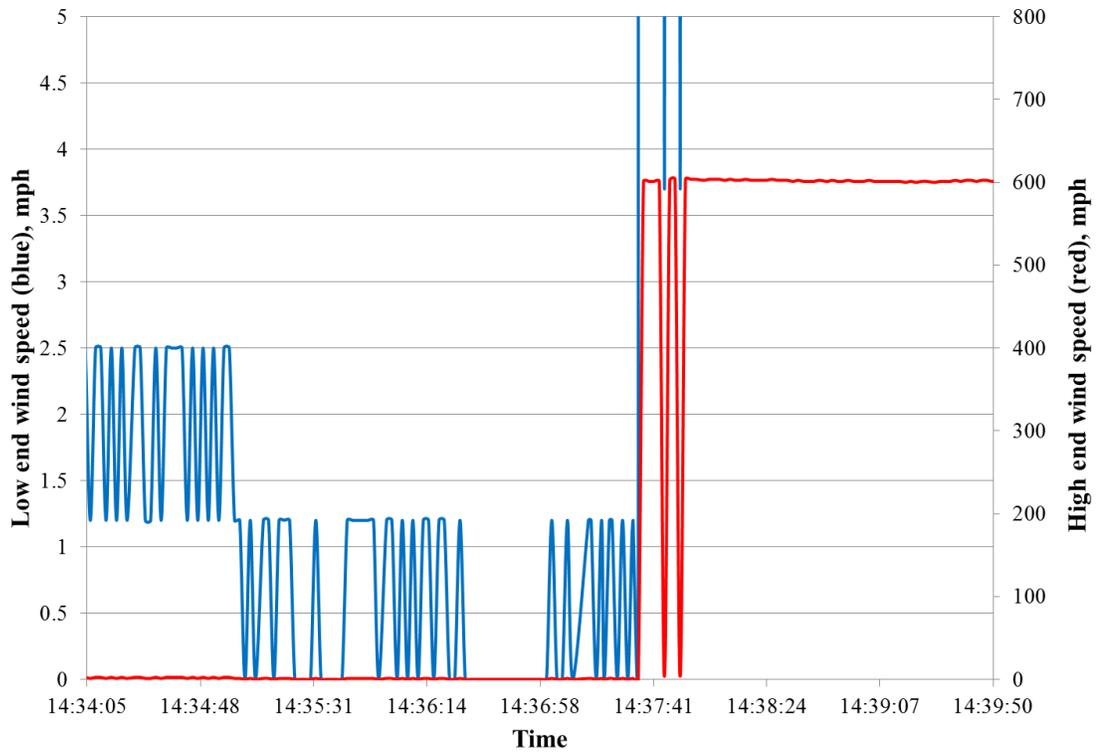


Figure 4.25: Initial wind speed values

This problem was initially thought to be due to a malfunction with the coding of the anemometer interrupt in the current Arduino Sketch. However, after further field tests and multiple revisions and tests of the current sketch, this was found to not be the case.

A single sensor unit was mounted in the OSU subsonic wind tunnel for additional testing. Data was first collected for current hardware and software configuration that was showing abnormally high wind speed values.

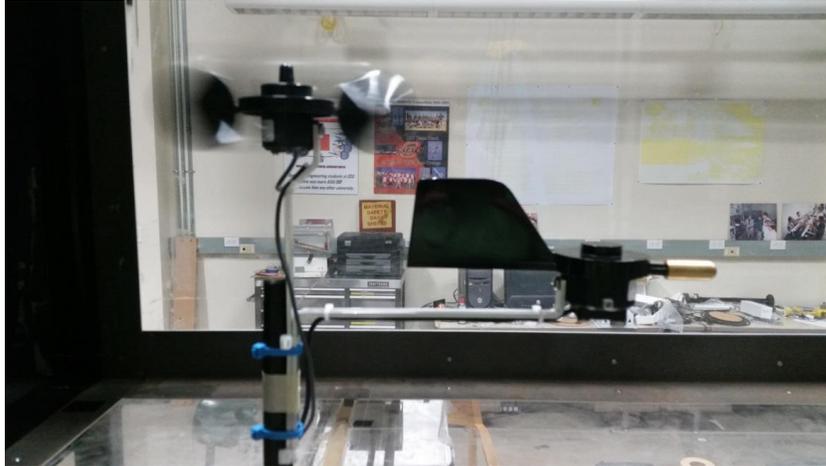


Figure 4.26: Wind tunnel testing

The anemometer pulse count and internally converted wind speed data was collected and averaged over 40 samples at intermittent wind tunnel dynamic pressure settings. Values were increased by 0.02 in  $H_2O$  between 0 and 0.2 in  $H_2O$  and then increased by 0.06 in  $H_2O$  from 0.2 to 0.4 in  $H_2O$ . Data was collected between 0.02 and 0.4 in  $H_2O$  which corresponds to a range between 6.4 and 28.5 mph. Data was collected for this range of tunnel speeds for different variations of the anemometer setup including adding internal and external resistors.

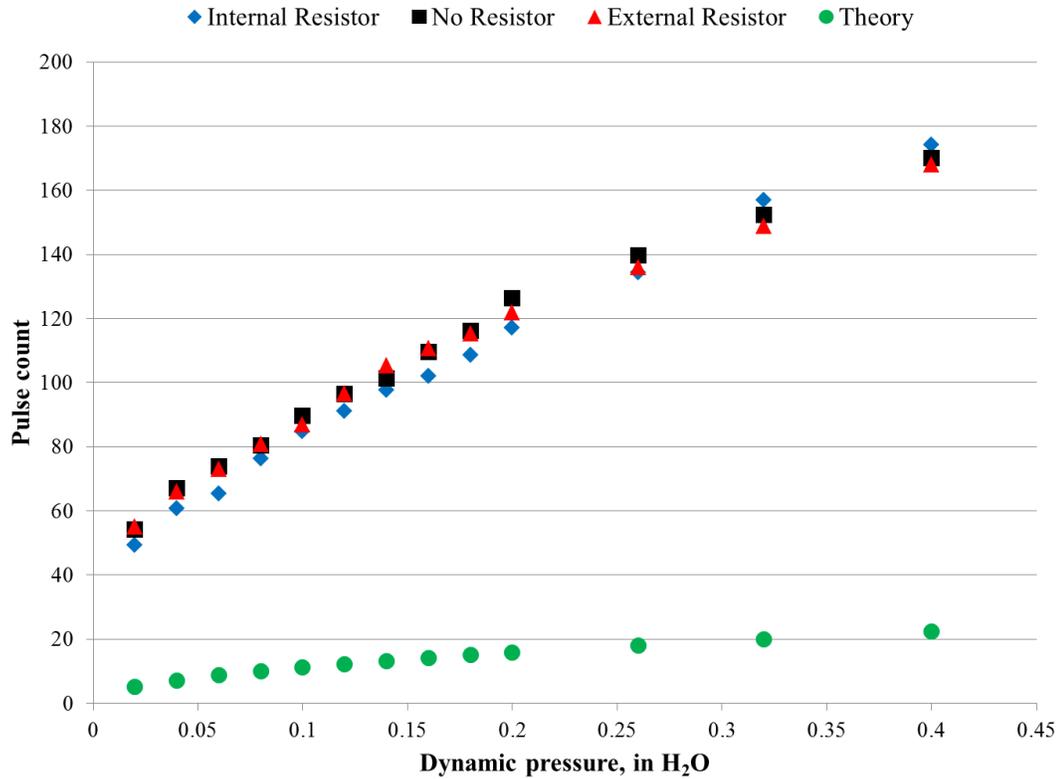


Figure 4.27: Anemometer initial testing results

Figure 4.27 gives the initial results. The “theory” data points were back-calculated based on the tunnel speed and Eq. 4.1 to estimate the number of rotations the anemometer should be experiencing.

After initial testing showed no improvement on the sensor’s ability to measure wind speed, an oscilloscope was connected to analyze the output signal from the anemometer to the Arduino. The signal was seen to have clear evidence of a pattern, but with noise (Figure 4.28).

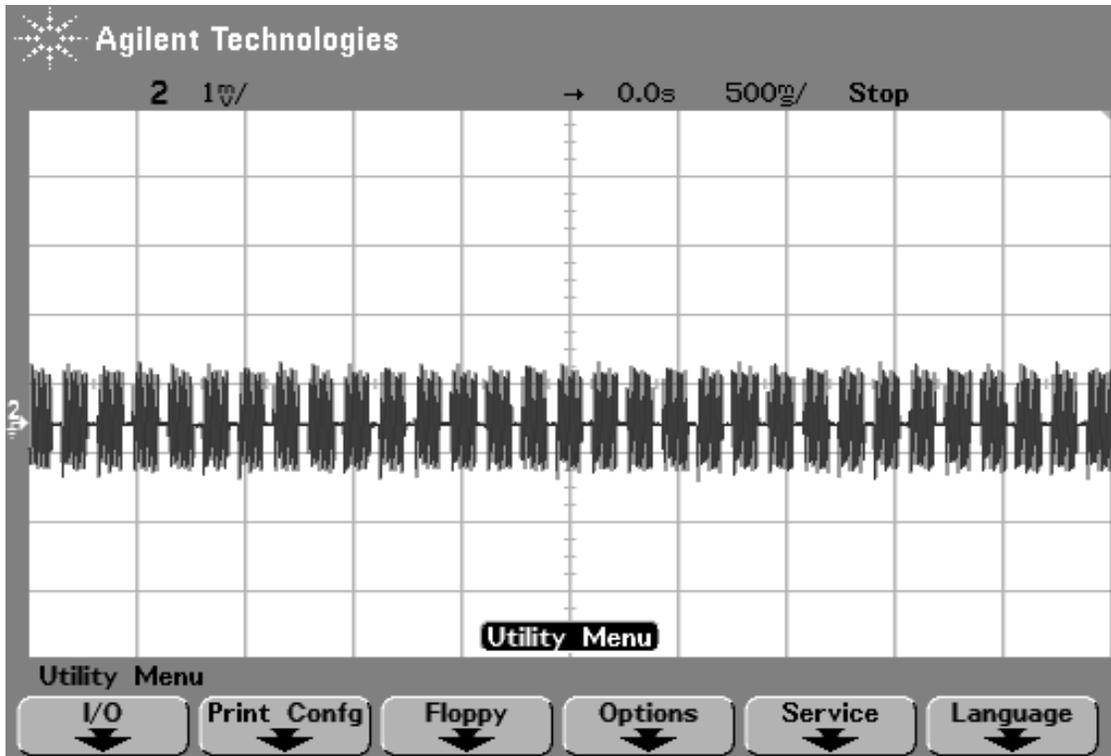


Figure 4.28: Initial anemometer oscilloscope readings

The fluctuations seen in Figure 4.28 were found to be due to debouncing problems within the reed switch of the anemometer. To fix the debouncing issues, a 100 nF capacitor was added to the anemometer. The resulting signal became clearer.

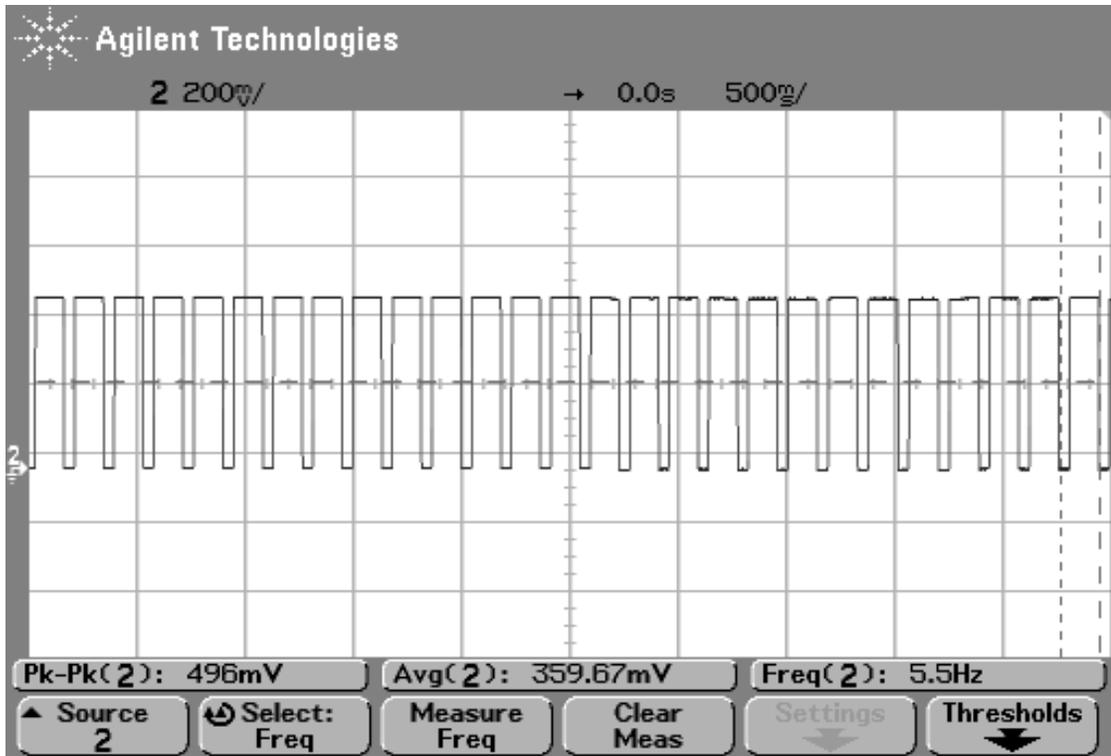


Figure 4.29: Oscilloscope readings with capacitor

Once the capacitor was added to the sensor setup, the sensor was placed in the wind tunnel once more to look at the outputs. The count method utilized by the Arduino began to show the correct corresponding speed values with the installed capacitor (Figure 4.30).

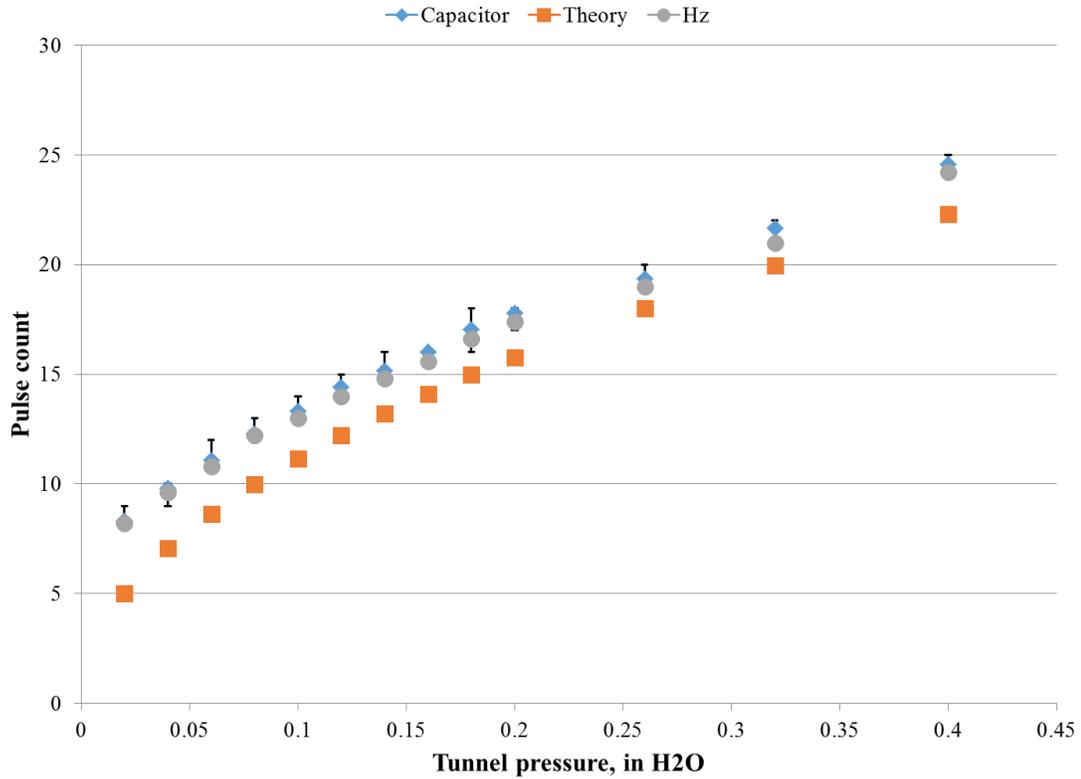


Figure 4.30: Anemometer results with added capacitor

For Figure 4.30, the Arduino output matched that with the oscilloscope frequency readings and only shifted slightly from the tunnel's default manometer.

### 4.3 SENSOR VISUALIZATION

Using over the counter electronic components based on the Arduino ATmega328 microcontroller, the ground sensors were designed to wirelessly transmit air data for ease of access and storage. The Arduino IDE is capable of displaying information from a connected Arduino unit or from an attached XBee module. However, data transmitted from the XBee system is encoded in hexadecimal. The Arduino IDE does not have the capability to convert the data transmission from hexadecimal to an easily understandable data type (such as ASCII) for

quick and easy analysis. Thus, a different viewing system was needed in order to be capable of easily accessing the data transmitted from the Arduino sensors.

37:38:2e:36:35:0D:0A:85	78.65
37:38:2E:36:32:0D:0A:85	78.62
37:38:2E:36:32:0D:0A:85	78.62
37:38:2E:36:32:0D:0A:85	78.62
37:38:2E:34:34:0D:0A:85	78.44

Figure 4.31: Arduino monitor hexadecimal display

The hexadecimal data package that is sent by the XBee radio is composed of multiple segments: a start delimiter, length bytes, API identifier, API frame ID, destination address, option byte, data packet, and checksum<sup>42</sup>.

Multiple systems were compared to determine the best method for accessing the transmitted data from the connected XBee module. These systems included Matlab, CoolTerm, and Processing. The new system needed to be capable of reading in data from the XBee module and explorer using a serial input. This GUI would be used to store and display the data from the individual sensors for quick reference of the air properties around the testing field during data collection. The GUI was also designed to incorporate additional algorithms and functions to detect or determine possible thermal updrafts. These additional functions are based on the previous simulations and adaptation of experimental result

#### 4.3.1 MATLAB SERIAL

Matlab is a high-level language and interactive environment for numerical computation, visualization, and programming developed by MathWorks<sup>43, 44</sup>. The tools and built-in math

functions can be used for a range of applications: such as signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology<sup>43, 44</sup>.

When programs were being considered for interfacing with the XBee network, Matlab programming skills were minimal for the author. Matlab has numerous help documents as well as a large database for users to share functions and programs. When investigating Matlab, only a handful of examples were found using the XBee protocols, of which most focused on the XBee/Arduino programming and not the Matlab portions. When a few Matlab functions were discovered, they were tested with some basic Arduino coding structure (such as sending a random integer every few seconds). Of these initial tests, none were successful. These tests were to utilize Matlab's ability to read in data through serial ports. However, during the initial phase of constructing an interface, Matlab was showing to be inadequate and due to time constraints, was not given any further consideration once a successful method was determined.

#### 4.3.2 COOLTERM

CoolTerm is an open source, free to use, serial terminal program created by Roger Meier<sup>42, 45</sup>. Capable of updating XBee firmware, CoolTerm also has built-in hexadecimal to ASCII view window to enable hexadecimal data to be viewed.

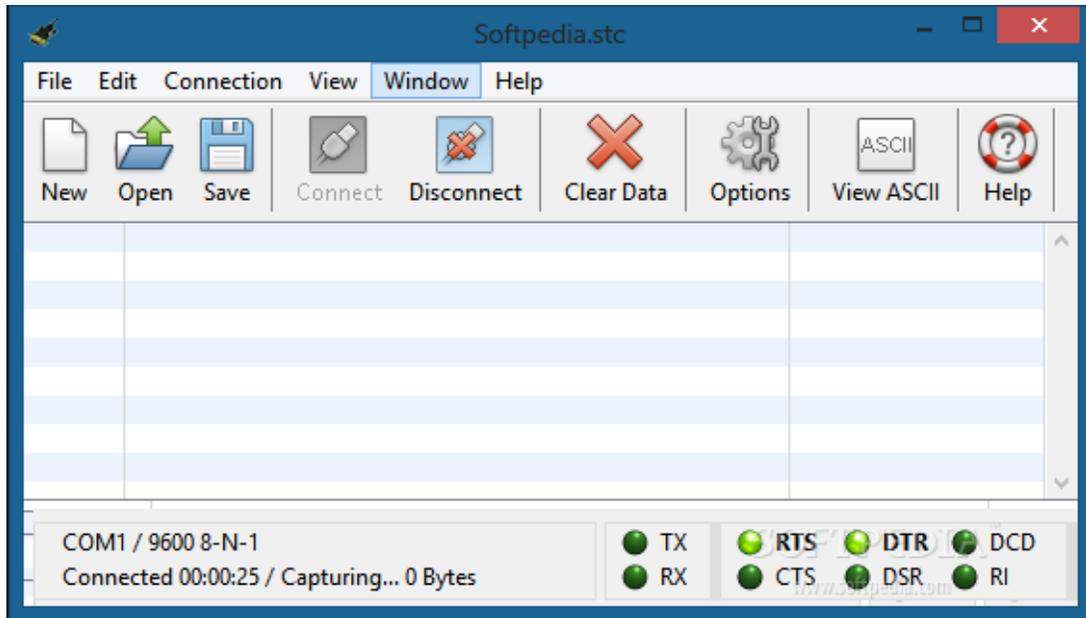


Figure 4.32: CoolTerm display

An initial test of the CoolTerm terminal was conducted by sending temperature values from an Arduino to the terminal using the XBee protocols. The terminal was successful in reading and displaying the data (Figure 4.33). The only data that was transmitted was the temperature data in degrees Fahrenheit.

```

~ }3 }3¢ @µV|¤$ 78.62
...~ }3 }3¢ @µV|¤$ 78.62
...

```

Figure 4.33: Example of CoolTerm output using temperature sensor

CoolTerm would read in the entire hexadecimal package and was capable of displaying the temperature data in ASCII format. However, additional packets were also being read and shown which would need to be eliminated to fully utilize the information that would be transmitted by 3 fully functional sensor packets. It was also noted during the initial tests that CoolTerm was also able to save the collected data in text format. This was used to confirm the information being displayed by the CoolTerm terminal.

CoolTerm was found to be capable of reading and converting the hexadecimal values from the Arduino system. However, CoolTerm is a terminal program and lacked the functionality to adequately display data being sent from multiple Arduino packages. For the data to be visualized would require an additional program outside of CoolTerm. Also, additional programs or functions would need to be utilized to convert saved data into a useful format since the terminal would save additional characters besides the payload.

### 4.3.3 PROCESSING

Processing is a programming language, development environment, and online community established in 2001. Initially created to serve as a software sketchbook to teach computer

programming fundamentals geared towards visual displays, Processing has evolved into a development tool for professionals<sup>42,46</sup>. Processing is an open source, free to download software package that allows users to create interactive programs for learning, prototyping, and production. Processing is capable of creating interactive programs with 2D, 3D, or PDF outputs. Processing has a large community following and is well documented through online and printed resources.

The Processing Development Environment (PDE) makes it easy to write Processing programs. Programs are written in the Text Editor and started by pressing the Run button. In Processing, a computer program is called a *sketch*. Sketches are stored in the *Sketchbook*, which is a folder on your computer. It's easy to open the sketches by clicking on the Open button.

Sketches can draw two- and three-dimensional graphics. The default renderer is for drawing two-dimensional graphics. The P3D renderer makes it possible to draw three-dimensional graphics, which includes controlling the camera, lighting, and materials. The P2D renderer is a fast, but less accurate renderer for drawing two-dimensional graphics. Both the P2D and P3D renderers are accelerated if your computer has an OpenGL compatible graphics card.

The capabilities of Processing are extended with *Libraries* and *Tools*. Libraries make it possible for sketches to do things beyond the *core* Processing code. There are hundreds of libraries contributed by the Processing community that can be added to your sketches to enable new things like playing sounds, doing computer vision, and working with advanced 3D geometry. Tools extend the PDE to help make creating sketches easier by providing interfaces for tasks like selecting colors.

Processing has different *programming modes* to make it possible to deploy sketches on different platforms and program in different ways. The current default programming modes are *Java* and *Experimental*. Other programming modes, such as *JavaScript* and *Android*, are added by selecting "Add Mode" from the menu in the upper-right corner of the PDE.

During initial testing of Processing, a short sketch was modified from an existing example to incorporate a simplified sensor package to wirelessly transmit data to the computer for

visual display<sup>42</sup>. For this example, Processing version 2.2.1 was used. The simplified sensor consisted of a RHT03 sensor with respective Arduino sketch preloaded. On the computer side, the Processing sketch was to wirelessly read in the temperature and relative humidity from the RHT03 and display the results along with the 64 bit address of the sensor. The original sketch was to read in just temperature. Since Processing is primarily used for visual displays, the data was shown as a thermometer with a moving red bar to represent the mercury. Once the Processing sketch was modified and XBee radios configured, the system was tested. The initial test of a single RHT03 sensor worked, thus showing the feasibility of Processing to read and display the required data. Once one sensor was demonstrated, a second RHT03 unit was added to the wireless network to examine the feasibility of communicating with multiple sensor units along with the functionality of object and class functions (Figure 4.34)

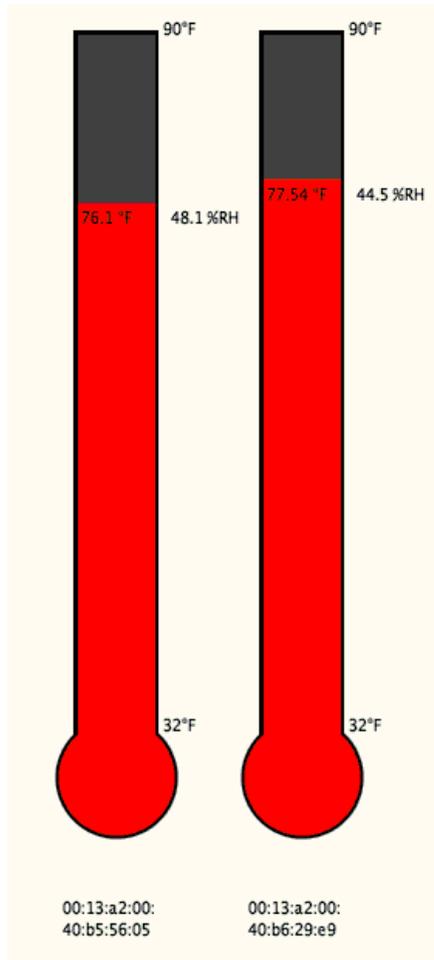


Figure 4.34 Sample Processing sketch for testing

The display shows the temperature, relative humidity, as well as the 16 bit address of the respective XBee radio. This Processing sketch would later serve as an example and the foundation of the visual display that was to be created for the final sensor packages.

In the end Processing was chosen due to its simplicity with XBee integration and for the boundless possibilities with representing the data visually. The amount of new information and programming skills that would need to be utilized was comparable between incorporating Processing and Matlab, with both programs having a massive advantage with data representation over CoolTerm. Using CoolTerm would still require an additional program to represent and

display the data which would add additional complexity to the system. Processing allows for a simplistic, single interface, to interpret the sensor data.

#### 4.4 GUI DESIGN

Experimental data was to be primarily collected around the OSU Flight Field, in Payne County, OK (36°09'43.5"N 96°50'08.3"W) where thermals were known to form. Data would be collected from 3 separate sensors at 2 second intervals. A local computer would be needed to record and display current data from these sensors. A graphical user interface (GUI) was developed using Processing to read data from the Arduino based sensors and display the information.

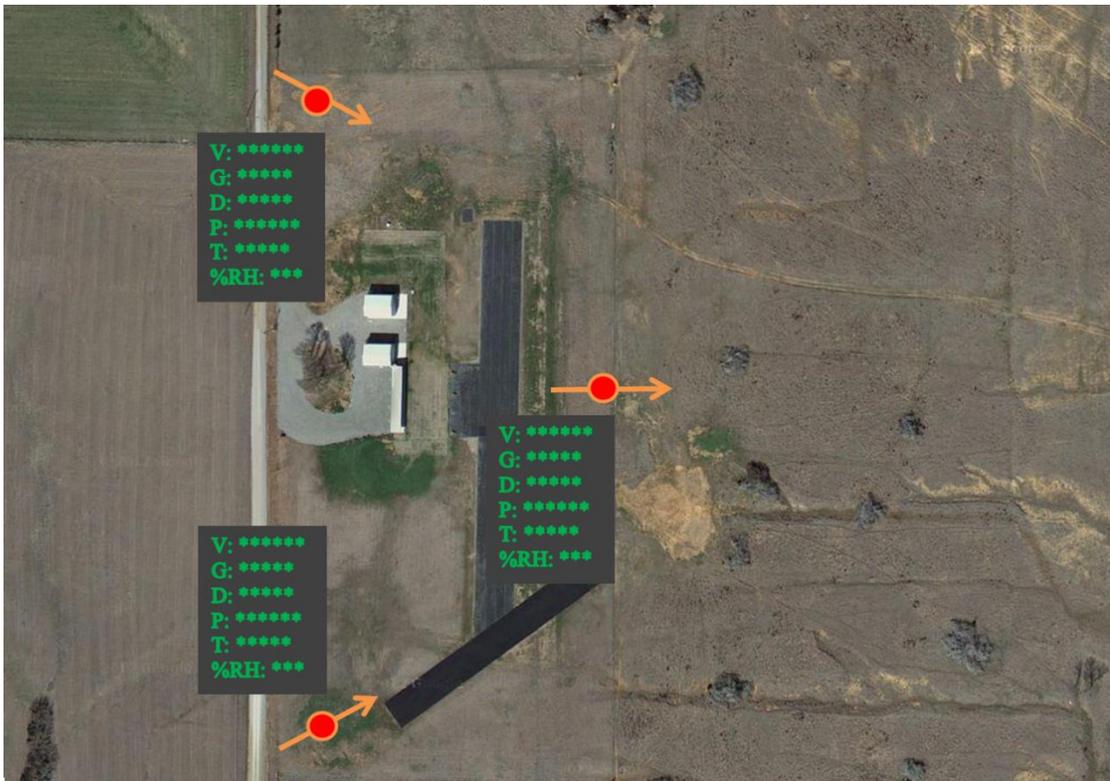


Figure 4.35: Initial GUI concept for ground sensor interface

This GUI would display the data from the individual sensors for quick reference of the air properties around the testing field during data collection. The GUI was also designed to incorporate additional algorithms and functions to detect or determine possible thermal updrafts. These additional functions would be based on previous simulations as discussed in Chapter 3, along with adaptation of experimental results.

The GUI was to have 3 primary responsibilities: read data from the XBee radios used on the remote sensors, store the information, and visually represent the data for real-time analysis. The following subsections will go into greater detail on the coding involved to accomplish these 3 tasks. Note that some of the coding structure and algorithms used primarily to initialize Processing and call minor subroutines will not be discussed. Only the major functions and concepts will be discussed. Additional information about coding in Processing can be found online or through printed references.

#### 4.4.1 READ XBEE DATA

Reading data from the XBee radios was the first task for the GUI to accomplish. In order to read the data, an installed XBee library was used. In order to use the XBee library, the library and applications were imported into the Processing directory and called for initialization.

```
import processing.serial.*;
import com.rapplogic.xbee.api.ApiId;
import com.rapplogic.xbee.api.PacketListener;
import com.rapplogic.xbee.api.XBee;
import com.rapplogic.xbee.api.XBeeResponse;
...
String version = "1.3";
String mySerialPort = "COM5";
...
XBee xbee = new XBee();
```

Figure 4.36: GUI XBee library import and initialization

For the GUI, version 2.2.1 of Processing was once again utilized. It should be noted, that due to changes in versions, certain libraries and functions were no longer available with version 2.2.1 that are needed by the Processing XBee library: specifically the serial communication files. Processing 2.2.1 uses a different set of serial communication libraries, and thus the old libraries (those the XBee library associates with) must physically be added to the Processing sketch's library. Adding the corresponding files to the 'code' subfolder of the sketch, along within the sketch, will allow the XBee library to function properly. Additional features in version 2.2.1 versus older versions, such as the 'table' function (which will be discussed in a later section), made it useful to stay with updated versions.

Once the XBee library was initialized, the data packet from the XBee radio could be gathered. The next task for the GUI would be to determine if there is any information waiting to be read by the XBee coordinator. The `XBeeResponse()` function accomplishes this task by waiting and looking for incoming data packets to be used (Figure 4.37).

```
SensorData getData()
{
  SensorData data = new SensorData();
  int value = -1;
  String address = "";
  try
  {
    XBeeResponse response = xbee.getResponse(3000);
    println("Received response " + response.toString());
    ...
    if (response.getApiId() == ApiId.ZNET_RX_RESPONSE &&!response.isError())
    {
      ZNetRxResponse dataSample = (ZNetRxResponse)(XBeeResponse) response;
    }
    ...
  }
}
```

Figure 4.37: GUI check for incoming data

As mentioned previously, the information sent via the XBee radio is a hexadecimal value that is comprised of start delimiter, length bytes, API identifier, API frame ID, destination address, option byte, data packet, and checksum. The Processing XBee library has numerous functions that can be used to gather specific parts of the hexadecimal packet. For the sensor GUI, the `getAddress()` and `getData()` functions were used to grab only the XBee address, along with the data packet or payload respectively. The `getAddress()` function would grab the 64 bit address of the sensor's XBee radio that was currently transmitting data to the coordinator. This was necessary to keep track of where the data was coming from and to correctly display the information.

```
int[] addressArray = dataSample.getRemoteAddress64().getAddress();
String[] hexAddress = new String[addressArray.length];
for (int i=0; i<addressArray.length;i++)
{
    hexAddress[i] = String.format("%02x", addressArray[i]);
    String senderAddress = join(hexAddress, ":");
    if (senderAddress.equals("00:13:a2:00:40:b5:56:05"))
    {
        data.address = "Alpha";
    } else if (senderAddress.equals("00:13:a2:00:40:b6:29:e9"))
    {
        data.address = "Bravo";
    } else if (senderAddress.equals("00:13:a2:00:40:b5:56:09"))
    {
        data.address = "Charlie";
    } else {
        data.address = senderAddress;
    }
}
...
```

Figure 4.38: GUI code to read sensor address

The 64 bit, hexadecimal address was then converted to a string classification. This was the same classification used to describe and name the sensors: Alpha, Bravo, and Charlie. Since the 64 bit address is constant and unique for each XBee radio, the function in Figure 4.38 was hardcoded with the known hexadecimal addresses to reassign the sensor address with a corresponding sensor name. This new string classification would be used to keep track of the data and location when displayed and when used in later analysis.

The `getData()` function was responsible for gathering only the information outsourced from the sensors: temperature, relative humidity, pressure, wind speed, and wind direction. Once the payload was read by the GUI, the payload was partitioned into its individual components and converted from hexadecimal to ASCII (Figure 4.39).

```
int[] payload = dataSample.getData();
StringBuilder outputT = new StringBuilder();
for (int i=0; i<(payload.length-1); i++)
{
    String strComb = String.format("%x", payload[i]);
    outputT.append((char)Integer.parseInt(strComb, 16));
}
String theDataPacket = outputT.toString();
String[] DataParts = splitTokens(theDataPacket, ":");
try
{
    data.temp = Float.parseFloat(DataParts[0]);
}
catch (NumberFormatException e)
{
    data.temp = 40;
}
```

Figure 4.39: GUI code to read and convert packet (only temperature shown)

With the partitioning and conversion, the GUI was also programed to catch read errors. If data was corrupted or unreadable by the GUI, a dummy value was stored in the variable to allow continuous display and storage, but to also allow the user to be able to easily spot errors in the data. For example, in Figure 4.39, if the temperature value was unreadable, a value of 40 °F was used. At the times of using this code and sensor package, the average temperature was above 70 °F. This value could easily be exchanged for a value more easily viewed by the operator.

```
else if (!response.isError())
{
    println("Got error in data frame");
}
else
{
    println("Got non-i/o data frame");
}
}
catch (XBeeException e)
{
    println("Error receiving response: " + e);
}
```

Figure 4.40: GUI error exceptions

#### 4.4.2 STORE XBEE DATA

Another important part of the GUI was to not only visualize the information but to also store the sensor data for later usage. Processing 2.0 and later versions, incorporated a `table()` function where rows of information could be added by partitioned column references. This is an additional reason why version 2.2.1 was chosen over previous versions even with having to manually add the serial commands incorporated with the XBee library. This table feature allows

for easier access to the data via spreadsheet applications, such as Microsoft Excel, as well as allowing for easier storing of data.

```
table = new Table();
table.addColumn("Time");
table.addColumn("Name");
table.addColumn("Temperature");
table.addColumn("Humidity");
table.addColumn("Pressure");
table.addColumn("Speed");
table.addColumn("Angle");
...
saveAs(sensorName, temperatureFar, relativeHum,
pressure, windSpeed, direction);
```

Figure 4.41: GUI code to initialize saving of data

Figure 4.41 illustrates how the table is created, by naming the requested columns which can be called upon later. This allows for instead of having to combine the data to a single string and save it for later partitioning, Processing allows for the data to be saved under a called column.

```

void saveAs(String name, float temp, float humidity, float pressure,
float speed, float direction)
{
    TableRow newRow = table.addRow();
    newRow.setString("Time", hour()+":"+minute()+":"+second());
    newRow.setString("Name", name);
    newRow.setFloat("Temperature", temp);
    newRow.setFloat("Humidity", humidity);
    newRow.setFloat("Pressure", pressure);
    newRow.setFloat("Speed", speed);
    newRow.setFloat("Angle", (direction));
    saveTable(table, "data/Sensor_Data.csv");
}

```

Figure 4.42: GUI code of saving data within columns

#### 4.4.3 VISUALIZE XBEE DATA

In the same way as RC and glider pilots use ground flags to spot shifts and changes in wind speed and direction, these sensors were to be used to indicate thermal updrafts for either verification of local thermals or possible incorporation with UAV autopilots for thermal indication. Based on the “third” vector approach, the interface would measure the local wind conditions, and through the use of equations and similar methods used for simulations discussed in Chapter 3, the thermal vector would be calculated and displayed.

To visualize the data, the GUI was designed to give the user a top-down view of the surrounding area that contained the sensors while displaying the current air values at each location. Using Processing, a background was created using an image of the surrounding area, in this case the OSU Flight Field, which would serve as the map of the local area being surveyed by the sensor network.

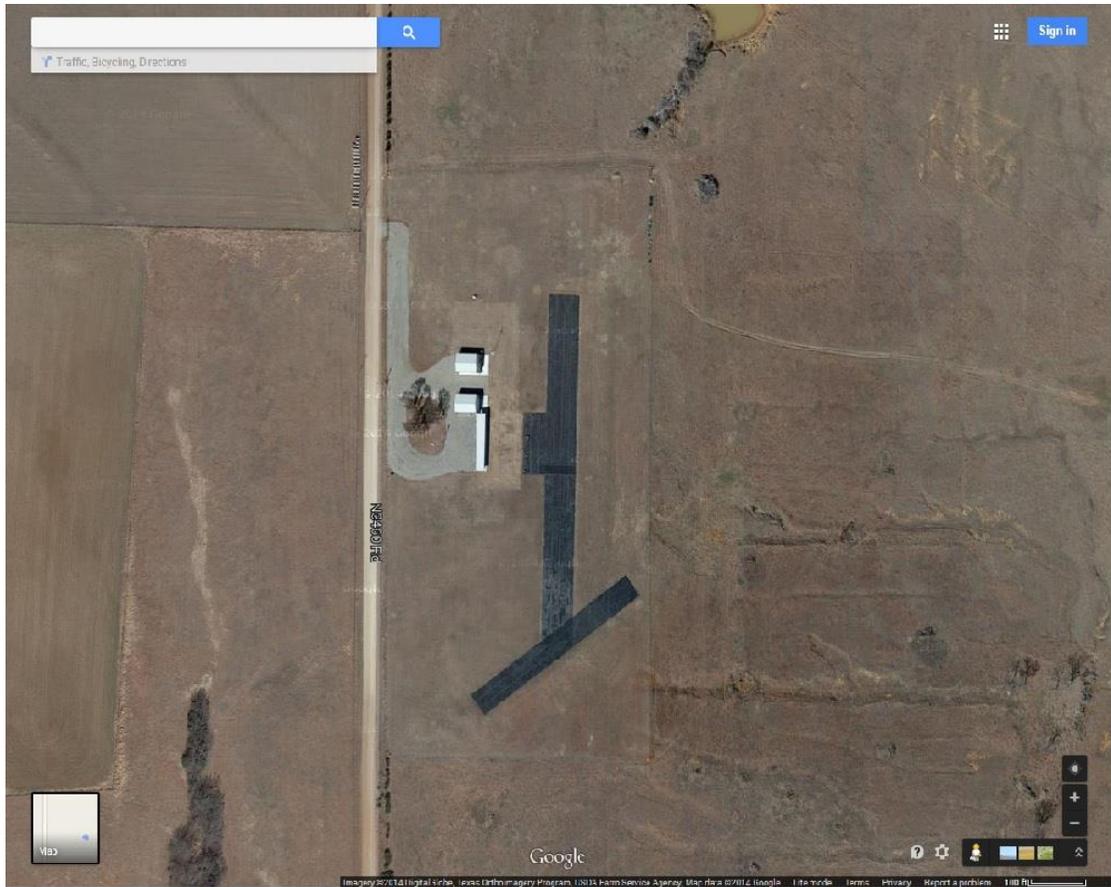


Figure 4.43: Primary background image (1000x800 pixels)

The background is loaded during the setup subroutine of the GUI and is utilized through every loop of the GUI, as other features will overlay on top of the background and need to be reset to allow updated information to be displayed. Features that would be overlaid on the background involve raw data from the sensors, local time stamp, and thermal direction indicators.

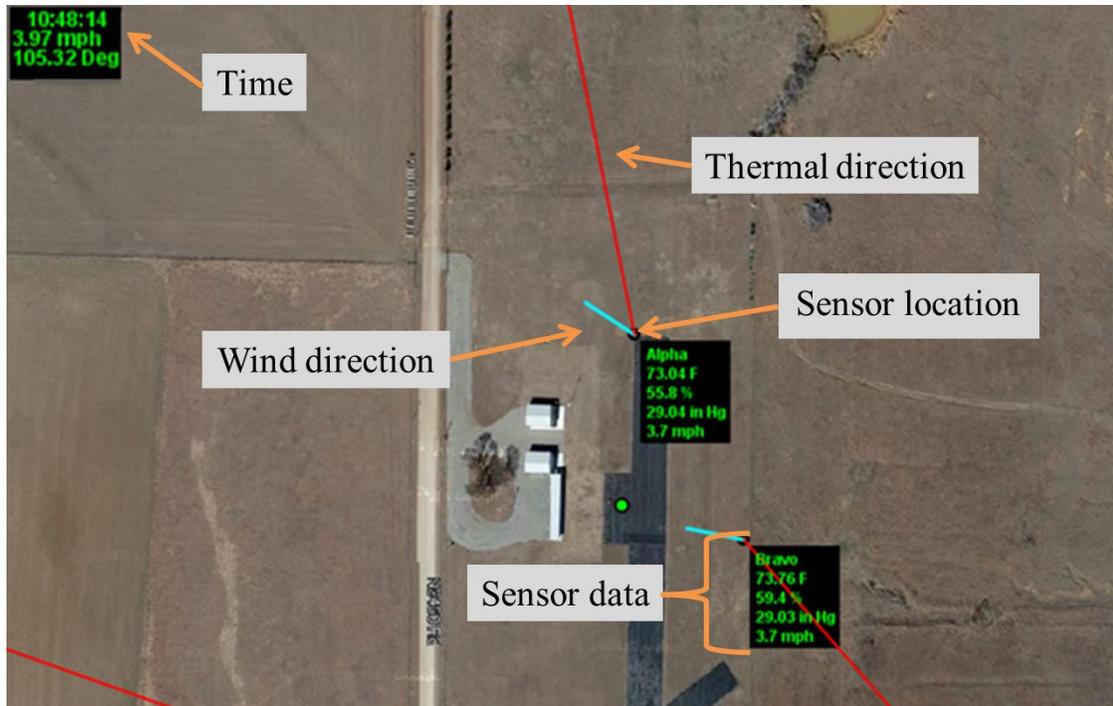


Figure 4.44: GUI overlay

Each sensor was represented at their respective location, with raw data being displayed at each location. Displayed data includes: current wind speed, pressure, temperature, and relative humidity. Along with the raw data, the time is displayed via a text box in the top left-hand corner of the interface.

Since the sensors were not developed with active GPS components at the time of this research endeavor, the sensor locations were manually entered into the GUI for display. The raw data was read from processing and then displayed next to the specified location of the corresponding sensor identifier. This was accomplished through Processing's text box functions.

```

...
// text box
textAlign(LEFT);
textSize(12);
fill(0,252,0);
text(address, posX+10, posY+20);
text((temp)+" F", posX+10, posY+35);
text((humid)+" %", posX+10, posY+50);
text((press) + " in Hg", posX+10, posY+65);
text((spd) + " mph", posX+10, posY+80);
...

```

Figure 4.45: Code for text

To indicate the wind direction, however, instead of utilizing a numeric value, the GUI would draw a line or vector to indicate direction at that sensor, similar to that of a physical wind sock. Not only would this line show the direction, the length of the vector was programmed to correspond to the measured wind speed.

```

// Wind vector
pushMatrix();
  translate(posX, posY);
  rotate(radians(dir+90));
  strokeWeight(3);
  stroke(15,237,255);
  line(0, 0, spd*10, 0);
popMatrix();

```

Figure 4.46: Code for wind vector

The thermal direction indicators were drawn on the interface based on the calculated thermal direction (Eq. 3.21). The angle of the calculated thermal direction was used to plot a

thin, red line at the specified angle. Since only the angle is calculated, the line is drawn with a length of 2 times the width of the background. This was to insure the indicators would reach the interface boundaries.

```
// Thermal vector
pushMatrix(); // push this section out of the current loop
  ThermalDir = degrees(atan2((FAvgy/AvgF)-(AvgDiry/AvgN),
    (FAvgx/AvgF)-(AvgDirx/AvgN)));
  translate(posX, posY);
  rotate(radians(ThermalDir+90));
  strokeWeight(1);
  stroke(0);
  line(0,0,width,0);
popMatrix();
```

Figure 4.47: code used to draw thermal indicators

The final version of the processing code, which incorporates all of the functionality discussed in previous sections, as well as the visual cues discussed in this section can be found in Appendix 7.2.



Figure 4.48: Final version of GUI at the time of this document

## CHAPTER V

### 5 RESULTS

The following chapter will describe additional results that were acquired and examined during the course of this research endeavor. Using a combination of computer modeling and field experiments, data was collected on the effects that thermal updrafts had on the ground level sensors which would help draft parallels with what could be felt by local aircraft while in flight. Some of the initial and idealized simulation results have been discussed in Chapter 3 with the simulations discussed in this chapter being composed of additional simulations that were used to finalize Objective 4.

Experimental data consists of data recorded from the developed ground sensors, ground observations, and aircraft performance. The ground sensors were used to collect data on air properties around the air field while aircraft(s) were in search of thermal updrafts, as well as observations being made of local avian species utilizing local updrafts. The data was then examined during the time frames of known thermal activity to show correlations between forming thermals and the surrounding air properties. The results discussed here represent only a portion of the data that was collected, but reflect the overall consensus of the sum total of the results.

## 5.1 EXPERIMENTAL APPROACH

Combining information from past resources, either literature or simulations, allowed for an experimental approach to be taken. From the literature survey in Chapter 2, information was pieced together to find 5 different hypotheses regarding avian species ability to remotely detect thermal updrafts. From these hypotheses, along with anecdotal evidence from RC methods of thermal detection, the method of detecting local shifts or the “third vector” approach was chosen to be tested. Initially, simulations were created to examine the possibility of calculating these shifts in wind speed and direction using idealized thermal components: primarily using a sink profile with given sink strength to simulate the thermal inflow. From these initial simulations, it was seen to be feasible to use remote locations to triangulate a source of inflow. With each location keeping a record of individual averaging and instantaneous wind vectors, calculating the shift direction was feasible.

Taking the simulations to the next step was to physically measure atmospheric conditions in a local area to calculate actual wind shifts. With the sensors and interface developed in Chapter 4, raw wind data could be collected for analysis of surrounding thermal updraft activity. Thermal updrafts would be substantiated using local avian species observations (soaring species circling with minimum flapping motion with a constant or increase in altitude) and/or with local RC aircraft searching and utilizing nearby updrafts. If the ground based sensor platforms were found to be capable of triangulating thermal updrafts, additional experimentation involving localized UAVs equipped with onboard sensors would be used in parallel with the sensors to evaluate the changes in wind shifts while in flight with regards to nearby thermal activity. Based on literature and interviews with regards to RC utilization of thermals, the inflow from thermals are capable of easily being felt or have noticed effects between 5 and 20 ft above the ground. The ground based sensors used were initially placed at 5 ft and then moved to 9 ft for later collection dates. This height was regarded as being above the roughness layer for the inflow from the thermal<sup>1</sup>. Additional information regarding the actual thickness of the thermal inflow was

minimal, which was another reason for using instrumental UAVs for measurement and comparison.

## 5.2 BURN RESULTS

On April 20<sup>th</sup>, 2015, the OSU Natural Resource Ecology and Management department performed a control burn approximately 11 miles west of Stillwater, OK ( $36^{\circ}04'07.3''N$   $97^{\circ}12'41.2''W$ ). During this burn, the Natural Resource Ecology and Management team set fire to approximately 375 acres. On this date, the 3 ground sensor packages were set up before the burn at one of the corners of the fields that were to be burned (Figure 5.1) in the hopes of gaining additional information and insight into convective updrafts.



Figure 5.1: April 20th control burn location and approximate sensor placement

The distances between each of the sensors were approximately 645 ft between Alpha and Bravo, 420 ft between Alpha and Charlie, and 510 ft between Bravo and Charlie.

The burn was to aid the experimental data collection as a "controlled" experiment where the controlled burn would serve as a large, stationary heat source causing updrafts to occur over the field. The data gathered from the sensors during this time would allow for areas with known shifts to be analyzed after the burn took place. It was thought that either a majority of any detectable shifts would point toward the controlled burn, or that the inflow would be strong enough to have all sensors point inwards, toward the controlled burn. During this burn, data was collected over 1.5 hours with samples collected every 2 seconds for over 2600 total samples collected for each sensor.

Along with collecting data from the individual sensor packages, weather data was also obtained from online weather sources such as usairnet.com in order to receive a baseline for the local air conditions. Other sources of air data were obtained from the Oklahoma Mesonet system. The Oklahoma Mesonet consists of 120 automated stations covering Oklahoma, with at least one station in each of Oklahoma's 77 counties<sup>47</sup>. Air properties are measured by a set of instruments located on or near a 10-meter-tall tower with weather observations sent every 5 minutes to a central facility 24 hours per day year-round. During the burn on April 20<sup>th</sup>, 2015, data from one of the local Oklahoma Mesonet stations was acquired for comparison with the ground sensor stations. The location of station Marena (36°3'51"N, 97°12'45"W) was roughly 1600 ft from the location of placed sensor packages during the burn (Figure 5.2). The measured wind speed and direction was used for comparison.



Figure 5.2: Mesonet station, Marena, near controlled burn

To better understand the shifts on the prevailing winds due to the controlled burn, the changes in wind speed and direction were inspected first. Figure 5.3 shows a rose plot, or circular histogram, of the prevailing wind directions for 2 of the 3 sensors (Alpha and Bravo) during the burn.

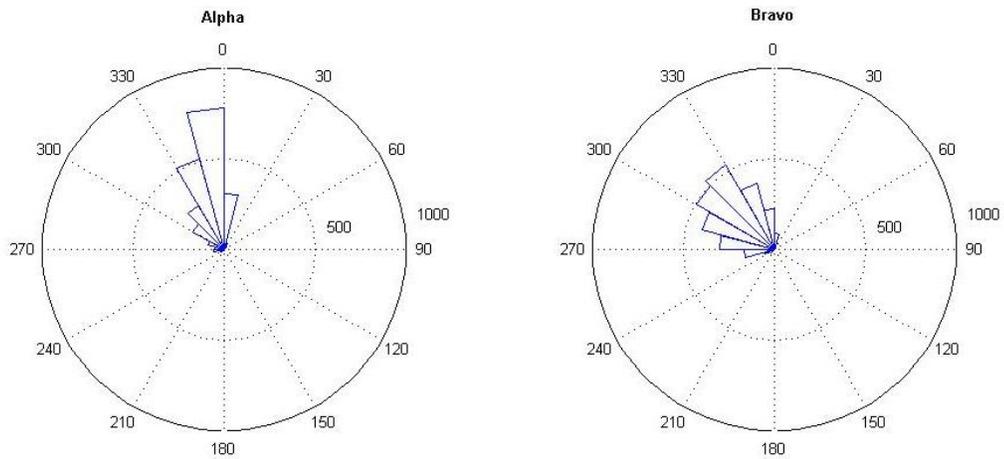


Figure 5.3: Directional rose plots for April 20th burn

The plots in Figure 5.3 show the frequency of the wind directions that were being read by the corresponding ground sensors over the 1.5 hours of data collection. A malfunction occurred with Charlie during this burn as the sensor was not reading wind direction. All other information (pressure, speed, temperature, and relative humidity) were still being displayed and stored. It was later determined that during one of the previous days of data collection, a gust of wind had knocked over Charlie which caused the E-Vane's frictionless, Hall Effect sensor to become detached. The data that was collected from Charlie will be discussed where appropriate.

With speed being a factor, the frequency of the wind speed was overlaid on the histogram to determine the wind speed changes and quantities that occurred and at what direction during the controlled burn.

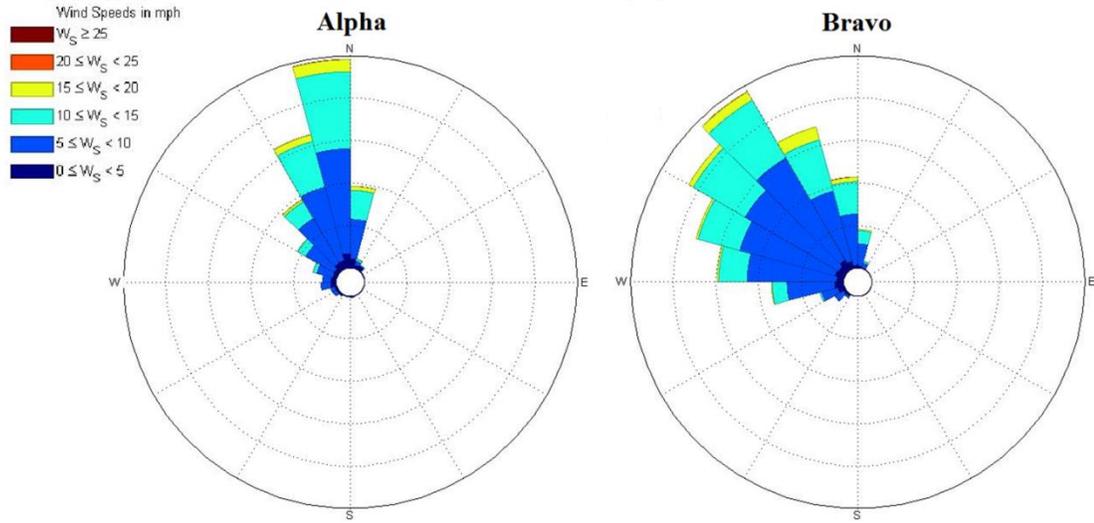


Figure 5.4: Wind rose plots for controlled burn

These new histograms illustrate that a majority of the collected wind speed data was between 5 and 15 mph. When looking at the raw wind directional data as compared to the Mesonet station, Figure 5.5 shows the scattered wind direction that was measured by the placed sensors.

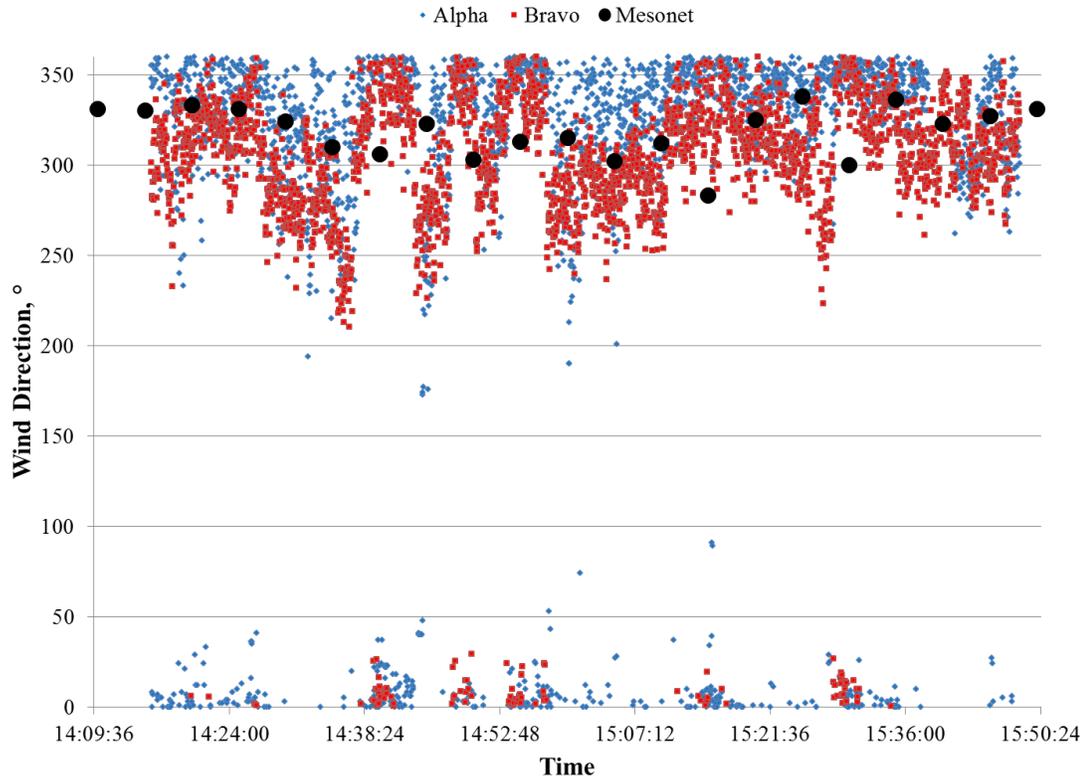


Figure 5.5: Wind direction for controlled burn

From the data collected during the 1.5 hours, the prevailing wind was calculated from the averages from the 2 working sensors. According to an aviation weather forecast for that day for Stillwater, OK, the prevailing wind was around 330° Azimuth, with an average speed of 12 mph. Data from the Mesonet station was also averaged during this time frame to examine the similarity and constancy of the collected data.

Table 5.1: Wind averages from different sources

Source	Speed MPH	Unweighed Average	Weighted Direction
Online	12	330	330
Mesonet	11	317	318
Sensor Alpha	8	337	339
Sensor Bravo	8	310	313

The unweighted averages were calculated using only the measured angle, while the weighted averages involved the wind speed with the respective speed value associated with it. All 4 sets of averages are seen to be similar in comparison with regards to wind direction, with only a few degrees difference between them. However, there is a 3 to 4 mph difference between the sensors with the Mesonet data and online source respectively. By looking at the averages on the histogram, Figure 5.6 shows the average direction that the wind acted on the individual sensors with both a weighted and non-weighted calculation taking place.

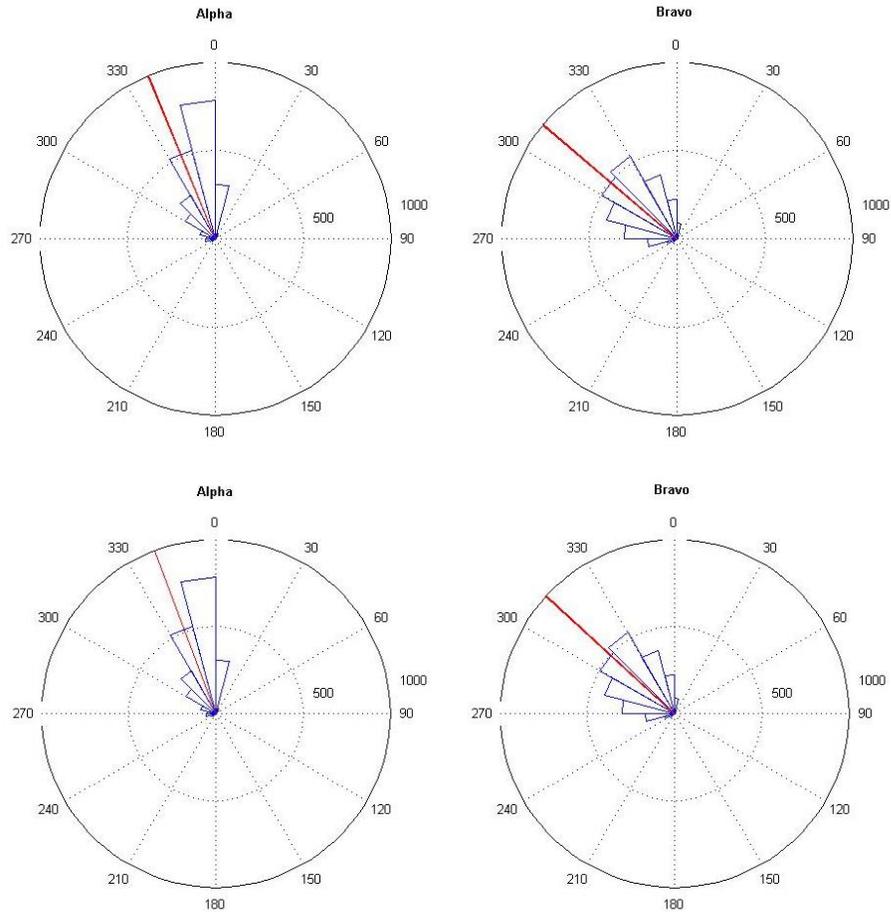


Figure 5.6: Directional rose plot with average wind direction (red) (Top), Speed weighted rose plot with weighted average (red) (Bottom)

From the controlled burn data, and additional subsequent data sets, it was determined that there was minimal discrepancy between the weighted and unweighted directional averages for a single data set.

As mentioned previously, Charlie was unable to measure the wind direction. The following is a quick recap of the additional data that was recorded during the controlled burn that features all 3 sensors along with the available Mesonet data.

The first piece of recorded data is the barometric pressure during the controlled burn (Figure 5.7).

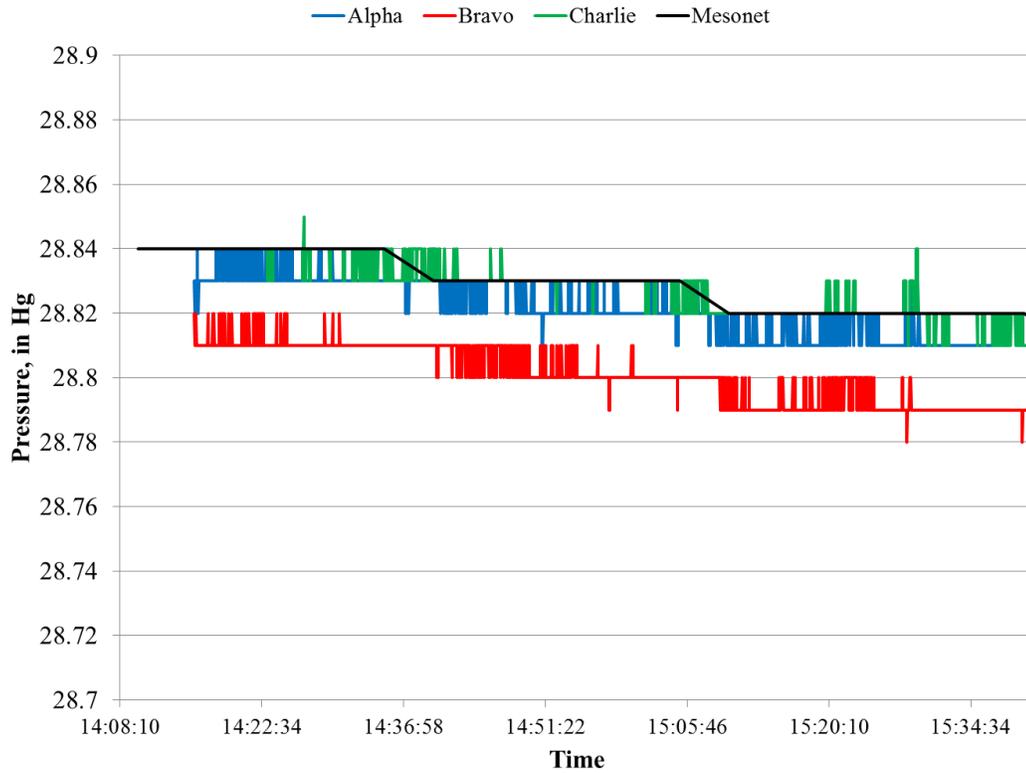


Figure 5.7: Pressure data for controlled burn

The measured pressure only slightly dropped for all 3 sensors during data collection. No additional observations were made regarding pressure during the controlled burn.

The recorded temperature can be seen in Figure 5.8.

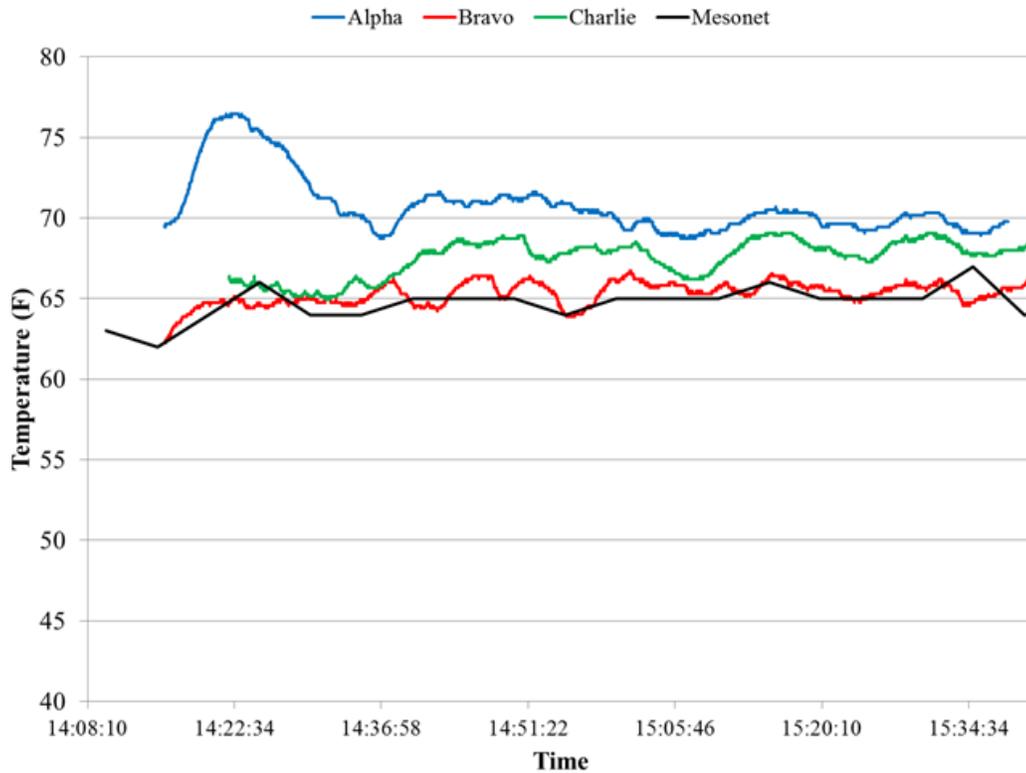


Figure 5.8: Temperature data for controlled burn

The first noticeable change in temperature is with Alpha sensor before time 14:30 C.S.T., where there is an over 6 °F change in temperature. This is attributed to the fire beginning near Alpha, and traveling north toward Charlie and then burning west toward Bravo. Also, Alpha was downwind of the fire during the duration of data collection, and would have been more susceptible to the heat from the burn. After this spike, all 3 sensors seem to follow a similar trend of rise and falls, with sensor Bravo being the closed to the Mesonet average.

The relative humidity data seemed to be noisy yet showed no large variation near the controlled burn (Figure 5.9).

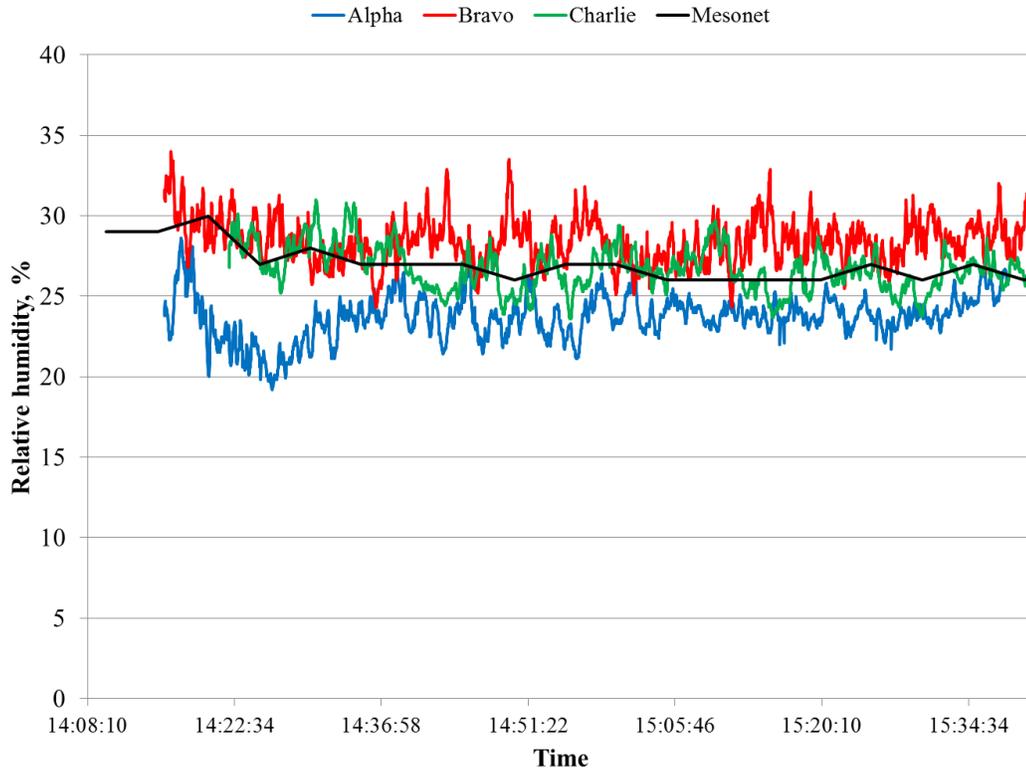


Figure 5.9: Relative humidity data for controlled burn

With the exception for the drop in humidity at Alpha in the beginning, the humidity values stayed between 20 and 35% relative humidity. This initial drop in Alpha is attributed with the rise in temperature during the same time as was seen in Figure 5.8, as relative humidity has an inverse relationship with temperature.

As discussed previously, the concept was to subtract the prevailing wind from the current measured reading to determine the wind “shift” or “third vector” which might be due to local thermal updrafts (thermal inflow direction). This subtraction method of using a moving average found from previous sets of sensor data was to be done with the data collected during this burn.

Matlab scripts were created to easily and quickly display the collected data and calculate the shifts in the measured wind. The shifts were calculated using Eq. 5.1, which is a variation to Eq. 3.16.

$$Shift_i = \vec{W}_i - \frac{1}{n} \sum_{k=i-n}^{i-1} \vec{W}_k \quad \text{Eq. 5.1}$$

Where  $\mathbf{W}_i$  represents the current wind vector,  $\mathbf{W}_k$  the average wind vector, and  $\mathbf{Shift}_i$  being the shift vector or calculated thermal vector.

Averages were initially taken over 15, 30, 60, and 150 samples, which would correspond to roughly 30 seconds, 1 minute, 2 minutes, and 5 minute averages respectively (Figure 5.10)

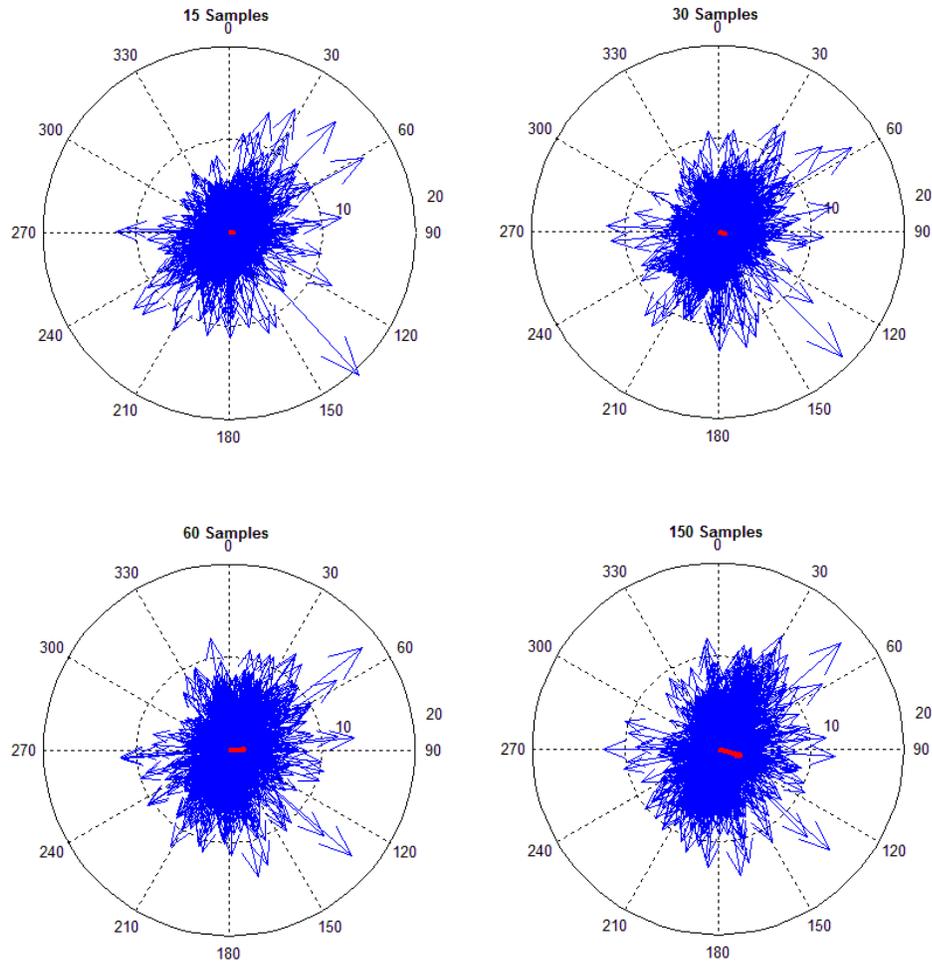


Figure 5.10: Alpha wind shifts sample rate comparisons

Figure 5.10 shows the calculated wind shifts (blue) over the entire sampling time, while also showing the total average shift (red) for sensor package Alpha during the control burn. Alpha was located east of the burn with the prevailing wind from the Northwest. It was the initial hope that the majority of the shifts would point toward the control burn, which in the case of Alpha has is not the case. The total number of shifts are scattered, where shifts were detected in all directions: some toward the control burn, others away from the burn.

In the case of Bravo, the same analysis was completed (Figure 5.11).

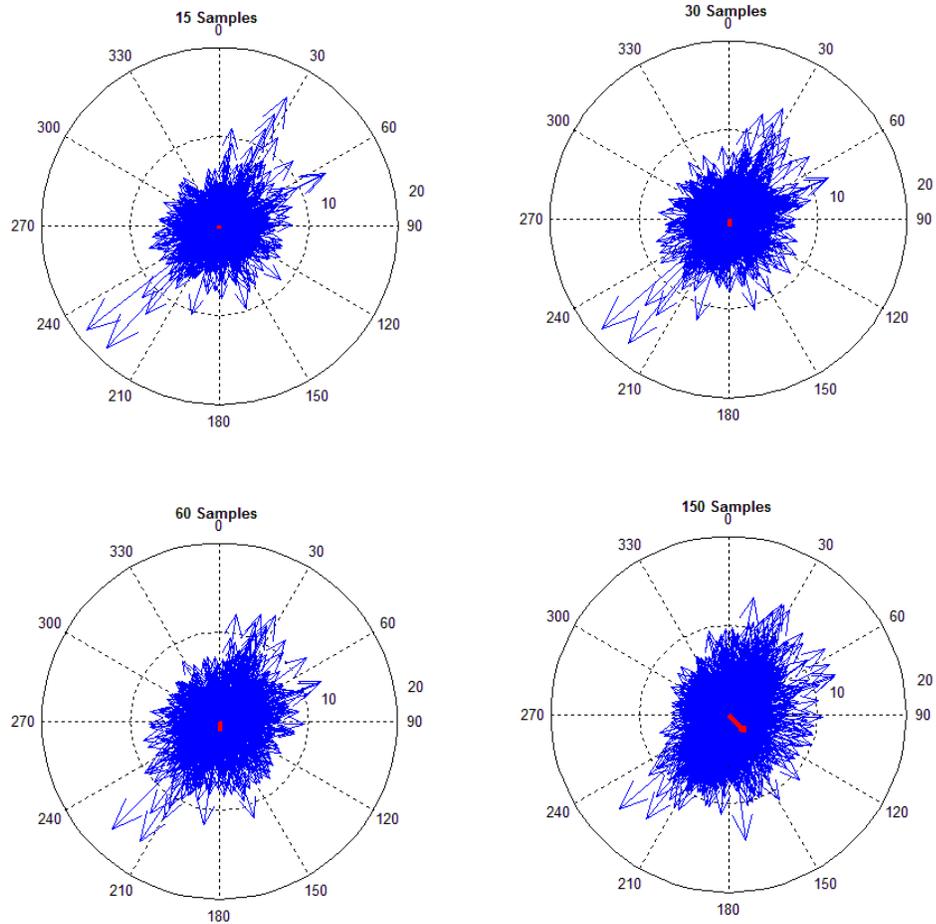


Figure 5.11: Bravo shift comparisons

As can be seen in Figure 5.11, the shifts are still numerous and spread out over the entire compass plot with no majority indicating a thermal to the South of Bravo’s location.

It was thought that taking an even larger sample size for the average would help in finding the more prominent shifts. Through looking at different averaging lengths, and through additional literature survey, a 10 min or 300 data point average was chosen for future data collection for the sensor interface<sup>48</sup>. Even though this averaging method was used for the sensor

interface during data collection, additional methods could be visualized and simulated at a later time by using the collected data.

### 5.3 FLIGHT FIELD DATA

Over 11 days' worth of data were collected with over 60 confirmed thermal updrafts. Once data were collected, videos were edited to overlay the documented thermals with the sensor data from the GUI. The following subsections will examine 2 data collection days with prominent thermal events that were found during the course of data collection in order to give an overview of the data that was collected as well as examine the found thermals.

To recap, sensors were constructed to measure air properties around a flight field in order to measure the changes that would occur around forming thermal updrafts. These sensors were designed to measure wind speed and direction, temperature, barometric pressure, and relative humidity. The information was collected at 2 second intervals and wirelessly transmitted to a local computer which displayed the values in real time. This data was collected over periods ranging between 1 to 4 hours, while signs of thermal updrafts were being visually recorded. These signs primarily included birds circling in thermals and/or RC aircraft becoming caught in the updrafts. During days of data collection, at least one RC glider was flown around the airfield in search of thermal updrafts. Along with using the RC glider, observations were also made on the local avian wildlife and wind socks to indicate signs of thermals, thus allowing the RC pilot to focus the thermal search pattern.

#### 5.3.1 JUNE 1<sup>ST</sup>, 2015 DATA

During June 1<sup>st</sup>, 2015, 3 sensor packages were distributed around the OSU UAS Flight Field, located in Stillwater, Oklahoma (36°09'43.5"N 96°50'08.3"W) for data collection.



Figure 5.12: OSU Unmanned Aircraft Flight Station with sensor locations

Field measurements were taken between 09:30 and 11:23 C.S.T. During this time, the prevailing wind was from the East with average speed of 4 mph with gusts up to 11 mph. During data collection, 11 thermal events were discovered.

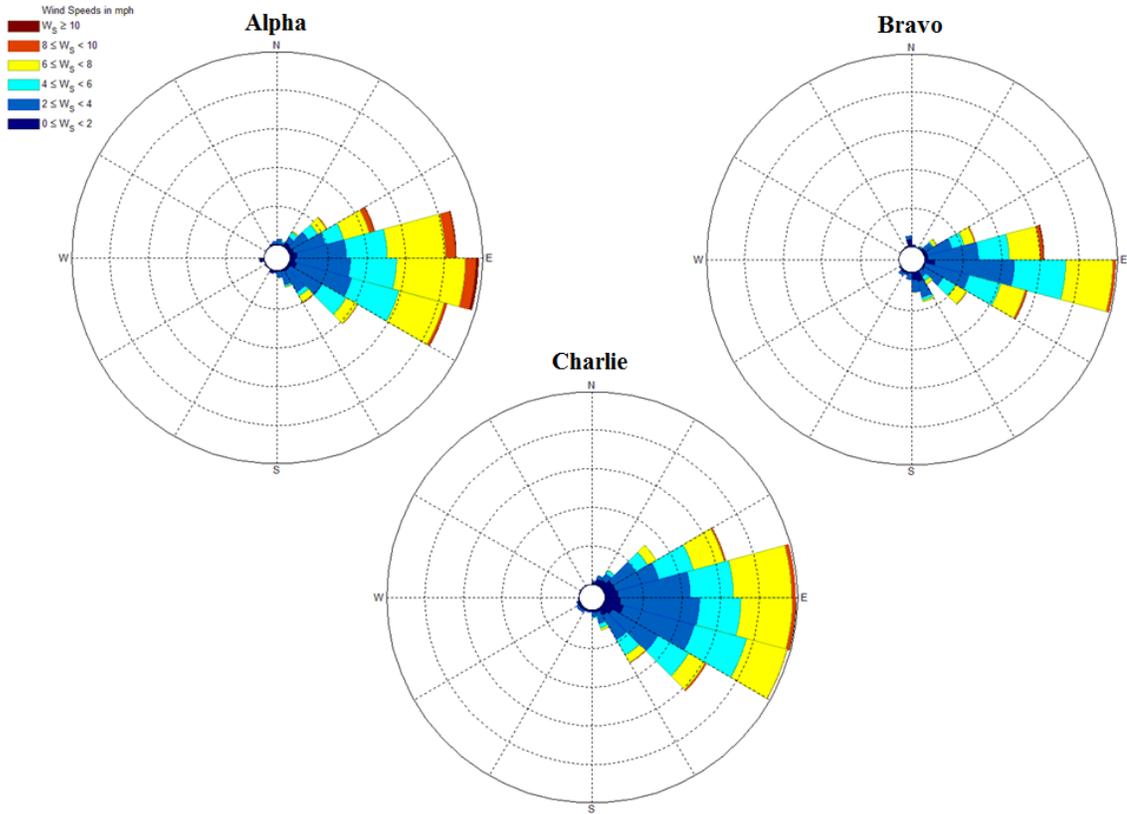


Figure 5.13: Wind rose of each sensor package during data collection

The sensor packages measured the azimuth direction of the wind from the attached electronic wind vane. Average wind direction was calculated from each individual sensor station and plotted along the rose plot (Figure 5.14) to compare with the day's weather forecast (winds from the East at  $100^\circ$  with speeds between 3 and 6 mph).

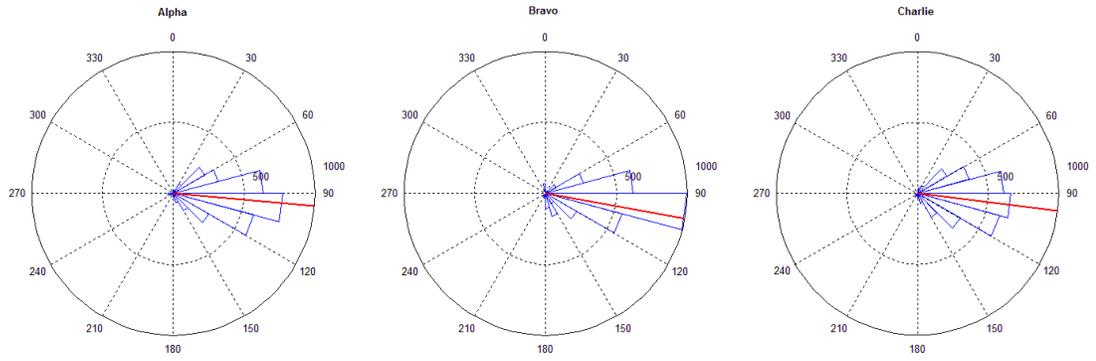


Figure 5.14: Rose plots of wind direction (blue) and average direction (red) during data collection

Along with wind speed and direction, air temperature, relative humidity levels, and barometric pressure were once again measured and examined at each location.

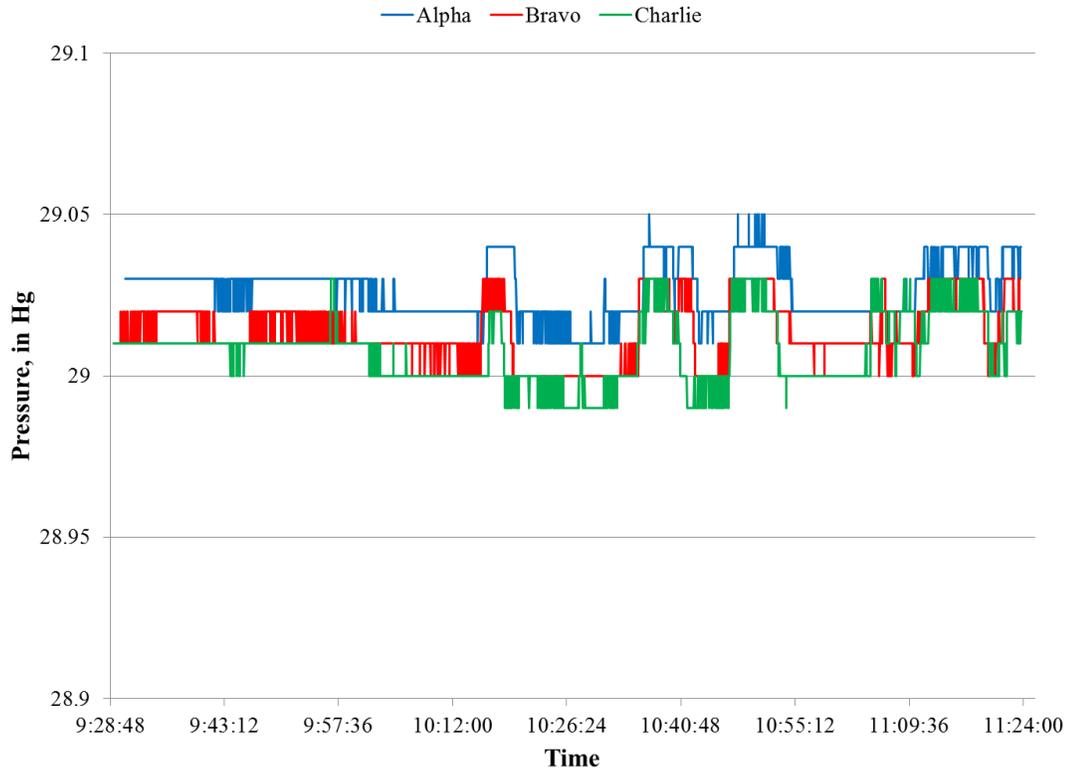


Figure 5.15: Pressure data for June 1<sup>st</sup>, 2015

Similar to what was seen during the controlled burn, little variation in pressure is noticed.

Temperature and relative humidity values during the day's data collection can be seen in Figure 5.16 and Figure 5.17 respectively.

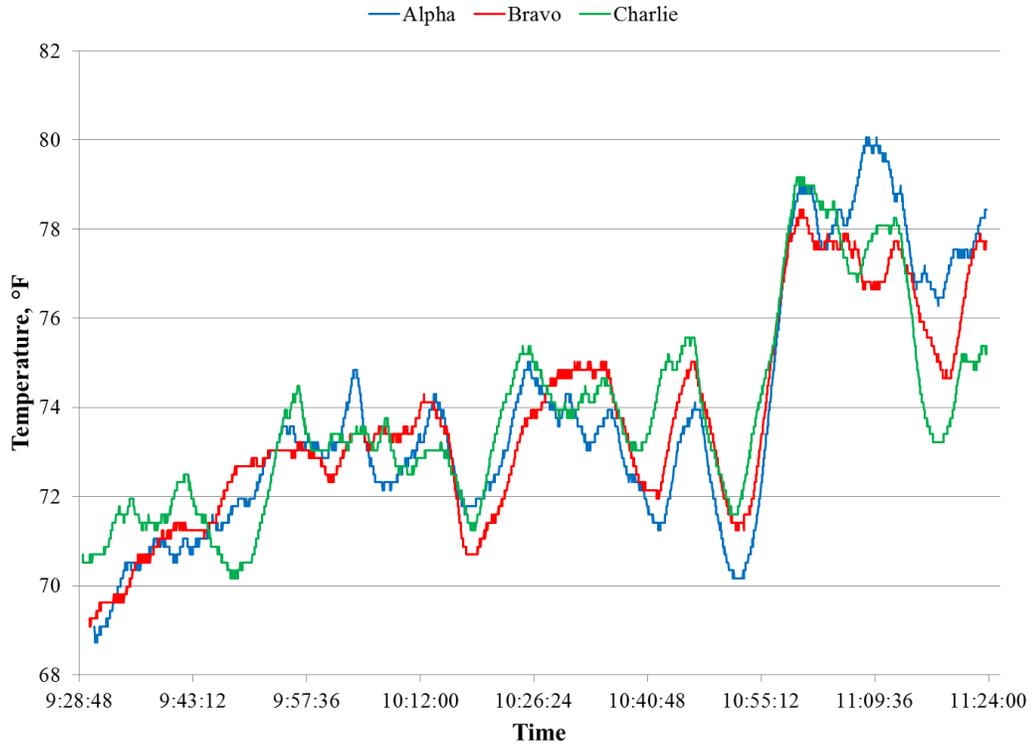


Figure 5.16: Temperature results during June 1<sup>st</sup>, 2015

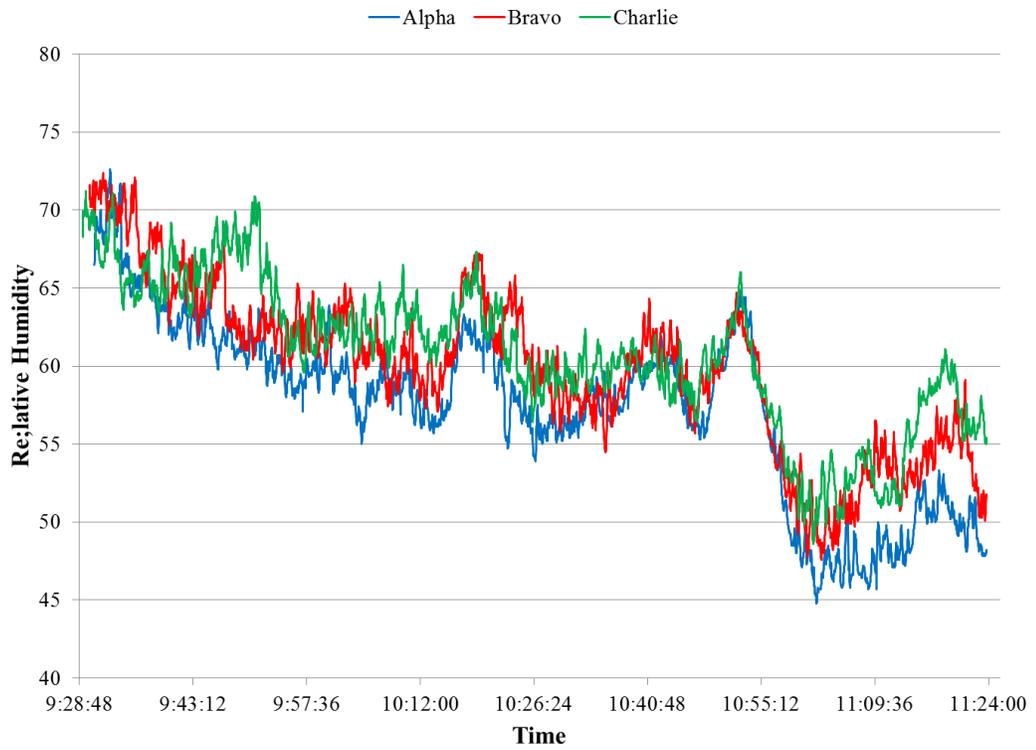


Figure 5.17: Relative humidity levels during June 1<sup>st</sup>, 2015

With data collection starting in the morning hours, the temperature was still steadily climbing. From looking at the total time frame, one can see the general trends of the temperature and humidity data, such as increase in temperature during the morning, but areas of abrupt change in multiple locations can also be seen in the previous figures as emphasized in Figure 5.18 and Figure 5.19.

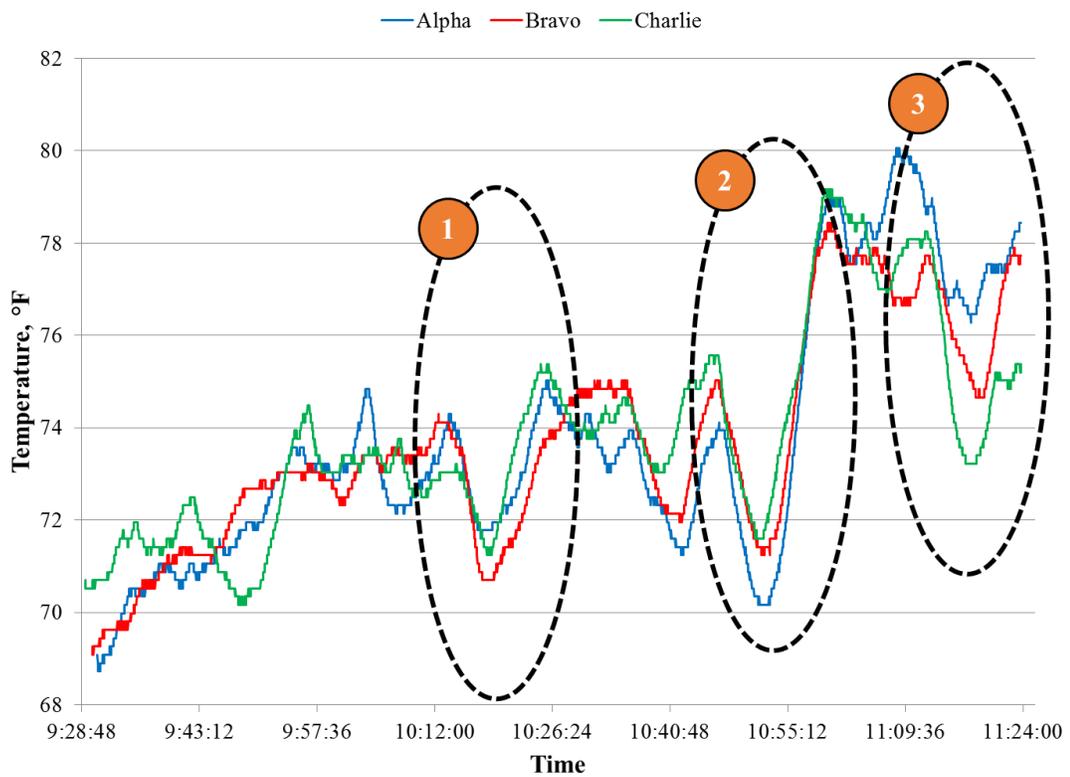


Figure 5.18: Noticed temperature drops for June 1<sup>st</sup>, 2015

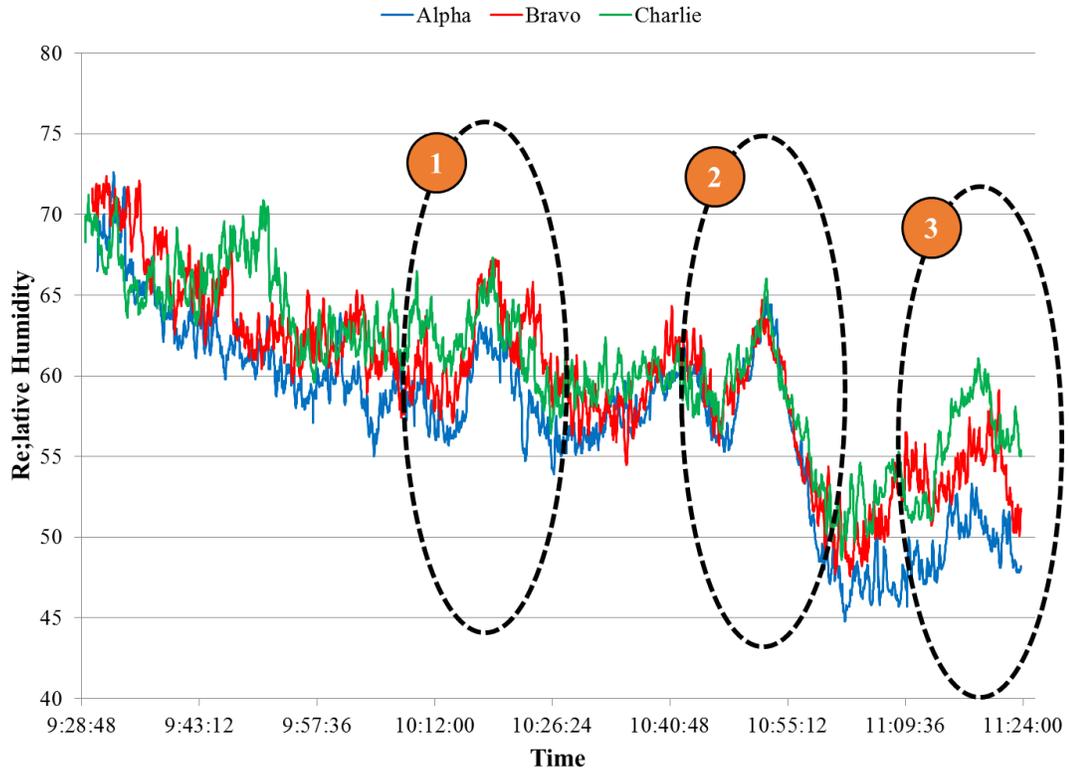


Figure 5.19: Noticed humidity spikes for June 1<sup>st</sup>, 2015

These areas, such as the quick change in temperature and humidity between 10:45 and 11:00 C.S.T., might be indications of thermals or another event. However a thermal updraft cannot be conclusively identified during these times except for event 3 labeled on the graphs. Event 3 coincides with the thermal discovered at 11:13 C.S.T.

At approximately 11:13 C.S.T., a thermal was found to the West of the airfield, which was downstream of the prevailing wind. This was substantiated by 1) the RC glider maintaining and gaining altitude while circling in a certain area and 2) vultures were observed in the same area circling while also gaining altitude with no flapping motion. Figure 5.20 shows the wind data collected during a smaller time period between 11:00 and 11:15 C.S.T.

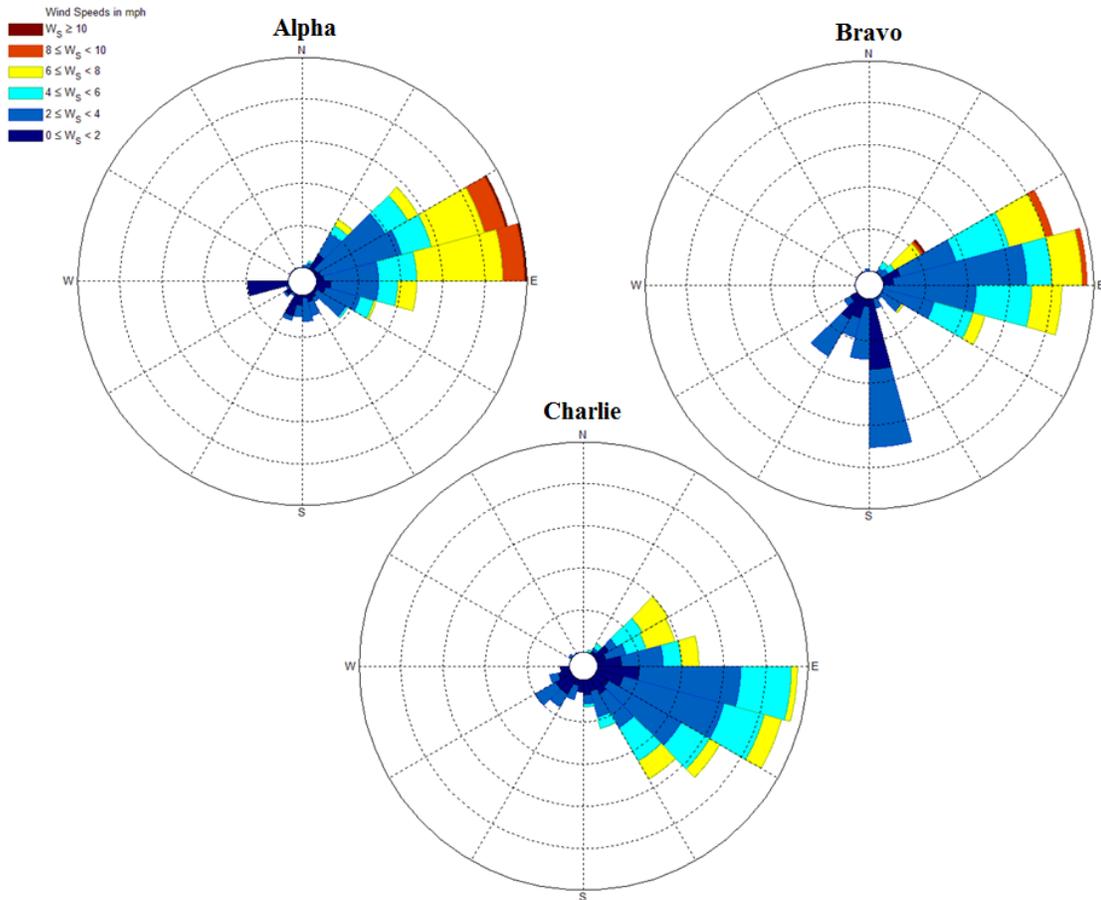


Figure 5.20: Thermal wind rose for June 1<sup>st</sup>, 2015 (11:00 - 11:15 C.S.T.)

This smaller time frame allows for a closer look into the local shifts that the sensor packages measure that could be due to the local thermal updraft. Primarily the winds are still seen to be from the East, yet there can also be seen times of extreme change. These changes can primarily be seen for Bravo sensor as the wind direction comes from the south for multiple data points at a slightly lower wind speed.

Images of the sensor interface during the found thermal were captured for analysis.

Figure 5.21 was captured at 11:13:10 C.S.T. with the approximate location of the thermal shown.

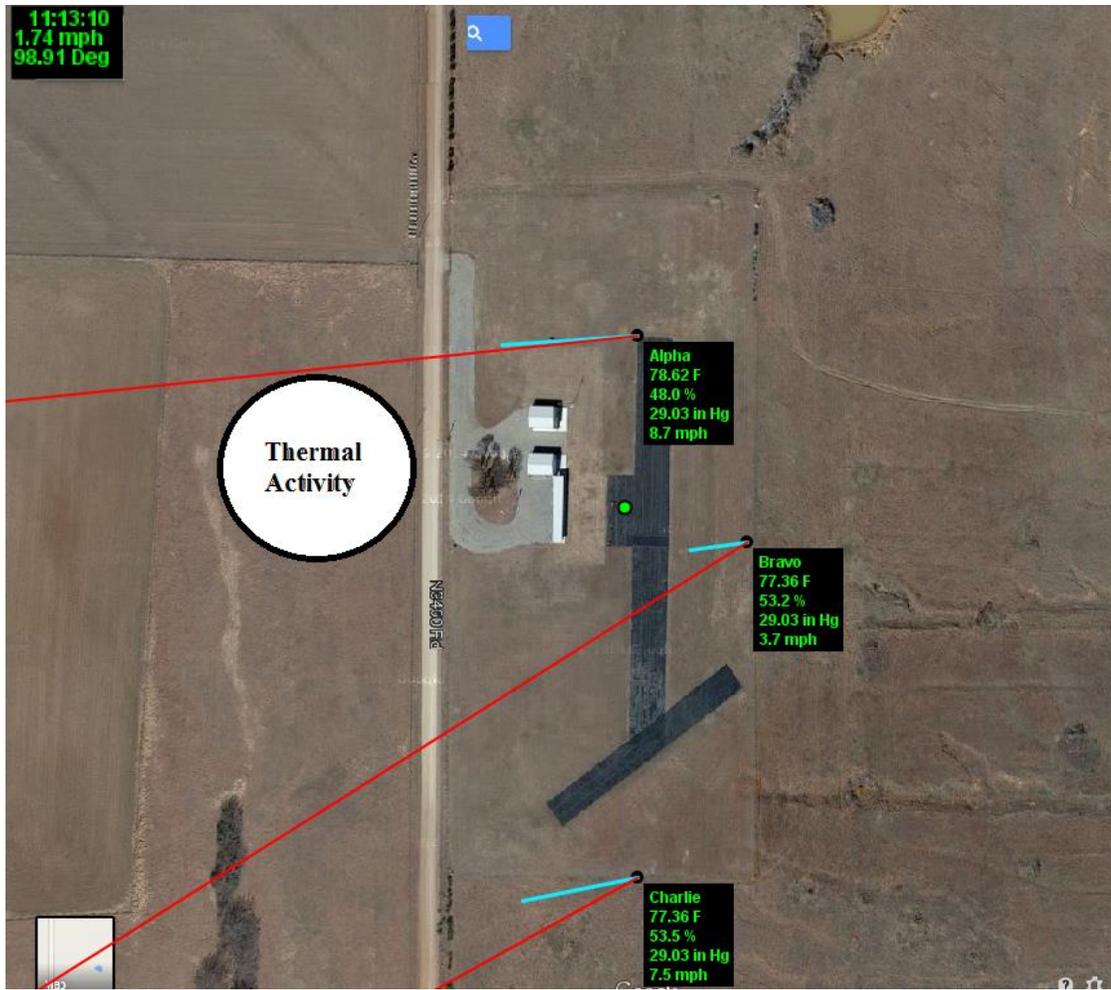


Figure 5.21: June 1<sup>st</sup>, 2015 thermal event

The sensor interface showed no triangulation from all 3 sensors during this time in response to the found thermal. Nor did the instantaneous wind vectors give any indication of a large inflow by pointing toward this area.

### 5.3.2 SEPTEMBER 10<sup>TH</sup>, 2016 DATA

On September 10<sup>th</sup>, 2015, 15 thermal events were documented over a 4 hour time span. For this data set, the average vector was composed of the previous 300 data points ( $N = 300$ ), while the current vector was composed of the current data point and the previous point before

( $F = 2$ ). The forecasted wind conditions for September 10<sup>th</sup> was approximately 8 mph wind at an azimuth indicator of 190°.

Again, pressure, temperature, and relative humidity levels were recorded for the day.

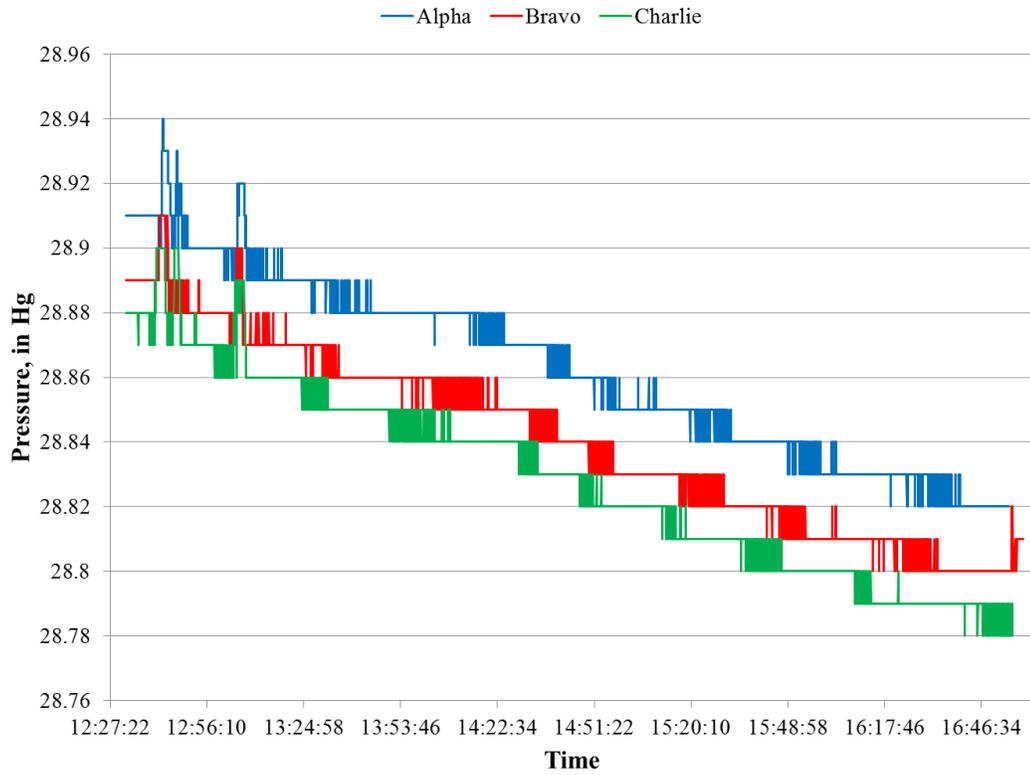


Figure 5.22: Pressure data for September 10<sup>th</sup>, 2015

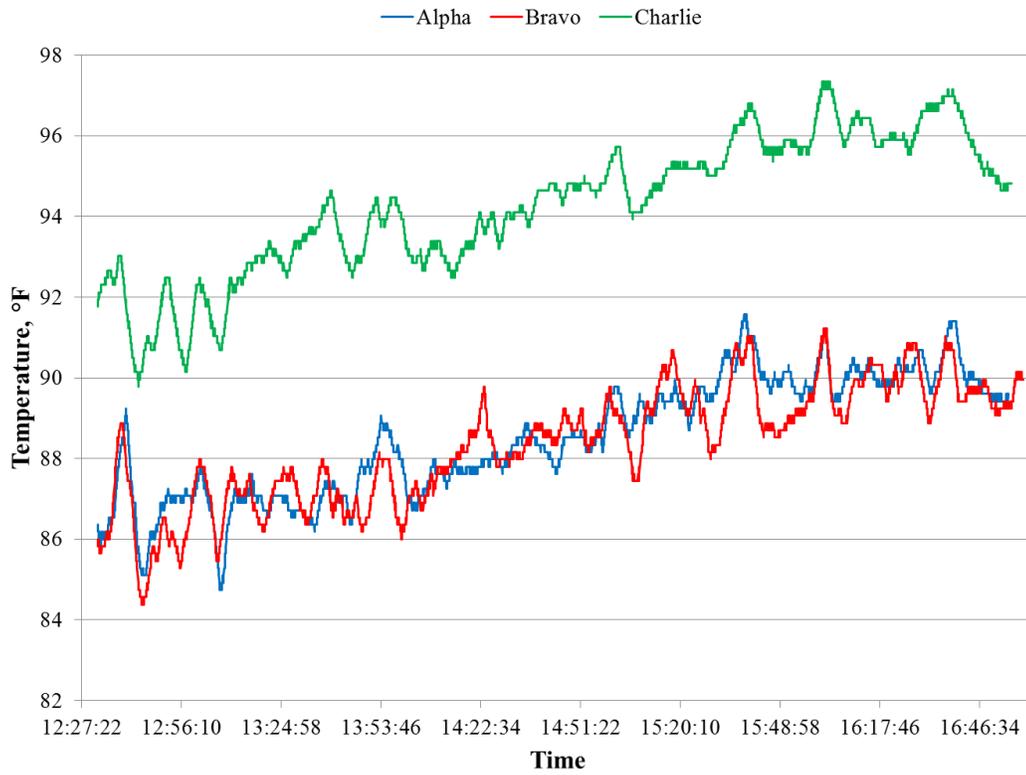


Figure 5.23: Temperature data for September 10<sup>th</sup>, 2015

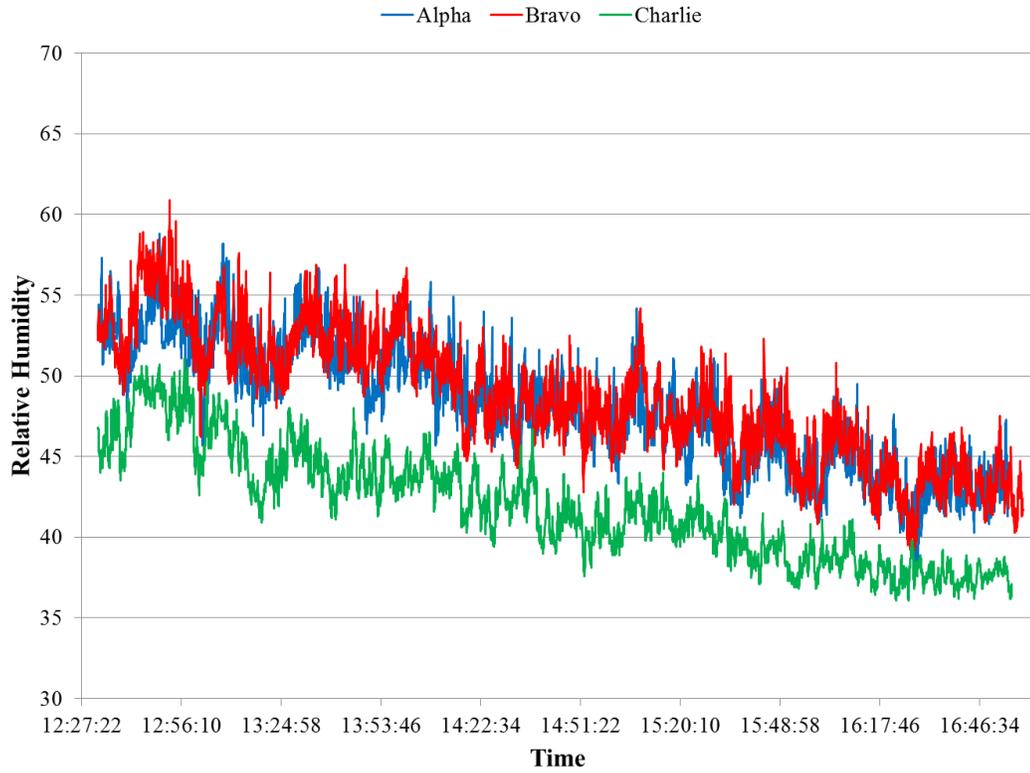


Figure 5.24: Humidity data for September 10<sup>th</sup>, 2015

Throughout the day, there was a steady drop in pressure and humidity, with an expected rise in temperature.

Primarily, the changes in wind speed and direction were examined. For this day of testing, the wind was primarily out of the South, which is consistent for all 3 sensors.

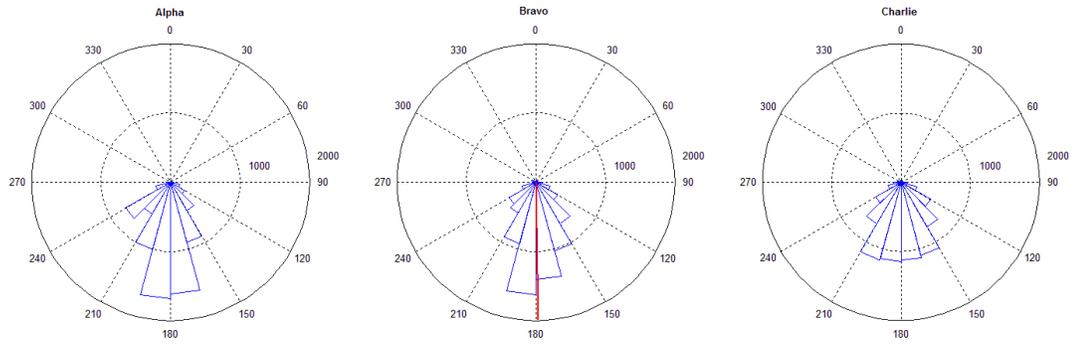


Figure 5.25: Compass plots for September 10<sup>th</sup>, 2015

All 3 sensors showed a majority of the wind speed values between 3 and 9 mph for the testing time.

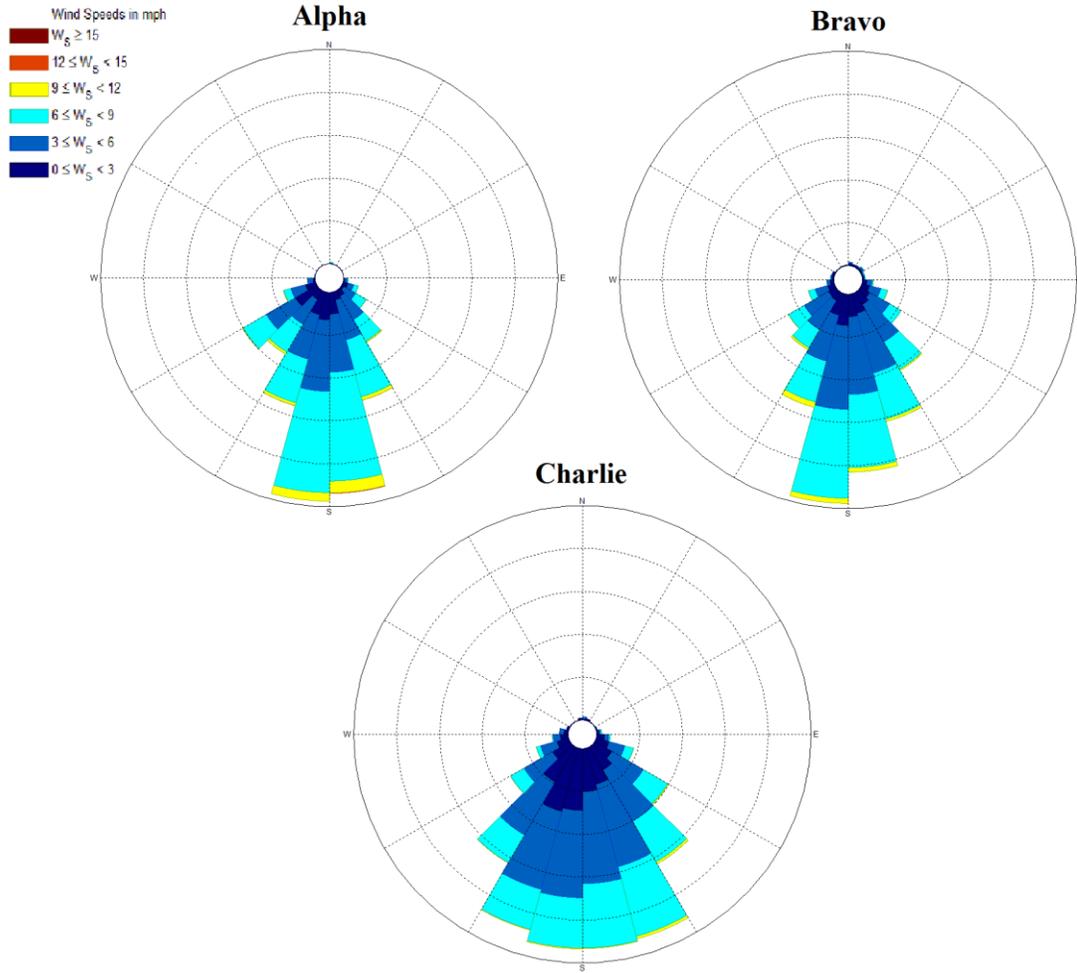


Figure 5.26: Wind rose plots for September 10<sup>th</sup>, 2015

During the 4.5 hours of data collection, 15 confirmed thermal events were found around the airfield.

Around 15:55 C.S.T., a large thermal was located northwest of the flight field (Figure 5.27).

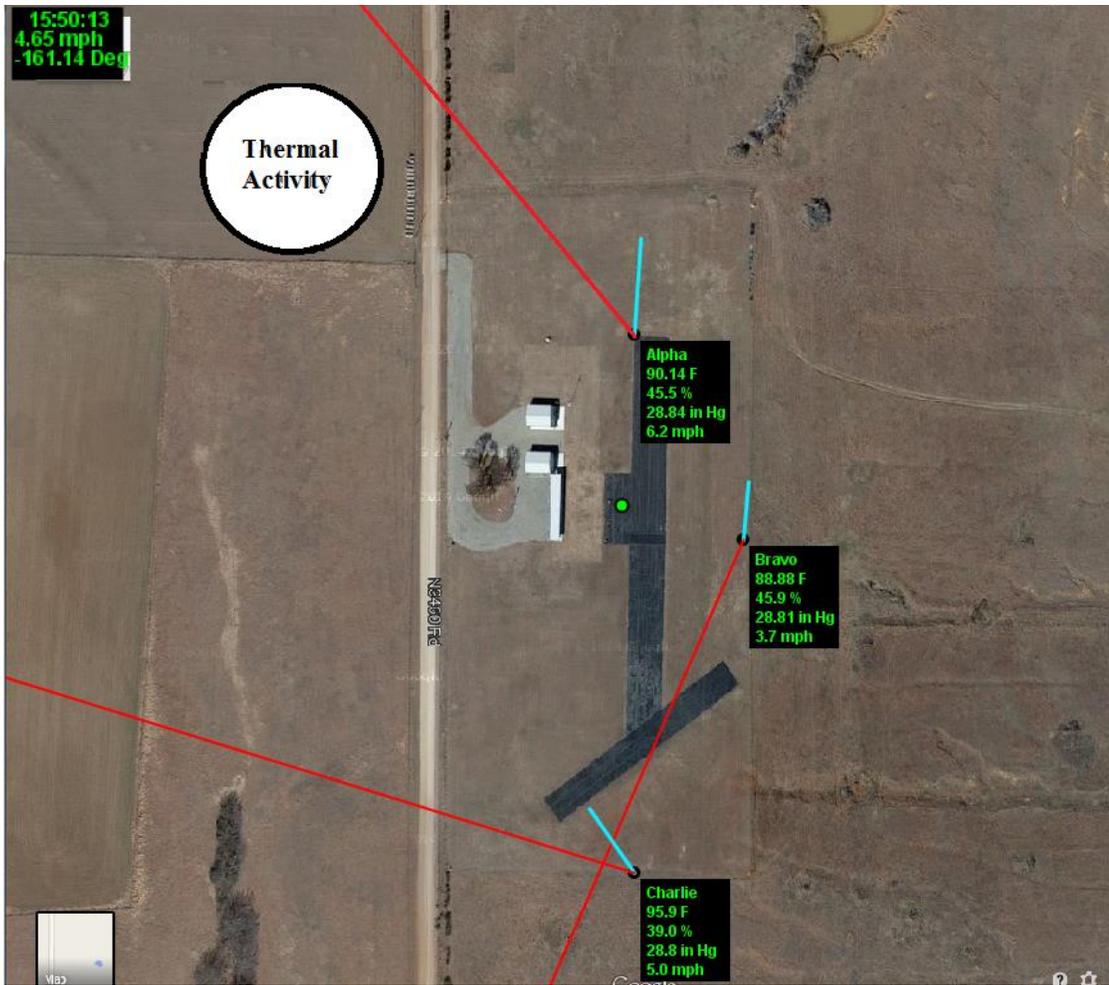


Figure 5.27: September 10th thermal location found at 15:55 C.S.T.

This thermal was being utilized by 10 large soaring birds and was also confirmed by an RC aircraft. The 3 on-site sensors showed a stronger pull towards the area of interest during this thermal event (Figure 5.28).

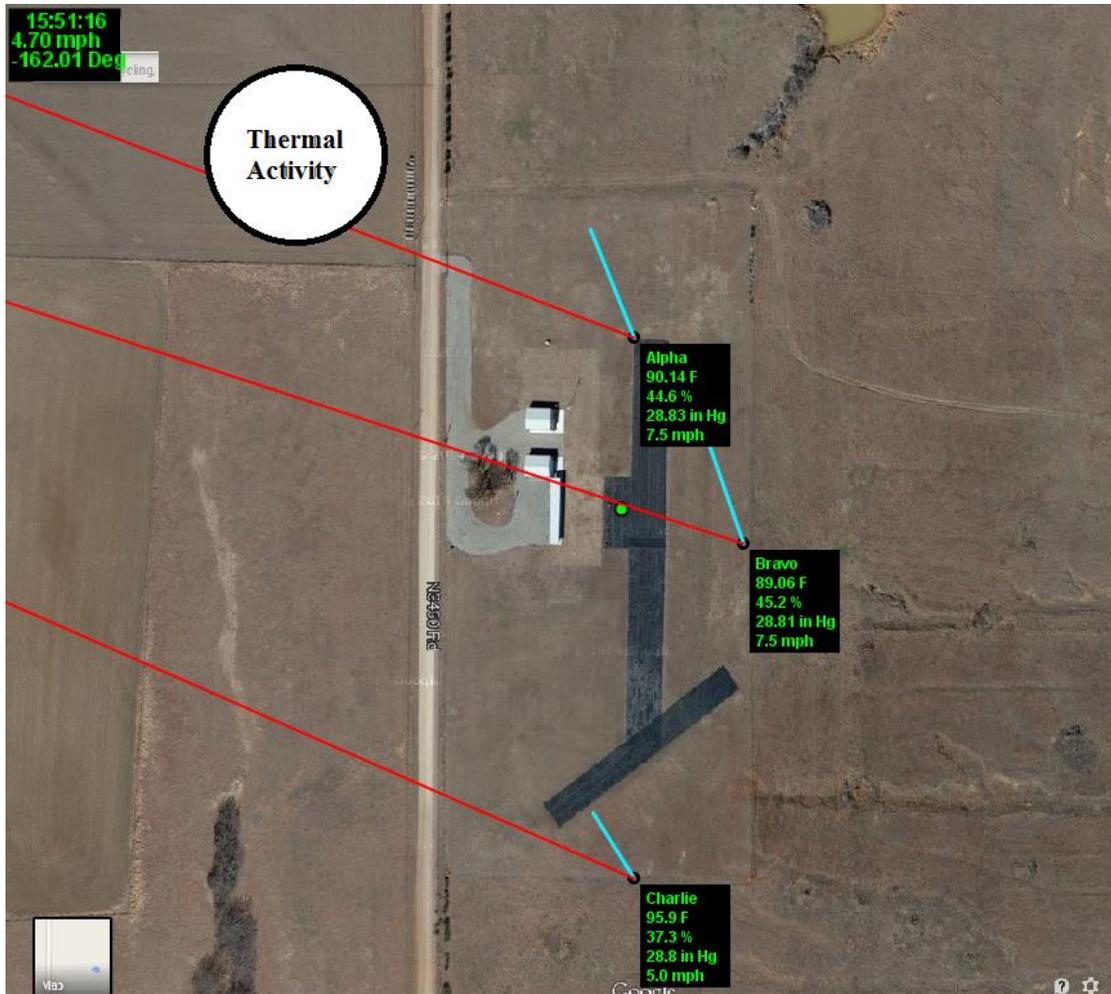


Figure 5.28: September 10th thermal pull

Not only did all 3 sensors show the current wind vector tilt in the direction of the thermal, but all 3 calculated thermal inflow vectors pointed in the vicinity of the thermal for a brief time (Figure 5.29).

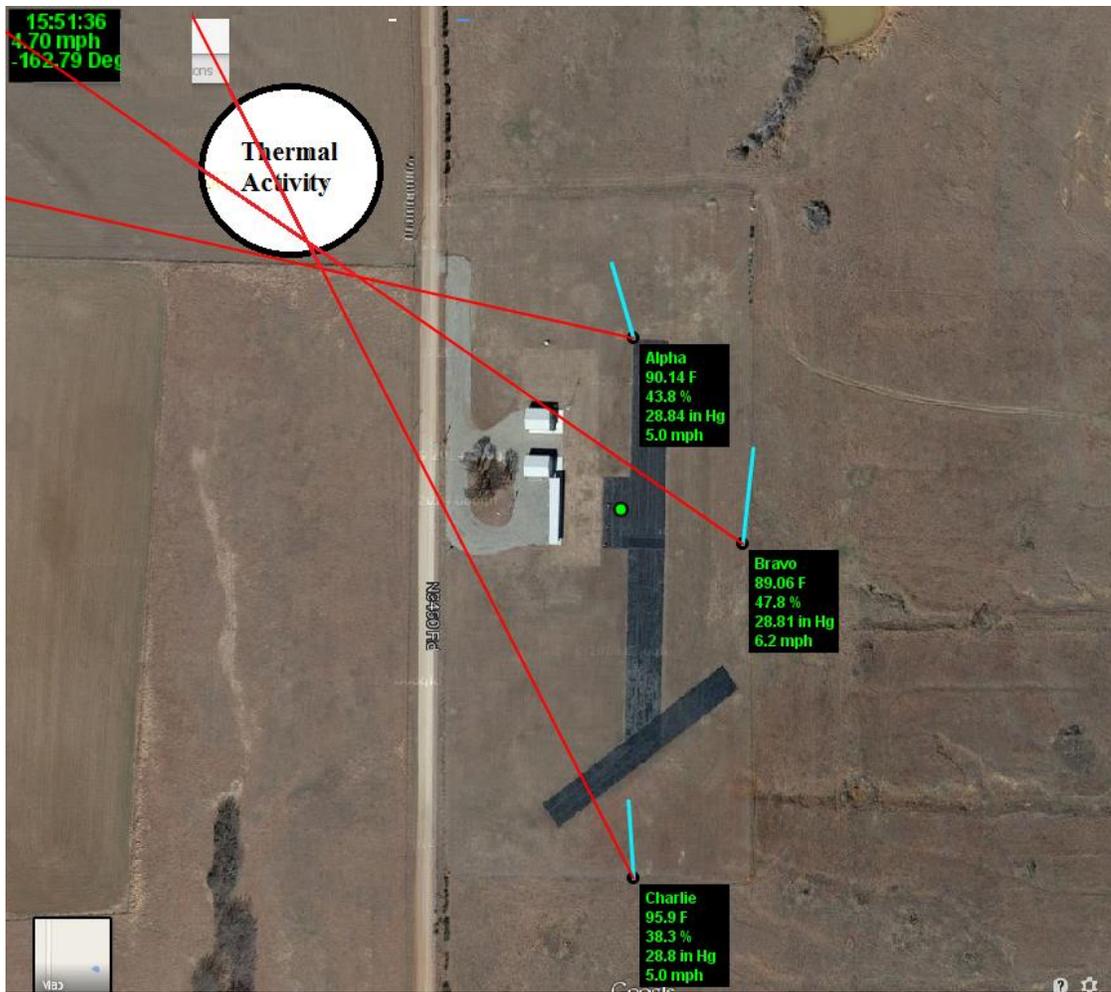


Figure 5.29: September 10th thermal indications

This is one of the few cases where the thermal was either strong enough or close enough to affect all 3 sensors and show strong evidence of the detection method to work. All other data points for the day however have shown to be inconclusive of indicating known thermal activity.

### 5.3.3 ADDITIONAL DATA

As mentioned over 11 days of data regarding discovered thermals were also collected and analyzed. The data collected on these days showed similar data trends along with similar lack of thermal identification by the current thermal locating technique. Not only was there minimal

conclusive evidence from the thermal triangulation with regards to found thermals, but there was also times of “false” intersections. These intersections were unsubstantiated by RC or avian evidence of thermal activity and could therefore not be used for thermal identification.

#### 5.4 UNCERTAINTY

Uncertainty analysis was utilized on the primary elements of the theorized thermal experimentation. Each of the individual components was examined to determine the inaccuracies of each individual part of the experimental apparatus. Along with each component being analyzed, an uncertainty analysis was also used to examine the possible propagation of inaccuracies with in the calculations used by the sensors and the interface.

Each component that comprised the sensor package had its own documented accuracy attached to it (Table 5.2).

Table 5.2: Component accuracy

Component	Accuracy
Pressure BMP180	$\pm 0.02$ hPa
Temperature RHT03	$\pm 0.5$ °C
Relative Humidity RHT03	$\pm 2\%$ RH
Inspeed E-Vane	$\pm 0.3$ to $0.5\%$ of signal range (12bit res) (6deg)
Inspeed Vortex	$\pm 1$ pulse (2.5mph) Over 2 sec (1.25mph)

With the focus of the thermal detection primarily dealing with wind speed and direction, the accuracy of the Inspeed E-Vane and Vortex sensors were used in uncertainty calculations regarding the experimental calculations.

#### 5.4.1 UNCERTAINTY EQUATIONS

Additionally while looking at the results, the uncertainty shows large quantities of inaccuracies while measuring the “third vector.” Uncertainty analysis was utilized on the primary elements of the theorized thermal experimentation. Each of the individual components were examined to determine the inaccuracies of each individual part of the experimental apparatus. Along with each component being analyzed, an uncertainty analysis was also used to examine the possible propagation of inaccuracies with in the calculations used by the sensors and the interface.

Each component that comprised the sensor package had its own documented accuracy attached to it. Inaccuracies within the individual components were not the only possibility for uncertainty within the data collection. The wind speed and direction were used in further calculations to determine areas of wind shift and to determine the direction of the shift. Within these calculations, uncertainty could propagate within the experimental environment and allow for additional uncertainty. For a given calculation  $R$ , that is computed from an  $n$  number of measurements  $(x_1, x_2, \dots x_n)$ , the propagation of uncertainty in the result can be computed by Eq. 5.2<sup>39,40</sup>.

$$W_R = \sqrt{\left(\frac{\partial R}{\partial x_1} W_{x_1}\right)^2 + \left(\frac{\partial R}{\partial x_2} W_{x_2}\right)^2 + \dots + \left(\frac{\partial R}{\partial x_n} W_{x_n}\right)^2} \quad \text{Eq. 5.2}$$

Or by looking at the percentage of the result, the uncertainty equation could be rewritten as

$$\frac{W_R}{R} = \sqrt{\left(\frac{\partial R}{\partial x_1} \frac{W_{x_1}}{R}\right)^2 + \left(\frac{\partial R}{\partial x_2} \frac{W_{x_2}}{R}\right)^2 + \dots + \left(\frac{\partial R}{\partial x_n} \frac{W_{x_n}}{R}\right)^2} \quad \text{Eq. 5.3}$$

The first uncertainty analysis was computed on the calculation of the individual  $x$  and  $y$  components of the wind vectors. Using Eq. 3.17 and Eq. 3.18 as the resultant equations for propagation, the uncertainty calculation for the both the  $x$  and  $y$  components were derived to be Eq. 5.4 and Eq. 5.5 respectively.

$$\frac{W_{V_x}}{V_x} = \sqrt{\left(\frac{\partial V_x}{\partial speed} \frac{W_{speed}/speed}{V_x}\right)^2 + \left(\frac{\partial V_x}{\partial direction} \frac{W_{direction}/direction}{V_x}\right)^2} \quad \text{Eq. 5.4}$$

$$\frac{W_{V_y}}{V_y} = \sqrt{\left(\frac{\partial V_y}{\partial speed} \frac{W_{speed}/speed}{V_y}\right)^2 + \left(\frac{\partial V_y}{\partial direction} \frac{W_{direction}/direction}{V_y}\right)^2} \quad \text{Eq. 5.5}$$

The uncertainty for the purchased wind speed anemometer was 1.25 mph, while the uncertainty for the electronic wind vane was 6°. Each of these uncertainties was initially equated as a percentage of the calculated result. Due to the sine and cosine terms, the uncertainty was recalculated to show the value of the uncertainty instead of the percentage. This was due to the areas where the sine and cosine became 0 (0° and 180° for sine, and 90° and 270° for cosine), the percentage would approximate infinity due to the 0 term in the denominator. Thus the

uncertainty was to be recalculated as a specific value instead of a percentage (Eq. 5.6 and Eq. 5.7).

$$W_{V_x} = \sqrt{\left(\frac{\partial V_x}{\partial speed} W_{speed}\right)^2 + \left(\frac{\partial V_x}{\partial direction} W_{direction}\right)^2} \quad \text{Eq. 5.6}$$

$$W_{V_y} = \sqrt{\left(\frac{\partial V_y}{\partial speed} W_{speed}\right)^2 + \left(\frac{\partial V_y}{\partial direction} W_{direction}\right)^2} \quad \text{Eq. 5.7}$$

After changing the uncertainty equations to values instead of percentages, the uncertainty was heavily driven by the anemometer's resolution at low speeds, with the influence of direction adding a slight sinusoidal motion elsewhere.

It was also desired to determine the propagation of the uncertainty within the final calculation of the thermal direction using Eq. 3.21. For this uncertainty analysis, the uncertainty would be composed of four separate components: the wind speed for both the measured vector and the average vector, along with the wind direction for the measured vector and the average vector. Eq. 3.21 could be rewritten as Eq. 5.8 using these four components.

$$\theta_{thermal} = \tan^{-1}\left(\frac{speed_i \sin(direction_i) - speed_{avg} \sin(direction_{avg})}{speed_i \cos(direction_i) - speed_{avg} \cos(direction_{avg})}\right) \quad \text{Eq. 5.8}$$

This would expand the uncertainty calculation to be expanded for four components resulting in Eq. 5.9

$$W_{thermal} = \sqrt{\left(\frac{\partial\theta_{thermal}}{\partial speed_i} W_{speed}\right)^2 + \left(\frac{\partial\theta_{thermal}}{\partial speed_{avg}} W_{speed}\right)^2 + \dots} \quad \text{Eq. 5.9}$$

$$\left(\frac{\partial\theta_{thermal}}{\partial direction_i} W_{direction}\right)^2 + \left(\frac{\partial\theta_{thermal}}{\partial direction_{avg}} W_{direction}\right)^2$$

Again, due to the trigonometric functions, it was necessary to keep the uncertainty in terms of the units of the calculation and not as a percentage value of the resultant.

#### 5.4.2 CALCULATED UNCERTAINTY

When first examining the uncertainty, the propagation of uncertainty was calculated using set variations between the two vectors. Eq. 5.9 was used with changes in the instantaneous wind speed and direction, while maintaining constant average wind conditions. Figure 5.30 shows the 3-dimensional surface plot of the calculated uncertainty for average wind conditions of 5 mph and 30°.

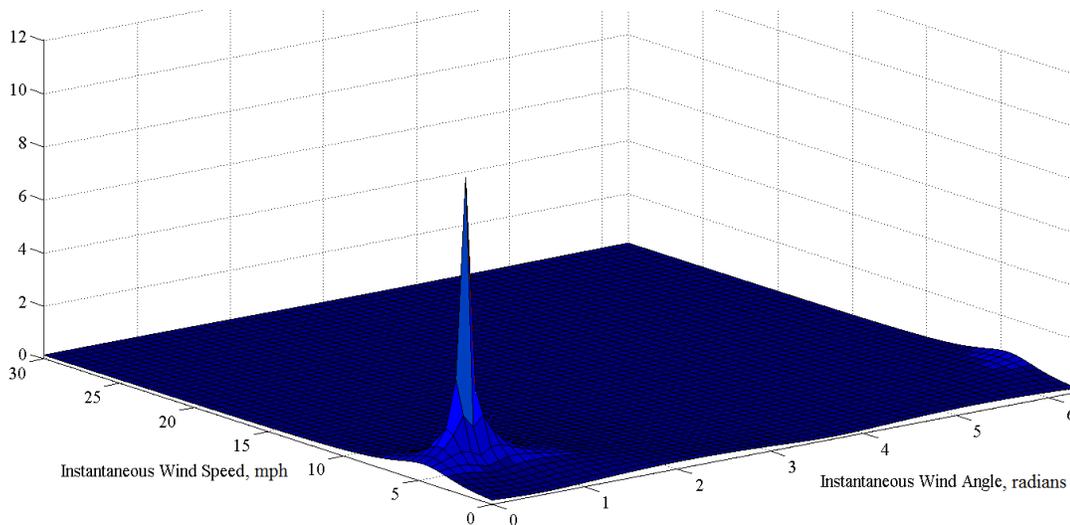


Figure 5.30: Uncertainty levels across a simulated test case using an average vector of 5 mph at

30°

Figure 5.31 shows a 2-dimensional cross section of the uncertainty versus the angle at the same conditions.

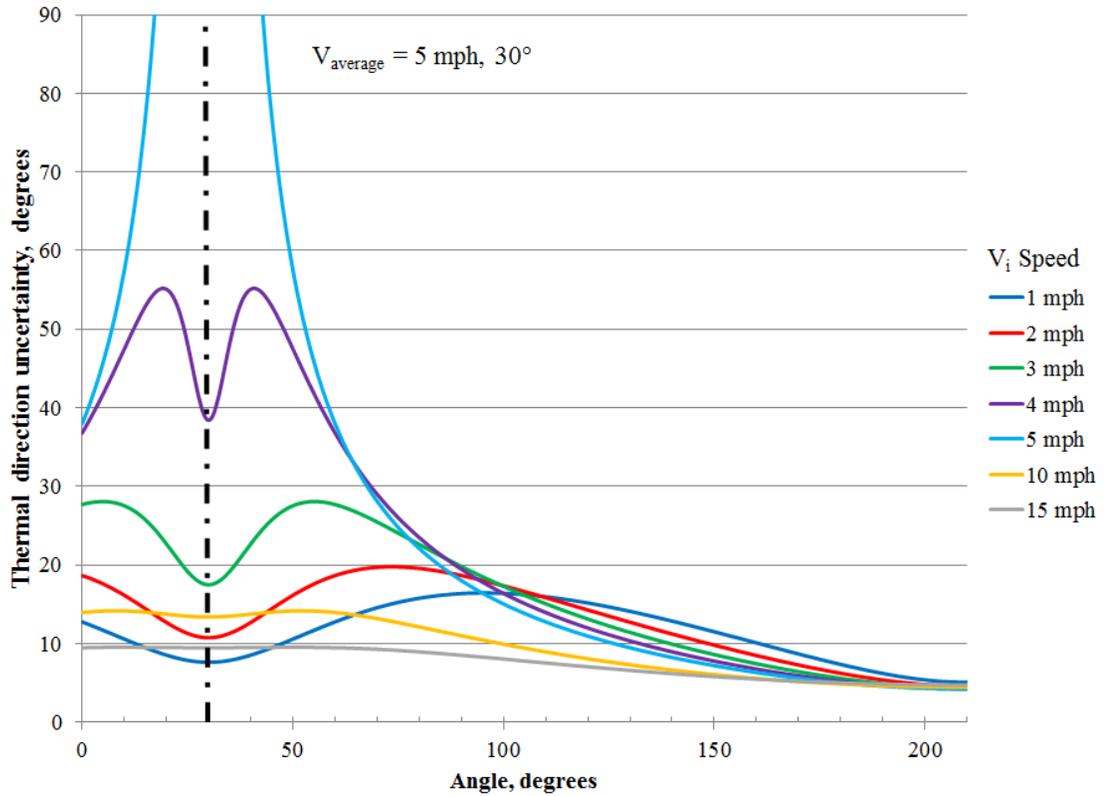


Figure 5.31: Uncertainty with differences between the instantaneous and average vectors

Other values of the average were compared and also showed high levels of uncertainty when the speeds and directions were similar. The trends for the other comparisons were similar. Meaning, when calculating the thermal direction at times when the instantaneous and average wind conditions are similar, the uncertainty of these values increases such that the thermal calculation can no longer be considered valid. This also coincides with the thermal events that were purely down wind of the sensors (as with June 1<sup>st</sup>) in which there was no indication of the

thermal on the GUI. These uncertainty values only show the minimum of the uncertainty when using this method, focusing on the uncertainty within the equation base. This does not factor in the statistical uncertainties of the changes with the wind.

### 5.4.3 UNCERTAINTY WITHIN DATA

Along with modeling the uncertainty, sensor data from the sample data points discussed earlier were also used. Taking data when thermals were found, the average wind using the previous 300 data points was calculated and used for the uncertainty calculation when compared to the current value. With the strong thermal indicator from September 10<sup>th</sup>, Figure 5.32 and Figure 5.33 show the uncertainty of the thermal direction calculation for a 15 minute window containing the thermal found at 15:55 C.S.T.

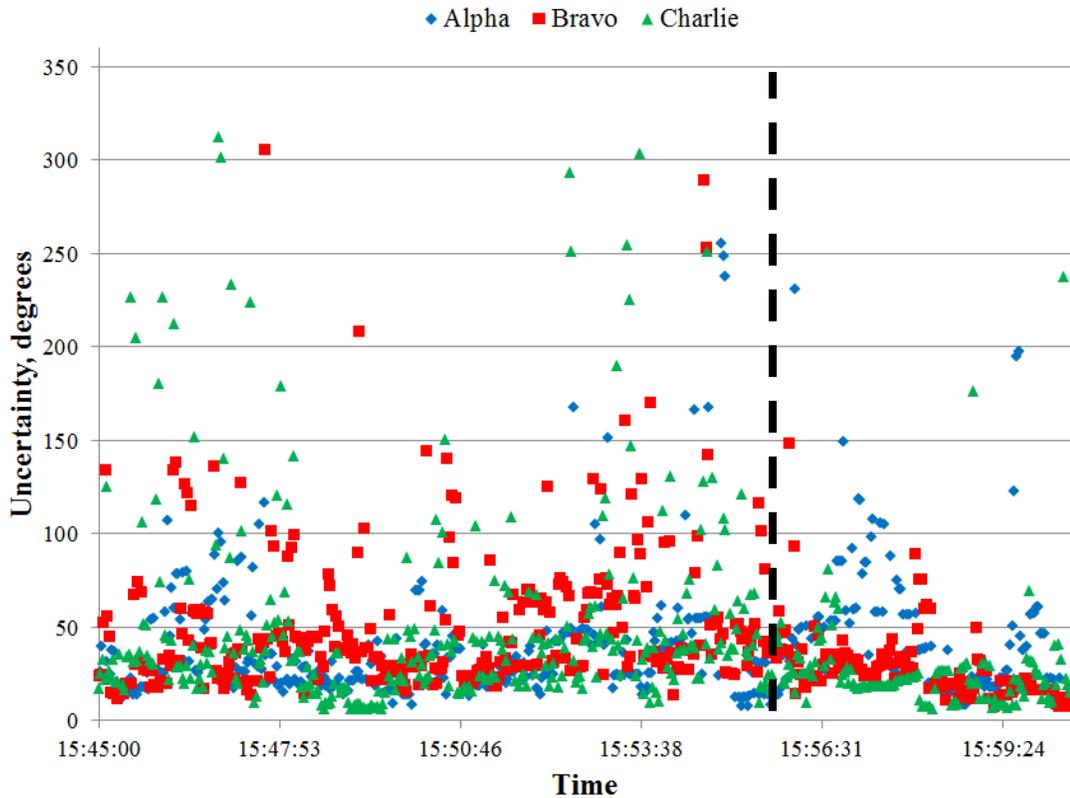


Figure 5.32: Sept 10<sup>th</sup> uncertainty using raw data

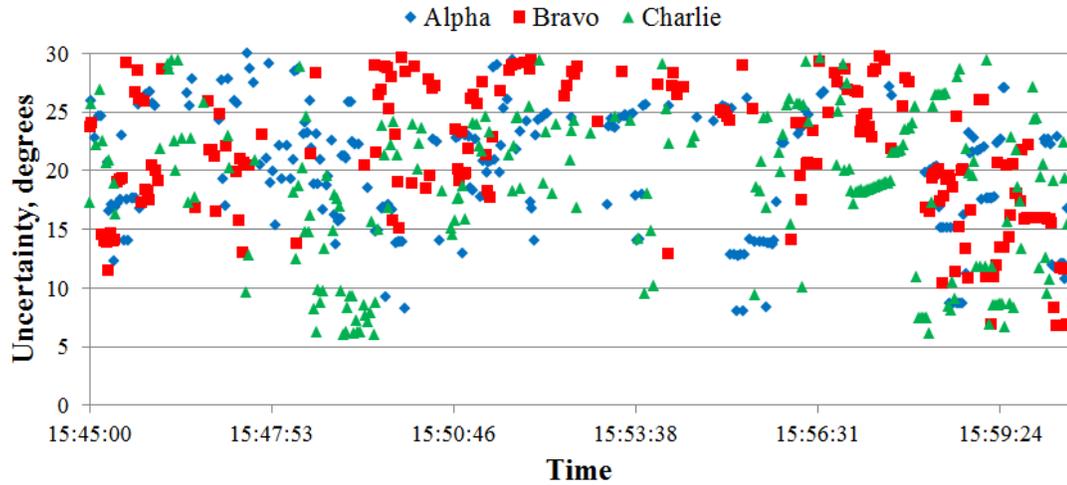


Figure 5.33: Sept 10<sup>th</sup> uncertainty using raw data close-up view

Figure 5.33 shows the minimum uncertainty during this thermal event only reaches 6°, and only for a few data points on sensor Charlie. Sensor Alpha reaches a minimum of 12° before the thermal, while Bravo reaches 8° before and during the found thermal. On average the uncertainty was roughly 45° during this event for each sensor.

The data for June 1<sup>st</sup> was also examined for the uncertainty. Similarly, the uncertainty is sporadic and only reaches areas under 10° on a few points during data collection (Figure 5.34).

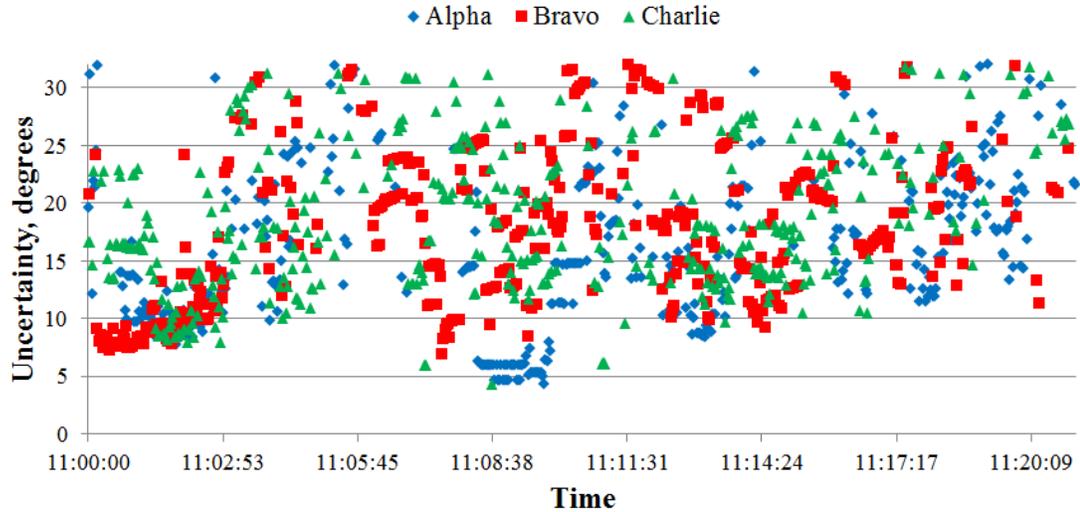


Figure 5.34: June 1<sup>st</sup> uncertainty using raw data, close-up view

#### 5.4.4 STATISTICAL WIND DATA

With actual field data being accessible, it was decided to analyze the statistics of the wind data being recorded at the OSU Flight Field. The following is an account of the methods and results of looking at the statistical quantities that are observed when taking an average wind value for use in the thermal detection angle.

It was necessary to calculate 2 separate standard deviation with the wind data: one being the standard deviation of the measured wind speed, and the second for the wind direction. The calculation for the wind speed would require normal standard deviation equations for a population<sup>49</sup>.

$$\sigma_{speed} = \sqrt{\frac{\sum (x - \bar{x})^2}{n}} \quad \text{Eq. 5.10}$$

However, for the direction standard deviation to be calculated, a different approach was necessary. As with the vector notations discussed in Chapter 3 with separating the vector into  $x$  and  $y$  components, the standard deviation would also need to incorporate a similar approach to account for circular nature of the wind direction ( $0^\circ$  and  $360^\circ$  being the same value). With multiple methods existing for calculating circular standard deviations, the Yamartino method was utilized<sup>40, 41, 50, 51</sup>. The Yamartino method is widely used weather statistics as well as capable of being calculated in a single pass through the data. This would be an advantage if the need arose to incorporate this calculation within the GUI. To utilize the Yamartino method, the unit vector  $x$  and  $y$  components would need to be calculated for the data set.

$$Sa = \frac{1}{n} \sum_{j=1}^n \sin(\theta_j) \quad \text{Eq. 5.11}$$

$$Ca = \frac{1}{n} \sum_{j=1}^n \cos(\theta_j) \quad \text{Eq. 5.12}$$

With the components calculated, the standard deviation ( $\sigma_\theta$ ) could be found utilizing Eq. 5.13.

$$\sigma_\theta = \sin^{-1} \left[ 1 + \left( \frac{2}{\sqrt{3}} - 1 \right) \epsilon^3 \right] \quad \text{Eq. 5.13}$$

Where

$$\varepsilon = \sqrt{1 - (Sa^2 + Ca^2)}$$
 Eq. 5.14

This allows for the standard deviation of each directional data set to be calculated.

Looking at the previously discussed thermal event on Sept 10<sup>th</sup>, the standard deviations within the measured wind data can be visualized. First a look at the standard deviation within the wind speed calculation.

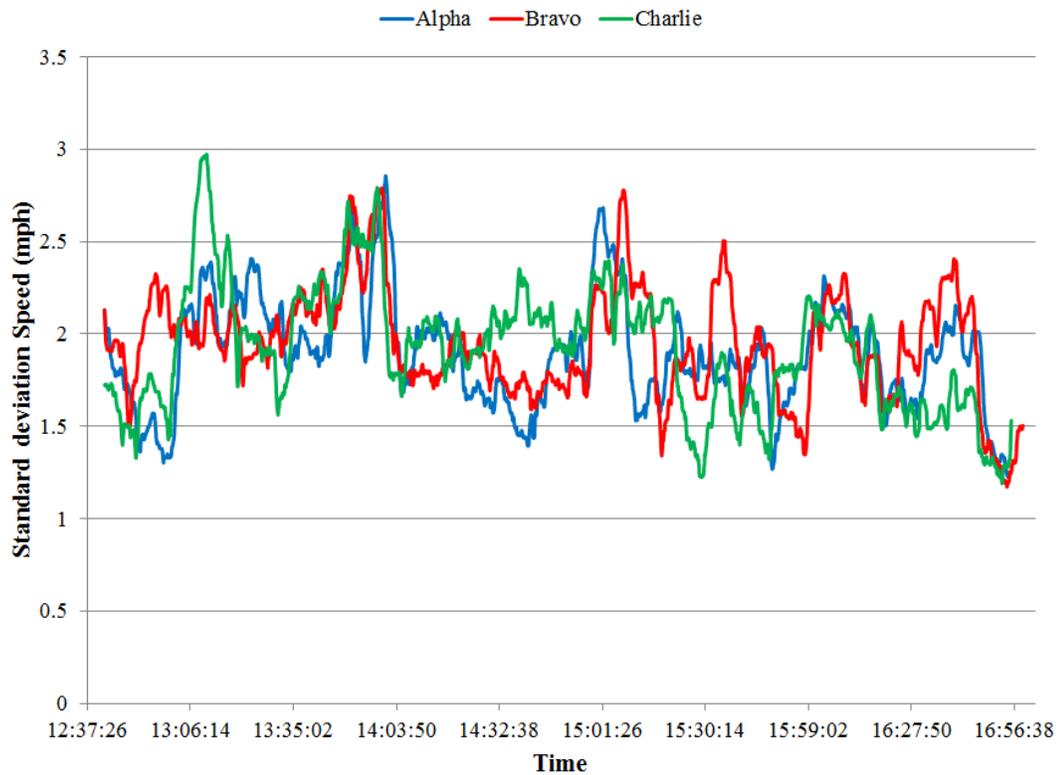


Figure 5.35: Sept 10th standard deviation in measured speed

Figure 5.35 shows the standard deviation for each sensor's speed measurement during the entire day's worth of data collection, where the standard deviation is for the 300 data points being used in the average vector calculation. The trend of the speed is to be around 2 mph for all 3 sensors.

The direction standard deviation showed a different trend.

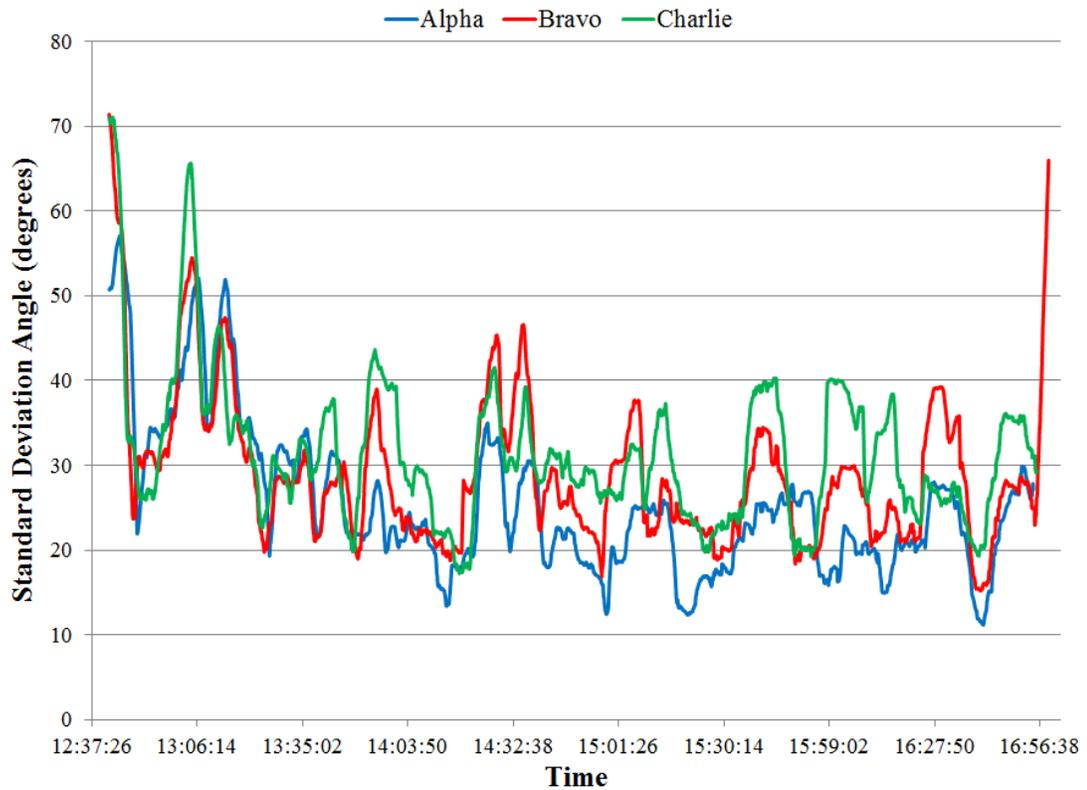


Figure 5.36: Sept 10th standard deviation in measured angle

The values of the directional standard deviation show a high fluctuation. With a minimum value of  $11^{\circ}$  and a maximum of  $71^{\circ}$ , these values average around  $30^{\circ}$  for the day of data collection.

When taking a closer look at the previously mentioned thermal event on Sept 10<sup>th</sup>, Figure 5.37 and Figure 5.38 show the 15 minutes surrounding the event.

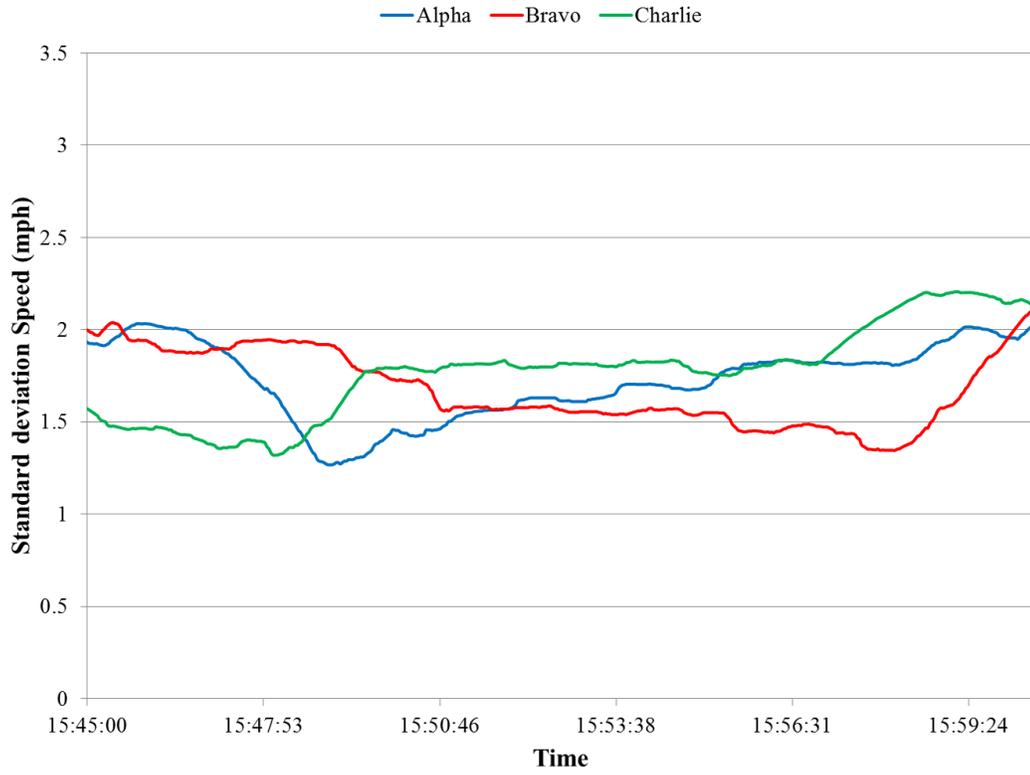


Figure 5.37: Standard deviation of wind speed during thermal event

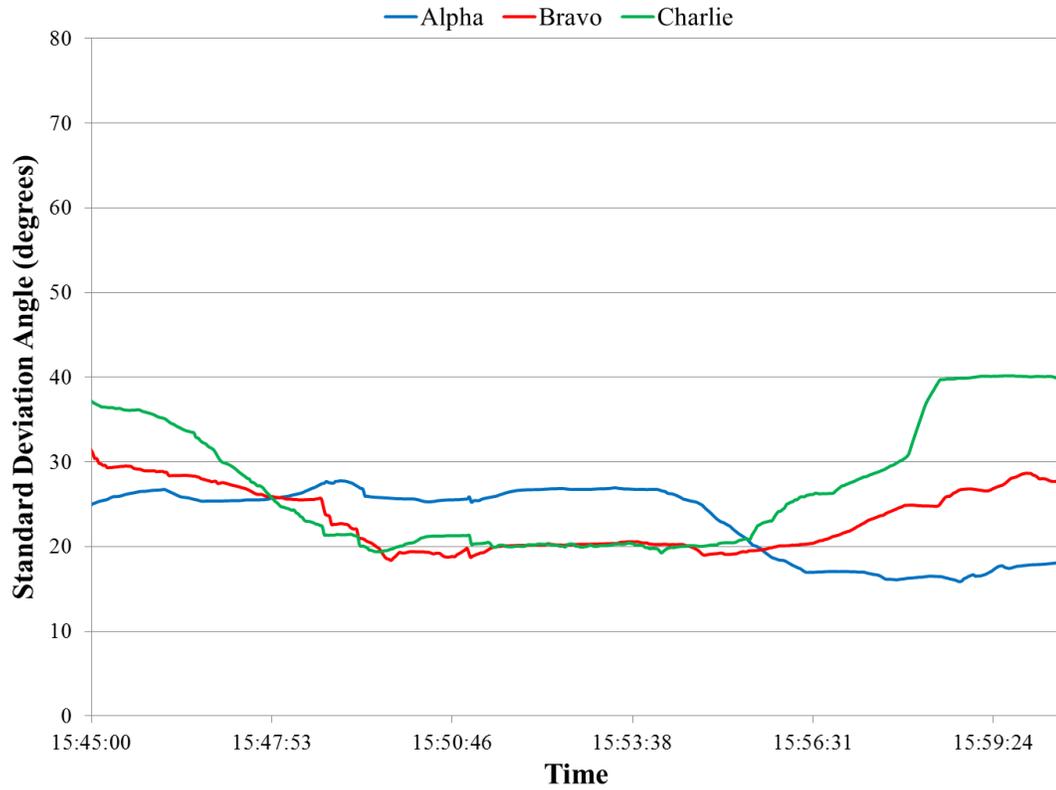


Figure 5.38: Standard deviation of wind direction during thermal event

After looking at the standard deviation within the used average vector, the standard deviation was also calculated with different population sizes to determine if the standard deviation could be lessened.

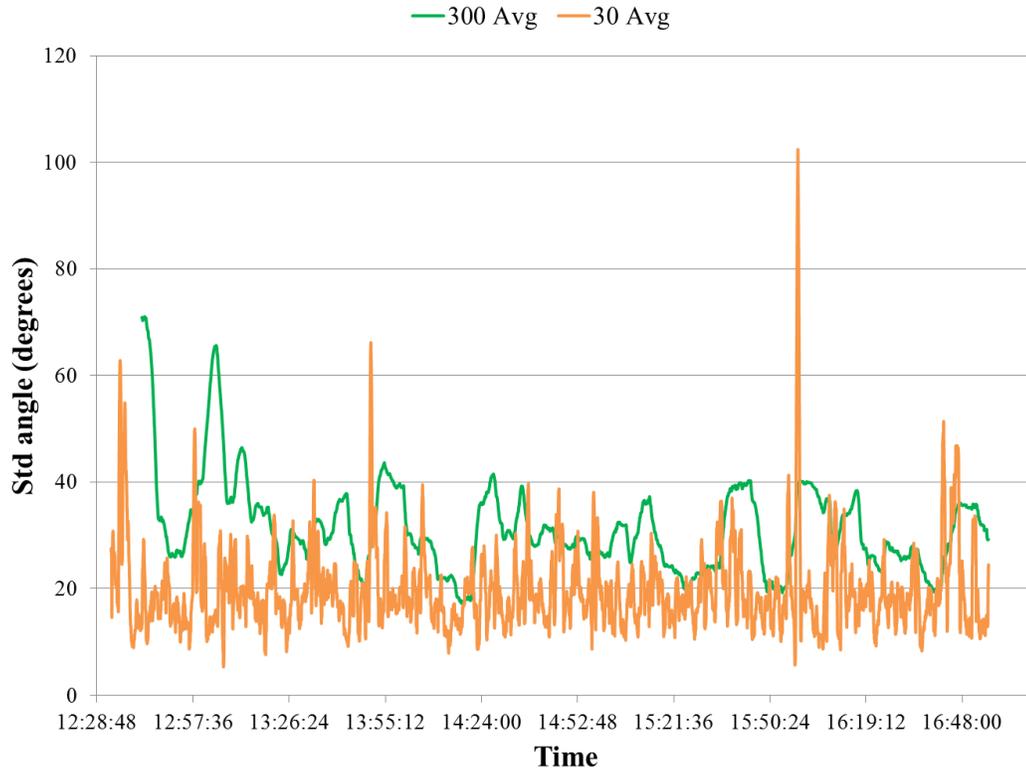


Figure 5.39: Sept 10th difference in standard deviation of single sensor (Charlie) using different average lengths

Using a lower population value lowered the overall average standard deviation for the entire data set, but also created more variation within the calculated average vector.

#### 5.4.5 ADDITIONAL SIMULATIONS

Additionally, the simulations that were developed for initial testing of conceptual experimentation, as explained in Chapter 3, were revisited to further analyze the uncertainty of using the thermal locating technique. These modified simulations were to be used to further analyze the effects of sensor accuracy, wind gusts, as well as the necessary conditions for this method to be feasible.

Initially the simulations were modified to incorporate a random perturbation into the measured wind factor felt by each modeled sensor node. This random perturbation was to simulate the uncertainty of the sensor components or that of wind gusts that were derived from previous field experiments.

As mentioned in Chapter 3, the simulations would take a known uniform wind, with a modeled thermal vector, to calculate a simulated sensor vector. From this vector, the thermal direction was then calculated by subtracting an average vector which was calculated from an array filled with previous sensor vectors.

The random component was

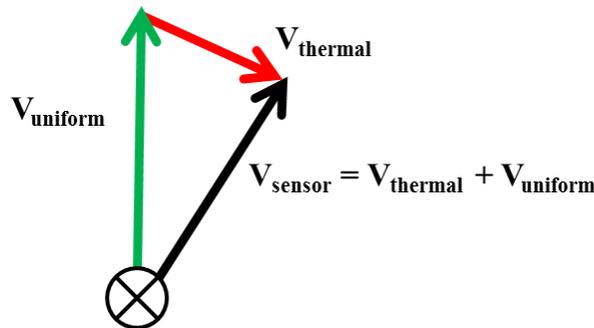


Figure 5.40: Vector components of simulation

To add uncertainty to the sensors, random values were added or subtracted from the speed or direction of the uniform wind component. A uniform wind component was still specific in the simulation, however now the value shown and used for all subsequent calculations would incorporate a random factor. To incorporate the random factor, the  $x$  and  $y$  components of the uniform wind vector would have a random value between 0 and the specified uncertainty added to the respective components (Eq. 5.15).

$$\begin{aligned} V_{average,x} &= (5mph \pm 1.2mph) \cos(0^\circ \pm 6^\circ) \\ V_{average,y} &= (5mph \pm 1.2mph) \sin(0^\circ \pm 6^\circ) \end{aligned} \quad \text{Eq. 5.15}$$

To incorporate this into the simulation, the uniform wind calculations were modified to add a random value between the negative and positive uncertainty value.

```
Vx = (WindSpd+(random(-1,1)*SpdR))*(cos(radians(WindDir+90+random(-DirR,DirR))));
Vy = (WindSpd+(random(-1,1)*SpdR))*(sin(radians(WindDir+90+random(-DirR,DirR))));
```

Figure 5.41: Code of uncertainty being added in simulation

In the case of incorporating a random speed component, a single pulse was added or subtracted resulting in a change of 1.2 mph to the magnitude of the wind speed.

Initially the simulations were ran with the UAS Airfield details incorporated to compare the effects of the added uncertainty with airfield experiments. The simulated sensors were placed in locations similar to that of the airfield experiments with a simulated thermal added in the Southeast corner of the field (Figure 5.42)



Figure 5.42: Simulation with added uncertainty

These simulations were run multiple times with different sink strengths to determine the required strength to accurately determine the location of the simulated thermal. For the simulation, an average value was calculated using 300 data points of the modified uniform wind component, with the thermal component being incorporated for only 30 data points. During the period of the simulated thermal, the number of thermal detections from the sensors were calculated out of the 30 possible occurrences. Thermal detection from the sensors was recognized when the 3 independent thermal indicators would triangulate on or near the simulated thermal location.

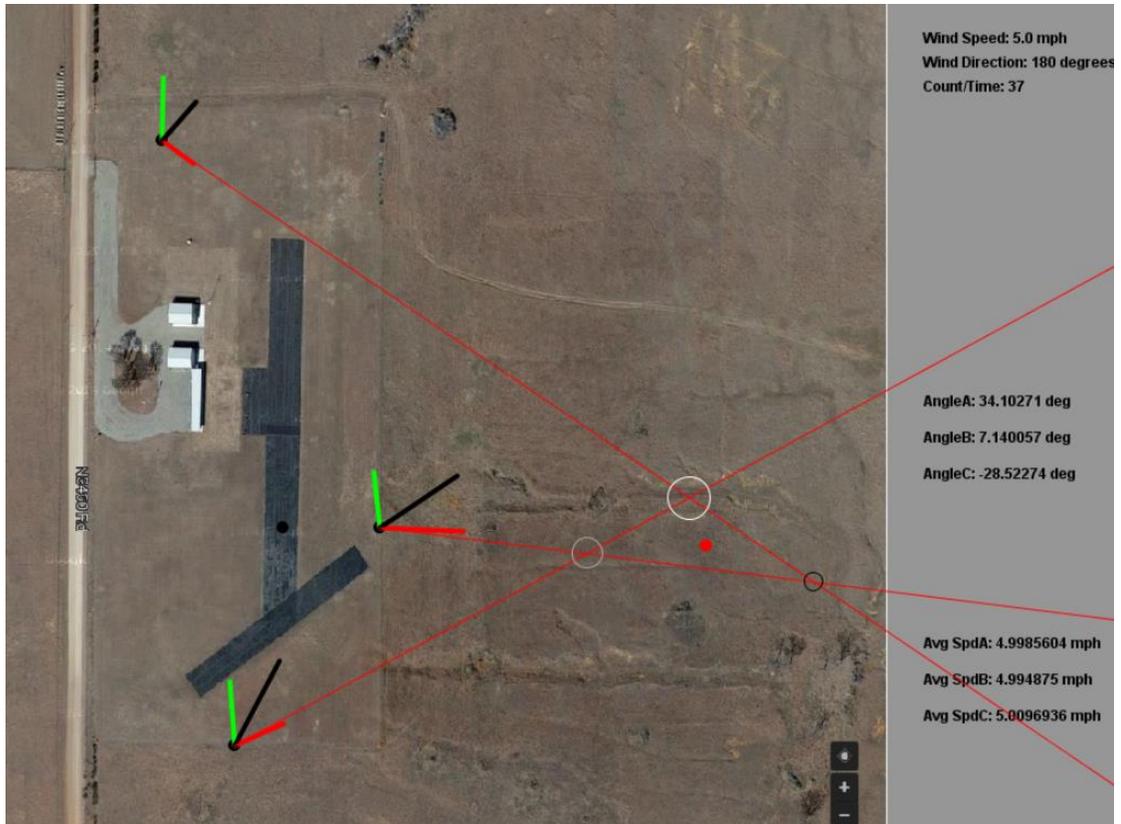


Figure 5.43: Simulation with added uncertainty with triangulation of thermal event

Each sink strength value was run 10 different times and then averaged to get an estimated percentage value of the accuracy of the detection for each strength value. From the initial runs, Figure 5.44 shows the percentage of detections using a simulated airfield configuration.

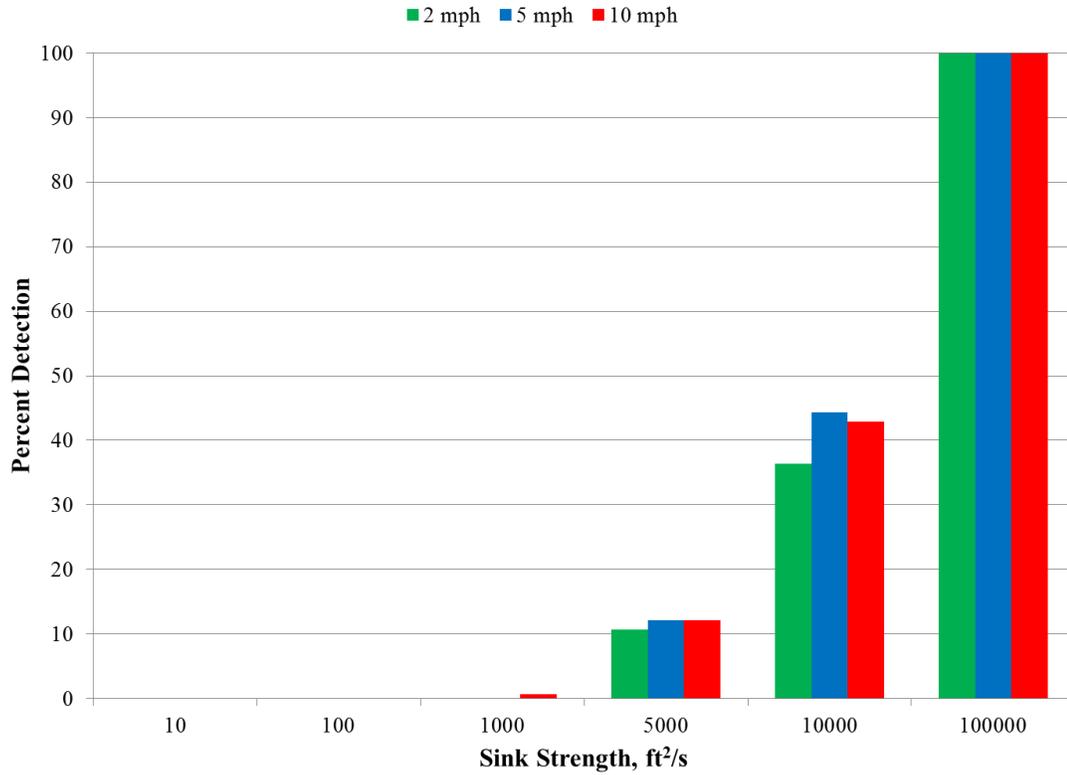


Figure 5.44: Histogram of initial detections vs sink strength

To further examine the effect of thermal strength, the modeled configuration was changed to incorporate a constant distance for each sensor to better analyze the affect that distance from the thermal plays in the calculations. To better determine the number of detections during the thermal activity of the simulation, additional algorithms were added to automatically determine when the simulated triangulation was within a certain distance from the center of the thermal (Figure 5.45).

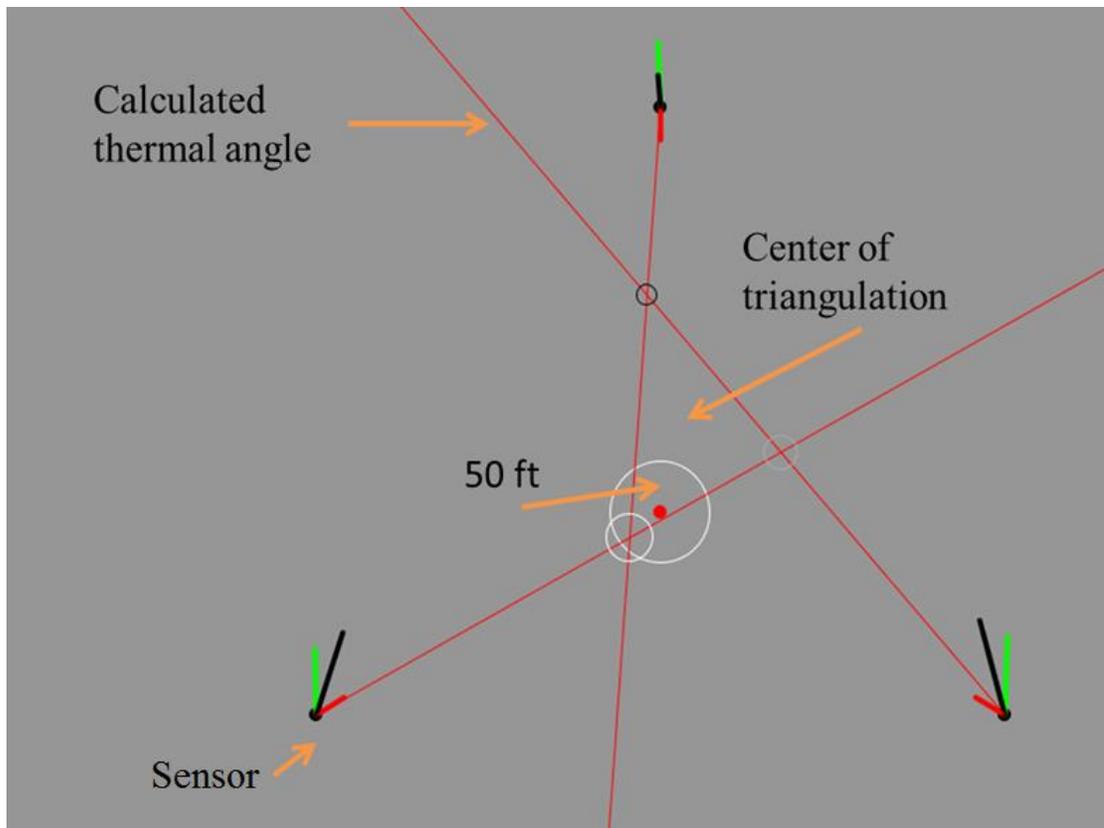


Figure 5.45: Optimal simulation setup

For the initial runs, a radius of 50 ft was used to count as an accurate detection range. If all 3 thermal identifiers intersected, the simulation would calculate the center of the intersection and determine whether it was within 50 ft of the thermal center. If the triangulation was within the 50 ft range, a counter would be updated for the individual simulation. Once again, each simulation was ran 10 times to allow for an average number of detection values for each sink strength value. Along with changing the strength value, the distance of each sensor was increased over the different simulations. Figure 5.46 shows the detection percentage with change in sink strength for multiple distances from the sink location using a 5 mph uniform wind speed.

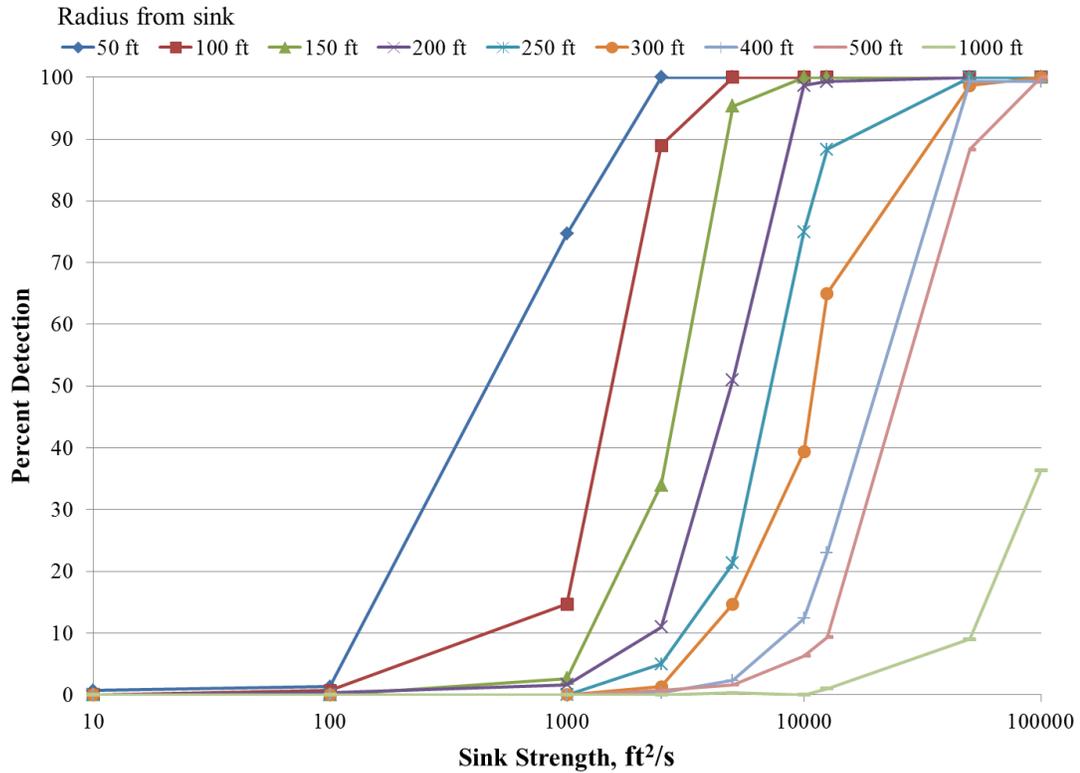


Figure 5.46: Detection vs sink strength with 5 mph uniform wind and sensor uncertainty values

As can be seen in Figure 5.46, the number of intercepts increases as the sink strength or as the thermal strength increases. Likewise the number of intercepts also increases as distance from the thermal decreases thus showing a correlation between the detectability versus strength and proximity to updraft.

This relationship better illustrates the correlation from the vector uncertainty in the previous section, where the uncertainty values become minimum when the instantaneous vector and average vector showed the most difference. For the vectors to show the greatest difference, there would need to be a strong thermal component, which would be caused by a strong thermal strength or a close proximity to the thermal event.

Additional simulation were ran with other values of uniform wind speed of 2 and 10 mph (Figure 5.47 and Figure 5.48 respectively) which also showed the same trend.

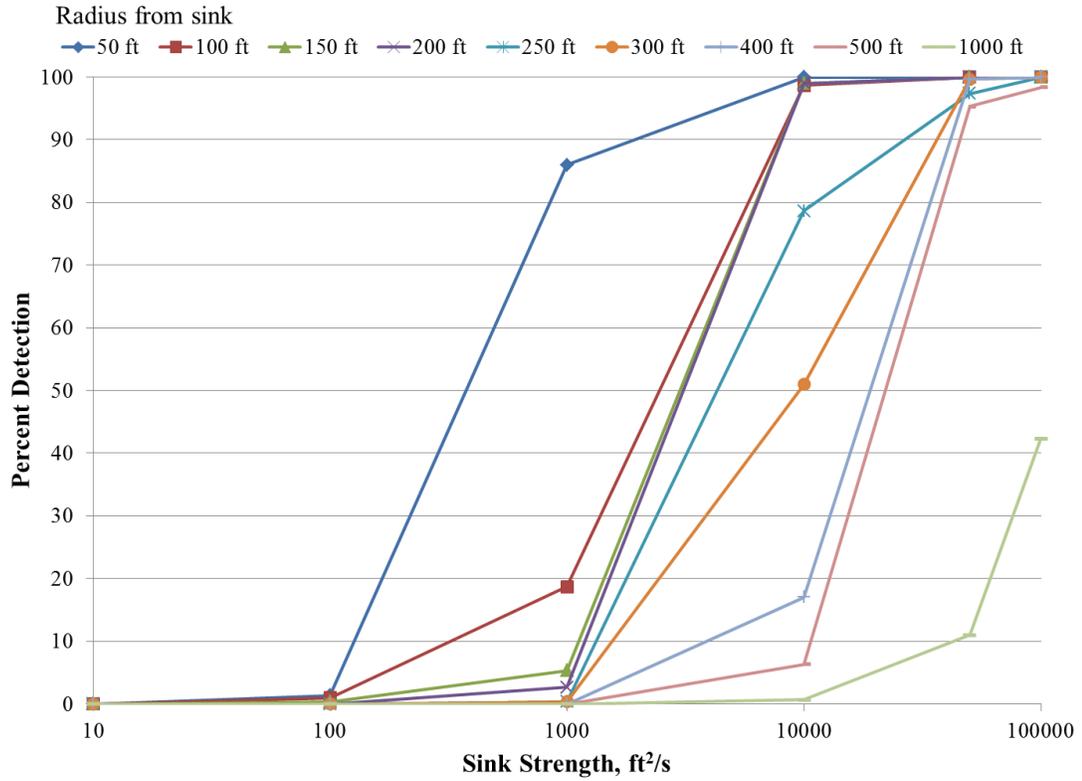


Figure 5.47: Detection vs sink strength with 2 mph uniform wind and sensor uncertainty values

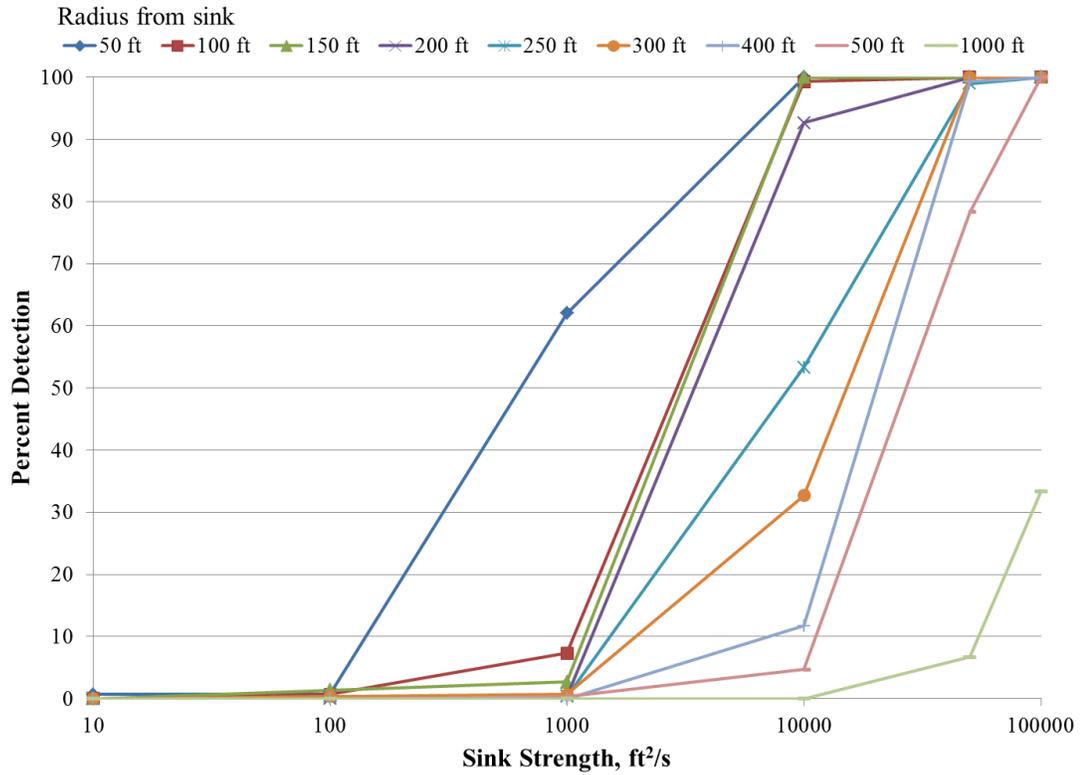


Figure 5.48: Detection vs sink strength with 10 mph uniform wind and sensor uncertainty values

Figure 5.46, Figure 5.47, and Figure 5.48, used a random perturbation similar to that of the minimum uncertainty assessed by the individual sensors. From the previous data sets, the standard deviation of experimentally collected wind data was calculated. These wind statistical values were then plugged into the simulations to analyze the effect of greater uncertainty: in this case the uncertainty due to wind gusts. From the statistical discussion in Section 5.4.4, a wind speed value of 2 mph with an angle value of 30° was used in the simulation. Figure 5.49 shows the effect using these augmented values with a uniform wind speed of 5 mph.

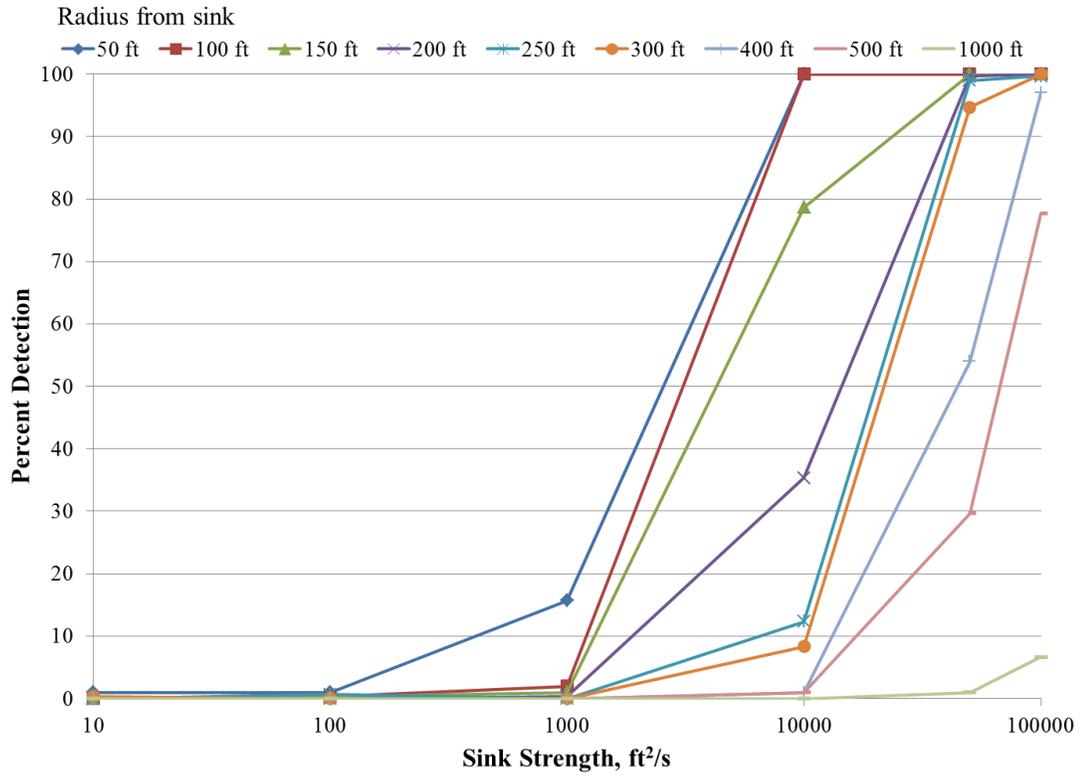


Figure 5.49: Detection vs sink strength with 5 mph uniform wind and gust uncertainty values

Again, the trend is similar: detection increases with an increase of sink strength as well as increases with a decrease in distance from sink location. Using a higher uncertainty value, the curves are shifted to the right. Figure 5.50 shows the comparison between the two used uncertainty values for the 5 mph uniform wind profile.

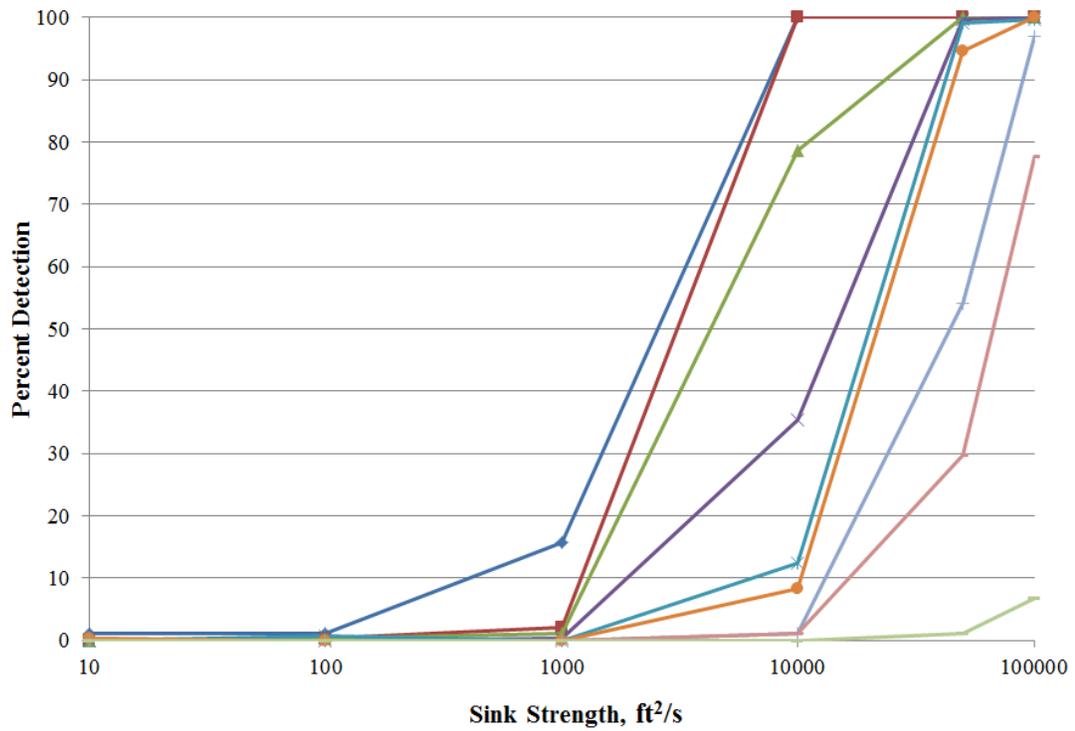
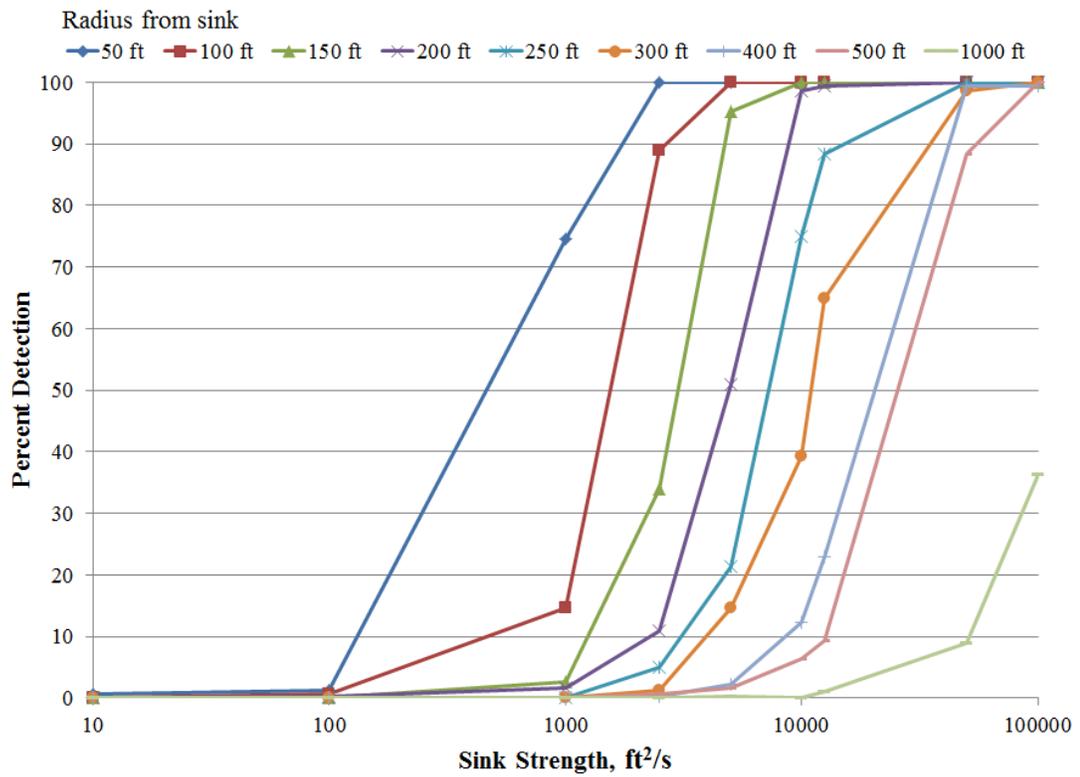


Figure 5.50: Detection vs sink strength comparisons of 5 mph uniform wind with sensor uncertainty (top) and gust uncertainty (bottom)

The simulations were modified once again to better look at a physical thermal, instead of a generic sink strength value. From Chapter 3, when discussing sink strength relation with physical thermal updraft dimensions, multiple thermals could be attached to a single sink strength value (Figure 3.6). For this modification, the sink strength was replaced with a velocity equation derived from conservation of mass calculations used in Chapter 3 (Eq. 3.15).

Similar to the detection method used in the previous simulations, the simulation would look for instances when the triangulation was within a specified location. For these new simulations, the specified location would be within the programmed thermal radius. As with the previous simulations, the uniform wind was set to a nominal 5 mph.

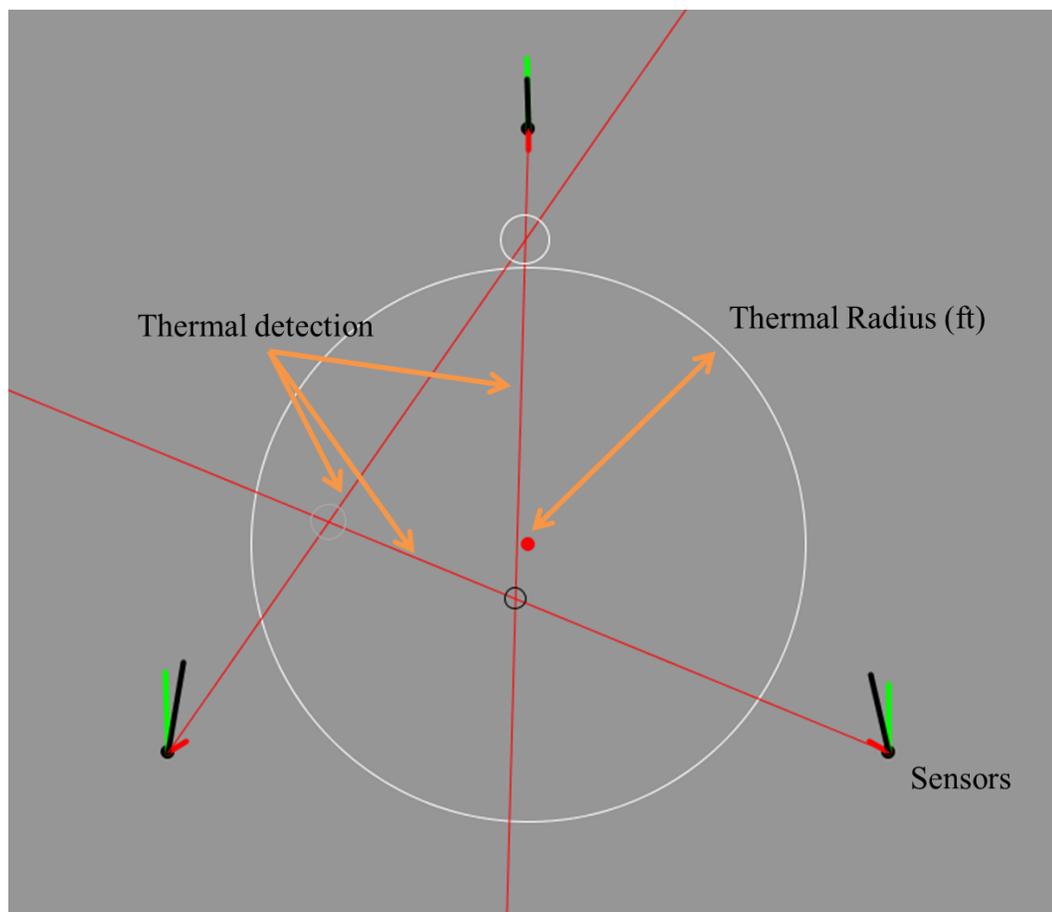


Figure 5.51: Simulation setup with thermal components

Multiple updraft velocities and thermal radii were simulated. Figure 5.52 shows the detection percentage when compared to the ratio of the radius from the thermal and the thermal radius.

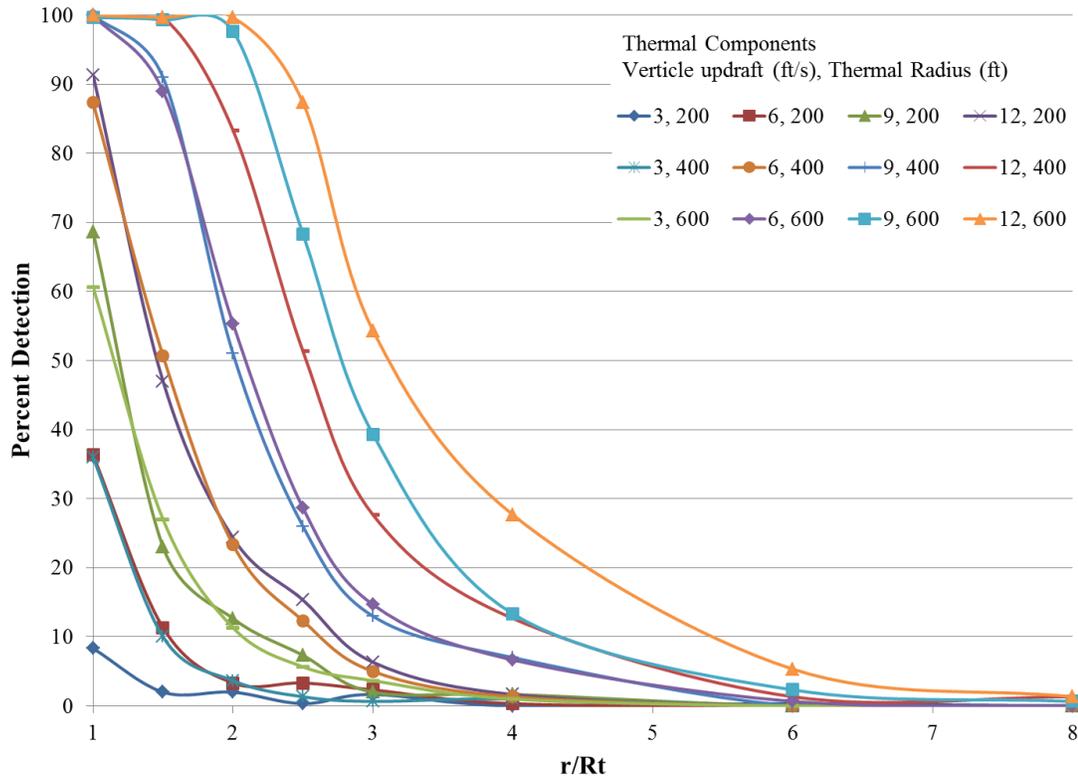


Figure 5.52: Detection vs  $r/R_t$  ratio for multiple thermal sizes

Once again, the trend of detection increasing with either the increase in thermal strength (increasing updraft velocity and/or radius) or with the decrease in distance (lower  $r/R_t$  ratios) can be seen.

Additionally, both the airfield simulation and optimal simulations were ran with no thermal component. This allowed for the simulation to calculate the shift direction based on the uncertainty pertaining to using only the uniform wind. When running these simulations, the

thermal indicators would change sporadically during the run, and would even give false positive locations of thermal updrafts (between 3 to 4% detection of all 3 sensors for the tested cases). This sporadic motion resembled much of what was seen from the recorded data during data collection at the airfields.

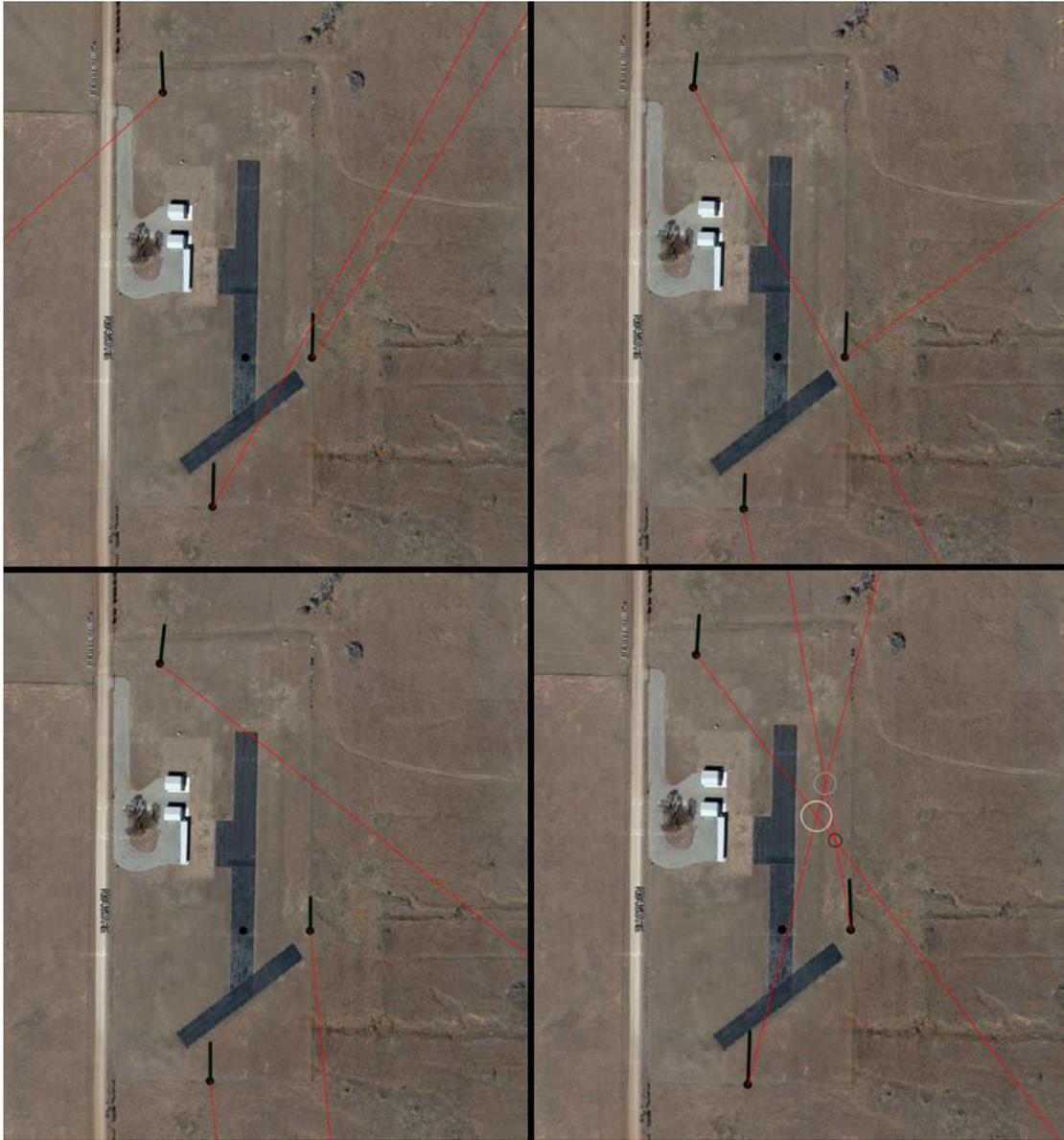


Figure 5.53: Indicators due to uncertainty with no thermal component

#### 5.4.6 COMBINED UNCERTAINTY ANALYSIS

Using the combined information from the minimum uncertainty within the sensors along with the statistical uncertainty within the average wind vector from the experimental data sets, a total uncertainty was examined. This new uncertainty analysis was to take a separate uncertainty level for each of the individual parts to make the components more representative of the uncertainty the thermal detection method would transpire. Using Eq. 5.9, a new uncertainty analysis was ran were now the uncertainty values were representative of the amount of uncertainty for the instantaneous and average vectors. The instantaneous vectors used the sensor, or minimum uncertainty values of 1.2 mph and 6°, while the average vector would use the calculated standard deviation data for the 300 data points used for the average vector.

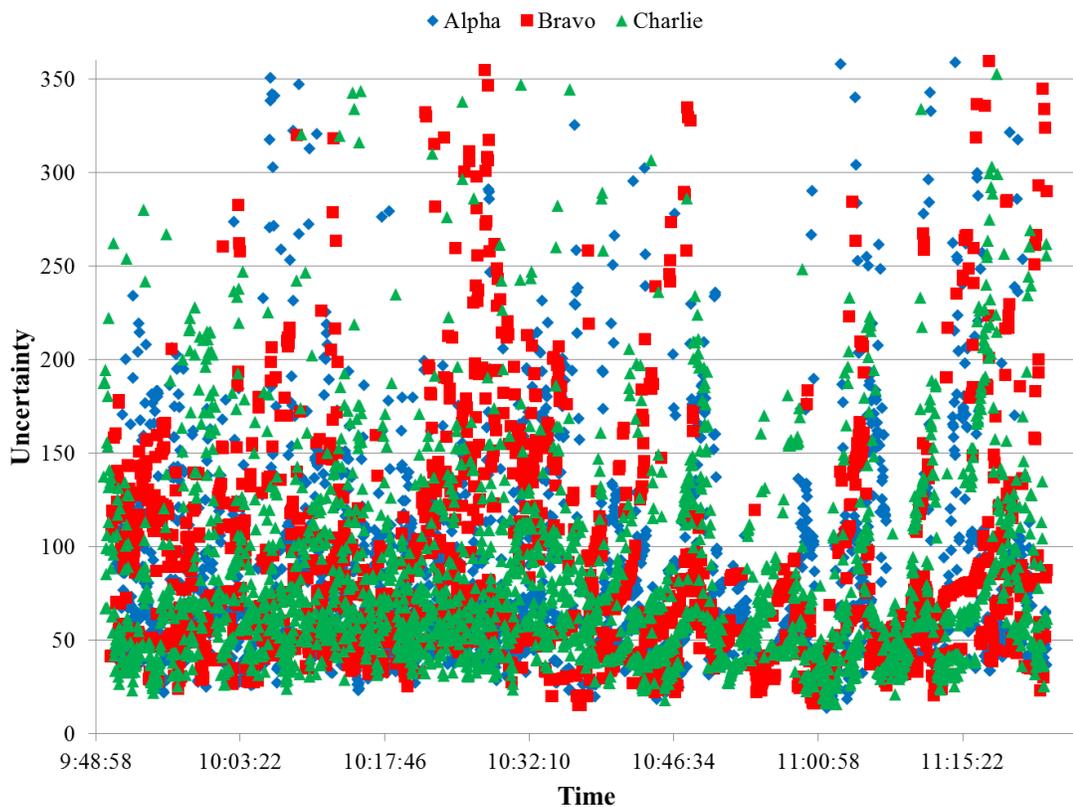


Figure 5.54: Total uncertainty for June 1<sup>st</sup>, 2015 thermal event  
162

Figure 5.54, as expected, shows an even high amount of uncertainty values for calculating a thermal direction with the data that was collected for June 1<sup>st</sup>, 2015. The maximum values for the uncertainty went far beyond 360°.

Table 5.3: Uncertainty ranges for June 1st, 2015

	Alpha	Bravo	Charlie
Maximum	6184	4737	3927
Minimum	13.6	15.3	15.7
Average	102.4	97.6	90.9

Taking a closer look at the thermal even during this day, the uncertainty levels are still high.

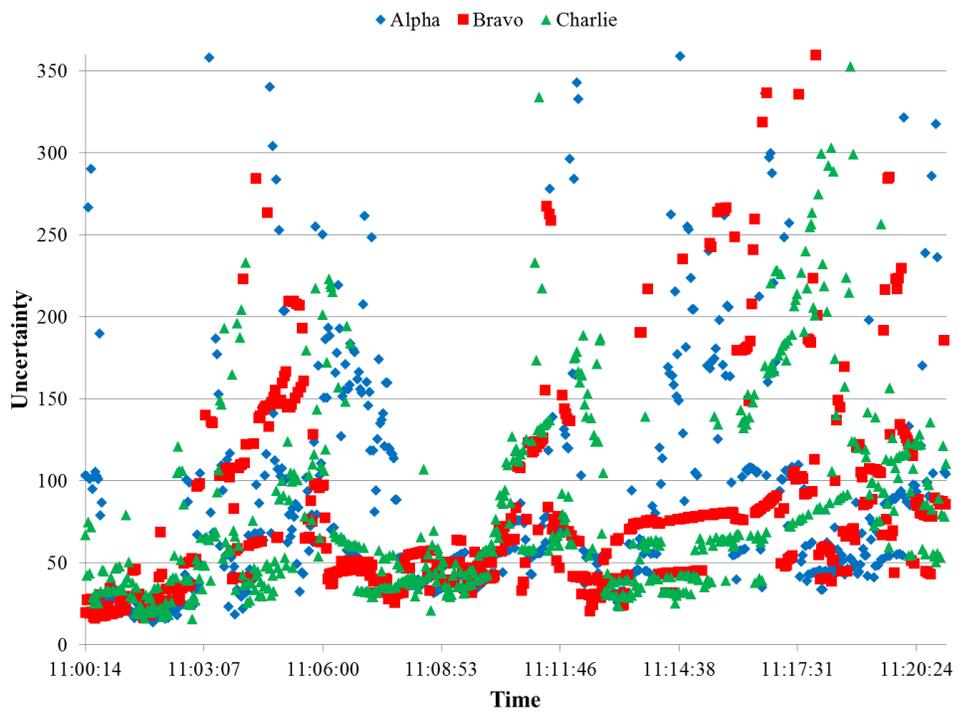


Figure 5.55: June 1<sup>st</sup>, 2015 thermal event uncertainty levels

Running a similar analysis for September 10<sup>th</sup>, 2015 shows a similar high trend of uncertainty values during the entirety of data collected that day, including the time surrounding the thermal event.

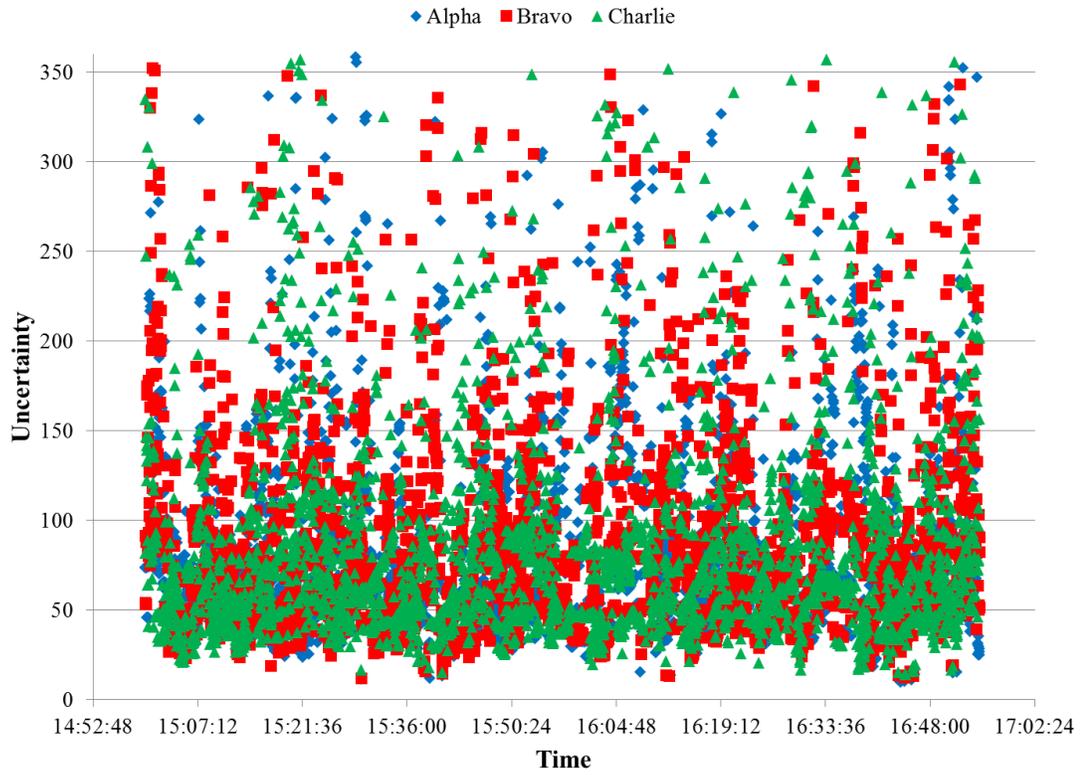


Figure 5.56: September 10th, 2015 uncertainty values

Table 5.4: Range of uncertainty for September 10th, 2015

	Alpha	Bravo	Charlie
Maximum	1483	4155	7049
Minimum	9.9	12.0	14.2
Average	89.5	103.3	110.6

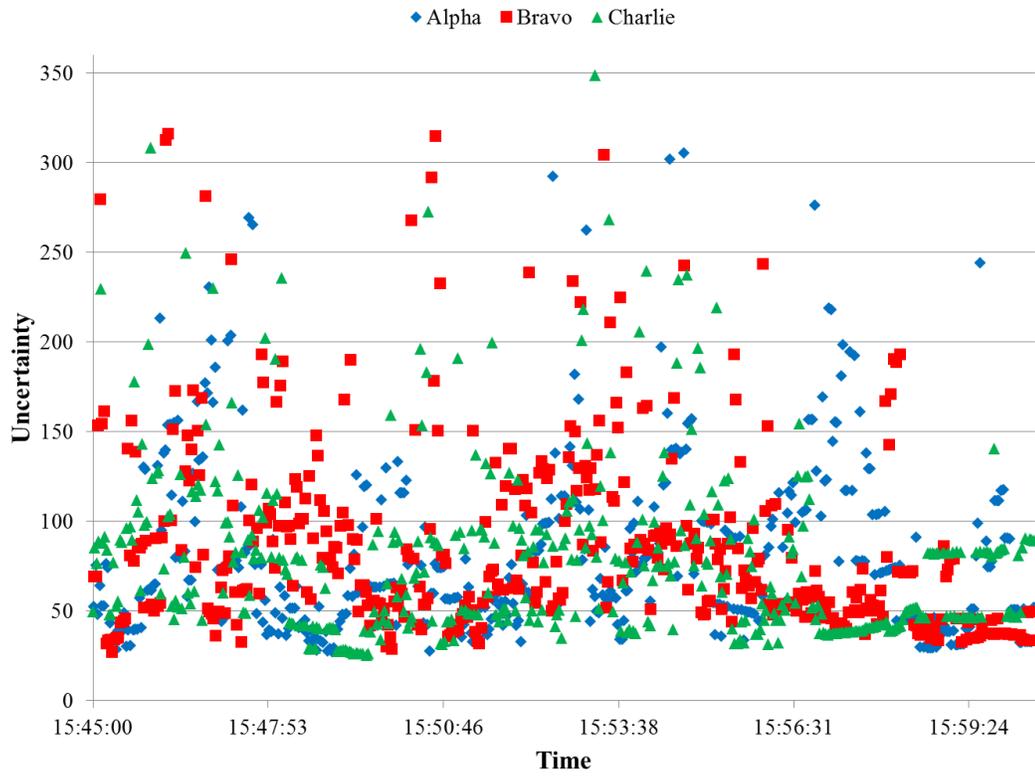


Figure 5.57: Uncertainty around September 10th, 2015 thermal event

## CHAPTER VI

### 6 CONCLUSIONS AND RECOMMENDATIONS

A method for detecting thermal updrafts, using theories gathered from an avian perspective, was examined through simulation and experimentation. Through the use of data collected around localized thermal updrafts, the idea of quantifying the changes in local wind conditions due to inflow from thermal updrafts was examined. In this chapter, conclusions and recommendations will be discussed based on the results and background information that has been discussed previously.

#### 6.1 CONCLUSIONS

When comparing the methods used by birds and human aviation, the method of incorporating feeling shifts of air properties was explored in the hopes for a better understanding of thermal predication for future thermal endeavors. The following derived conclusions will mirror each of the objectives that were portrayed back in Chapter 1. Conclusions will detail information gleamed from each of the objectives that were completed during the course of this dissertation.

CONCLUSION 1: No concise answer to how birds detect thermal updrafts. Through the literature review process however, 5 theories were found. A bird's method of detecting thermal updrafts could be through memory of thermal locations, seeing objects in thermals, sensing changes in surround air properties, actively searching for updraft components, or through an unknown extrasensory ability specific to avian biology. Again, there currently is no consensus on which of these methods is the primary or single method employed by soaring birds. Human aviation employs the use of spotting objects within or around thermals, knowing the terrain of where thermals may appear, or by feeling the updraft or downdraft of a thermal

CONCLUSION 2: Plausible to detect shifts in local wind conditions given ideal conditions. This objective was to develop computer simulations to show the possible effects thermal updrafts have on surrounding air masses. Using a simple sink and uniform wind components along with individual averaging methods, a thermal vector was able to be calculated which pointed towards the area of inflow. These initial simulations also visually showed how the "third vector" approach could be utilized under ideal conditions with no sensor or statistical uncertainty.

CONCLUSION 3: Viable in recording local atmospheric conditions for research use. The constructed sensors were built, tested, and utilized during this research endeavor. The methods utilized allowed for data collection through data storage and real-time visualization of the data. As seen in the results section, the sensors were capable of recording data regarding wind speed and direction, temperature, relative humidity, as well as barometric pressure. Though created for the indent of locating thermal updrafts, these sensors have the capability for being used for mobile weather stations for additional research data collection. The sensors were developed to allow modifications to be made easily in both the hardware and software aspects.

CONCLUSION 4: Additional simulations showed a large moving average could work with keeping track of updraft locations for a time under certain conditions. However, when adding additional uncertainty to the defined and calculated wind components of the simulation, the

thermal location was only plausible given the thermal component had a high strength or was in close proximity to the sensors.

CONCLUSION 5: Excluded from research. Due to the lack of thermal identification with the current method and the high uncertainty that was found from the previous results, additional experimentations involving UAV data collection for comparison during flight was not undertaken for this current endeavor.

CONCLUSION 6: From the combination of the seen results with the uncertainty analysis, it is unlikely that avian species primarily use feel as a method of detecting thermal updrafts. Even though the anecdotal information from manned and RC flight show strong evidence for this method, additional methods are used to pinpoint the thermal location such as monitoring RC aircraft to watch for signs of thermal activity. To quantify this method for use in the field would require additional information and more precise equipment to minimize the current uncertainty level. Even with minimizing uncertainty by updating the wind sensors, the uncertainty still has a massive value when the average vector is close to the current wind vector. It is in those cases, that the thermal vector can no longer be calculated with certainty. Ideal conditions would be when the thermal is very close to the sensors or the thermal has a strong inflow. It is in either of these two cases that would cause the difference in the average vector and current vector to increase thus decreasing the uncertainty. These uncertainties were only the minimum found when using this method. If the sensor uncertainty was removed, i.e. perfect sensors, there would still be uncertainty due to wind gusts or the wind's statistical uncertainty that would propagate uncertainty within the thermal detection method. Remotely locating thermal updraft with certainty is still an unknown phenomenon

## 6.2 RECOMMENDATIONS

The following will outline and discuss recommendations for continuation of research discussed in this dissertation. The recommendations consist of upgrades to the current

experimental setup for further experiments, as well as discuss future research avenues based on the conclusions made previously.

#### 6.2.1 ADDITIONAL EXPERIMENTATION

With regards of furthering the research of using the “third vector” approach, additional research would be useful in ways to minimize uncertainty levels within the experiment. First, it is recommended to find areas with ideal conditions to test sensors and methodology at the minimum uncertainty levels: with the ideal conditions being flat landscape with larger thermals in the defined area. Under more ideal conditions, there would be more opportunity for the difference between the average and instantaneous vectors to be the greatest and thus allow for the uncertainty of the experimentation to be at its lowest levels based on the sensor uncertainty.

In order to help further decrease the uncertainty when calculating the thermal direction, higher resolution wind speed and directional sensors should be incorporated to lower the sensor uncertainty. Incorporating a digital anemometer capable of producing additional pulses per rotation would be ideal for better resolution in low wind speed applications. However, additional methods would need to be explored in order to decrease the statistical uncertainty in the measured wind. New and improved averaging techniques, or inclusion of uncertainty analysis within the graphical interface could be used minimize the uncertainty when calculating thermal locations based on this prescribed method. Additional research in methods of minimizing the uncertainty within this approach would help detecting close updrafts.

#### 6.2.2 NEW EXPERIMENTATION

From the found avian hypothesizes, only one method was tested for this project. The other methods could also be further explored and tested. Testing the feasibility with memory, sight, and search techniques are still other avenues to research for application with UAVs. The search and memory techniques have been and are being used in previous and current UAV research as discussed in Chapter 2. However, additional experimentation with regards to sight is another option. Looking into information regarding detecting items within the updraft from a

distance could serve as a plausible detection method of UAVs. Another option would be to determine what, if any, could be found using ultraviolet instrumentation in regards to thermal updrafts. As birds see closer to the ultraviolet spectrum than infrared, additional in site in avian thermal detection could be gathered.

### 6.2.3 SENSOR UPGRADES

Improvements regarding the individual sensor stations are also recommended. These sensor platforms were designed to read local atmospheric conditions and relay the information wirelessly to a central hub. The construction of the current sensors accomplishes such a task, yet using the current platform as a foundation, improvements could be made to enhance the sensor's capabilities for future or additional research endeavors. The improvements mentioned were not implemented based on a combination of the time required and the impact the upgrade would make on the current research outlook.

#### 6.2.3.1 XBEE COMMUNICATIONS

One upgrade that could help increase the longevity of the sensors is to incorporate a sleep mode with the XBee radios. The XBee radios require a major portion of the power requirements of the sensors. Initializing the sensors as end nodes would allow the XBee radios to power down intermittently to a lower power level while not transmitting data back to the coordinator.

Another upgrade to possibly incorporate with the sensor network is reconfiguring the XBee radio topology to allow the sensors or router XBees to communicate not only with the coordinator radio but also with each other. This would require the XBee radios to be maintained as router nodes. Additional coding of the Arduino sketches along with reinstallation of the XBee firmware to allow additional communication could enhance the information gathered around the field.

#### 6.2.3.2 SENSOR POWER REQUIREMENTS

To also increase the longevity of the sensors, new power methods should be employed. For the power requirements used during data collection, a single 9 V battery was used. However,

even though this was the easiest to incorporate, it was also the least efficient. Incorporating a power converter with a 9 V or other battery supply would be better equipped to lengthen the power reserves.

The power needs could also be supplied by a solar array on each sensor station. This would allow longer functionality during daylight data collection and decrease possible data loss due to changing battery supplies.

#### 6.2.3.3 GPS UPGRADES

For the current sensor setup and interface, each sensor is preprogrammed with its location on the testing field. Adding a GPS unit to either each sensor, or noting the GPS location during setup would help in future data collection. However, adding GPS equipment would also increase the power supply to the sensor. If GPS was to be included, it would also be possible and beneficial to link the recorded data with a mapping service, such as Google earth, to get a more accurate geographical location.

## REFERENCES

1. Stull, R.B., An introduction to boundary layer meteorology. Atmospheric sciences library. 1988, Dordrecht ; Boston: Kluwer Academic Publishers. xii, 666 p.
2. Piggott, D., Gliding: a handbook on soaring flight. 2nd ed. 1967, London,: Black; distributed in the U.S.A. by Barnes & Noble. vii, 263 p.
3. Young, G.S., Turbulence Structure of the Convective Boundary-Layer .3. The Vertical Velocity Budgets of Thermals and Their Environment. *Journal of the Atmospheric Sciences*, 1988. 45(14): p. 2039-2049.
4. Akos, Z., et al., Thermal soaring flight of birds and unmanned aerial vehicles. *Bioinspiration & Biomimetics*, 2010. 5(4).
5. Barringer, L.B. and A. Klemin, Flight without power. 1942, New York, Chicago,: Pitman publishing corporation. ix, 221 p.
6. Irving, F., The paths of soaring flight. 1999, London River Edge, NJ: Imperial College Press; Distributed by World Scientific Pub. xiii, 133 p.
7. Wallington, C.E., Meteorology for glider pilots. 2nd ed. 1966, London,: Murray. xii, p. 302.
8. Cone, C., The theory of soaring flight in vortex shells. Parts I, II, and III. *Soaring*, 1961. 25: p. 4-5.
9. Pennycuick, C.J., Field observations of thermals and thermal streets, and the theory of cross-country soaring flight. *Journal of Avian Biology*, 1998. 29(1): p. 33-43.
10. Wilkins, E.M., Y. Sasaki, and R.H. Schauss, Vortex Formation by Successive Thermals - Numerical Simulation. *Monthly Weather Review*, 1971. 99(7): p. 577.
11. Antal, C., O. Granichin, and S. Levi. Adaptive autonomous soaring of multiple UAVs using Simultaneous Perturbation Stochastic Approximation. in *Decision and Control (CDC)*, 2010 49th IEEE Conference on. 2010.
12. Cone, C.D., Thermal Soaring of Birds. *American Scientist*, 1962. 50(1): p. 180-&.
13. Klesh, A.T., P.T. Kabamba, and A.R. Girard, Optimal cooperative thermalling of unmanned aerial vehicles, in *Optimization and Cooperative Control Strategies*. 2009, Springer. p. 355-369.
14. Qi, Y. and Y.Y.J. Zhao, Energy-efficient trajectories of unmanned aerial vehicles flying through thermals. *Journal of Aerospace Engineering*, 2005. 18(2): p. 84-92.
15. Shannon, H.D., et al., Measurements of thermal updraft intensity over complex terrain using American white pelicans and a simple boundary-layer forecast model. *Boundary-Layer Meteorology*, 2002. 104(2): p. 167-199.
16. Barate, R., S. Doncieux, and J.-A. Meyer, Design of a bio-inspired controller for dynamic soaring in a simulated unmanned aerial vehicle. *Bioinspiration & Biomimetics*, 2006. 1(3): p. 76-88.

17. Cone, C.D., Thermal Soaring by Migrating Starlings. *Auk*, 1968. 85(1): p. 19-&.
18. van Loon, E.E., et al., Understanding soaring bird migration through interactions and decisions at the individual level. *Journal of Theoretical Biology*, 2011. 270(1): p. 112-126.
19. Shamoun-Baranes, J., et al., Differential Use of Thermal Convection by Soaring Birds over Central Israel. *The Condor*, 2003. 105(2): p. 208.
20. Lish, J., Associate professor in Physiological Science at Oklahoma State University, phone and email conversations. 2012.
21. Leshem, Y. and Y. YomTov, The use of thermals by soaring migrants. *Ibis*, 1996. 138(4): p. 667-674.
22. Lawrance, N.R.J. and S. Sukkarieh. A guidance and control strategy for dynamic soaring with a gliding UAV. in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*. 2009.
23. Brown, R.E. and M.R. Fedde, Airflow sensors in the avian wing. *Journal of experimental biology*, 1993. 179(1): p. 13-30.
24. Gotch, A.F., *Playground in the sky: The art and joys of gliding*. 1955: Hutchinson.
25. Laughter, J.S., Thermal navigator. 1986, Google Patents.
26. Liebelt, J. and K. Schertler, Localization system for use in e.g. glider for finding local thermal area in flight strategy utilized in military area to increase flying altitude, has evaluation unit determining position of local prevailed thermal areas. 2011, Google Patents.
27. C, L., Temperature differential sensor for gliders. 1974, Google Patents.
28. Cocatre-Zilgien, J.H., Aircraft system monitoring air humidity to locate updrafts. 2000, Google Patents.
29. Norman, C.D., Thermalling sailplane turn indicator. 1985, Google Patents.
30. Thornburg, D., *Old Buzzard's Soaring Book*. 2nd ed. 1993: Pony X Press.
31. Soaring League of North Texas. 2015.
32. Wurts, J. Finding and Recognizing Thermals. [cited 2015 August 25, 2015]; Available from: <http://houstonhawks.org>.
33. Garcia, R. and L. Barnes, Multi-UAV Simulator Utilizing X-Plane. *Journal of Intelligent & Robotic Systems*, 2010. 57(1-4): p. 393-406.
34. Allen, M.J., Guidance and control of an autonomous soaring uav. 2007.
35. Allen, M.J., *Autonomous Soaring for Improved Endurance of a Small Uninhabited Air Vehicle*. AIAA, 2005.
36. Lawrance, N.R.J. and S. Sukkarieh, Autonomous Exploration of a Wind Field with a Gliding Aircraft. *Journal of Guidance, Control, and Dynamics*, 2011. 34(3): p. 719-733.
37. Kagabo, W.B., Optimal trajectory planning for a UAV glider using atmospheric thermals. 2010, Rochester Institute of Technology.
38. Kagabo, W.B. and J.R. Kolodziej. Trajectory determination for energy efficient autonomous soaring. in *American Control Conference (ACC), 2011*. 2011.
39. Anderson, J.D., *Fundamentals of Aerodynamics*. 2007: McGraw-Hill Higher Education.
40. Harper, B., J. Kepert, and J. Ginger, Guidelines for converting between various wind averaging periods in tropical cyclone conditions. World Meteorological Organization, WMO/TD, 2010. 1555.
41. Yamartino, R., A comparison of several "single-pass" estimators of the standard deviation of wind direction. *Journal of Climate and Applied Meteorology*, 1984. 23(9): p. 1362-1366.
42. Faludi, R., *Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing*. 2010: O'Reilly Media.
43. MatLab Website. [cited 2015; Available from: <http://www.mathworks.com/products/matlab/>.

44. Yakimenko, O.A., Engineering Computations and Modeling in MATLAB/Simulink. 2011: American Institute of Aeronautics and Astronautics.
45. Meier, R. CoolTerm. 2014; Available from: <http://freeware.the-meiers.org/>.
46. Processing Website. [cited 2014; Available from: <https://processing.org/>].
47. Mesonet Website. [cited 2015; Available from: <https://www.mesonet.org/>].
48. Jarraud, M., Guide to Meteorological Instruments and Methods of Observation (WMO-No. 8). World Meteorological Organisation: Geneva, Switzerland, 2008.
49. Sokal, R.R. and F.J. Rohlf, Biometry : the principles and practice of statistics in biological research. Extensively rev. 4th ed. 2012, New York: W.H. Freeman. xix, 937 p.
50. Paynter, D., Calculation and Re-Calculation of 60-Minute Sigma Theta and Stability.
51. Mardia, K.V. and P.E. Jupp, Directional statistics. Wiley series in probability and statistics. 2000, Chichester ; New York: J. Wiley. xxi, 429 p.

## APPENDICES

## APPENDIX A: ARDUINO CODE

```
/******  
* Code name: India  
* Code by Cody Pinkerman, PhD Candidate, Oklahoma State University  
* 2014-2016  
*  
* Code to incorporate wind sensor, e-vane, temperature,  
* humidity, and pressure sensors.  
* Arduino with attached sensors will output  
* values to computer via XBee.  
* Incorporate subroutines where necessary.  
*  
* - Changed anemometer to incorporate Pull-Up resistor  
* - Changed data output to be single string to be extruded in  
* Processing GUI  
*****/  
  
//libraries  
#include <DHT22.h> //library for RHT03 aka Temperature/Humidity  
#include <SFE_BMP180.h> //library for Pressure Sensor  
#include <Wire.h>  
  
//Temperature & humidity setup  
#define DHT22_PIN 11 //Pin 11 for input for humidity/temperature  
DHT22 myDHT22(DHT22_PIN);  
  
//Pressure setup  
SFE_BMP180 pressure;  
#define ALTITUDE 300.0 //Altitude of Stillwater, OK  
  
//Speed & direction setup  
#define uint unsigned int  
#define ulong unsigned long  
#define Pin_Speed 2 //digital pin 2 for speed  
#define speed_calc 2000  
int direPin = A3; //pin A3 for analog input for direction  
  
//Declare variables  
double faren; //temperature variable  
double humid; //humidity variable  
double pressu; //pressure variable  
float spd; //speed variable  
double ang; //angle variable  
double angT;  
  
int numRevsAnemometer = 0; //Incremented in the interrupt  
ulong nextSpeed; //next calc the wind speed  
ulong time; //Millis() at each start of loop().
```

```

String stringOut; //output string
String sep = ":"; //seperator

void setup()
{
  Serial.begin(9600);
  pressure.begin();
  pinMode(Pin_Speed,INPUT);
  digitalWrite(Pin_Speed, HIGH);
  attachInterrupt(0, countAnemometer, FALLING);
  nextSpeed = millis() + speed_calc;
  time = millis();
}

void loop()
{
  char status;
  double T,P,p0,a;
  DHT22_ERROR_t errorCode;
  errorCode = myDHT22.readData();

  //Temp & RH
  switch(errorCode)
  {
    case DHT_ERROR_NONE:
      faren = myDHT22.getTemperatureC()*1.8 + 32;
      humid = myDHT22.getHumidity();
      break;
    case DHT_ERROR_CHECKSUM:
      faren = myDHT22.getTemperatureC()*1.8 + 32;
      humid = myDHT22.getHumidity();
      break;
    default:
      faren = 11.11;
      humid = 11.11;
      break;
  }
}

```

```

//Pressure
status = pressure.startTemperature();
if (status != 0)
{
  // Wait for the measurement to complete:
  delay(status);

  status = pressure.getTemperature(T);
  if (status != 0)
  {
    status = pressure.startPressure(3);
    if (status != 0)
    {
      //wait for the measurement to complete:
      delay(status);

      status = pressure.getPressure(P,T);
      if (status != 0)
      {
        pressu = P*0.0295333727; //convert pressure from mb to inHg
      }
      else pressu = 11.11; //error retrieving pressure measurement
    }
    else pressu = 11.11; //error starting pressure measurement
  }
  else pressu = 11.11; //error retrieving temperature measurement
}
else pressu = 11.11; //error starting temperature measurement

// Direction data
angT = analogRead(direPin);
//analog 0-1023 for 0-5v
//0=0.25v (N), 360=4.75v then deg = 80volt-20
ang = (angT * 0.390625) - 20; //(angT*(400/1024))-20;
spd = calcSpeed();

//Send data to computer
Serial.print(faren); //temperature, F
Serial.print(":");
Serial.print(humid); //humidity, %
Serial.print(":");
Serial.print(pressu); //pressure, inHg
Serial.print(":");
Serial.print(spd,2); //speed, mph
Serial.print(":");
Serial.print(ang,1); //angle, degrees
//Serial.println(); //Serial monitor testing

delay(2000); //wait
}

```

```
//Count revs
void countAnemometer()
{
    numRevsAnemometer++;
}

//Calculate speed
float calcSpeed()
{
    int iSpeed;
    float xx;
    long spd = 25000;//2.5 mph * 1000 *10
    spd *= numRevsAnemometer;
    spd /= speed_calc;
    iSpeed = spd;    // Need this for formatting below
    xx = iSpeed/10.0;
    numRevsAnemometer = 0; // Reset counter
    return xx;
}
```

## APPENDIX B: GRAPHICAL USER INTERFACE CODE

```
/******  
*  
* Visual display for Ground Sensor Network  
* Code by Cody Pinkerman, PhD Candidate, Oklahoma State University  
* 2014-2016  
*  
* Program creates visual display for ground sensor data. Code creates  
* window with image background of testing area and then will display data  
* that is wirelessly transmitted via XBee Series 2. Data currently includes  
* temperature, humidity, pressure, wind speed, & wind direction.  
*  
* Data is saved in .csv format along with ability to screen captures.  
* Screenshots can be incorporated into video format via the Movie Maker  
* tool in Processing (Note: tried to incorporate program to  
* create stand-alone movie, but library was removed from recent build).  
*  
* Incorporated variable speed and direction average to eliminate  
* primary wind component and determine direction of thermal component.  
* Average is taken for individual sensors over AvgN data sets.  
* Default has been set to 300 data points, or 10 minute average.  
*  
* Psuedo functions have been left in which incorporate additions  
* that will find line intersections which will  
* 1: plot point(s) on map  
* 2: create snapshot when crossings occur  
* 3: play audio cue of where the intersection is relative  
* to specified location  
*  
* v1.4 only works with Arduino Sensor code India  
* Use Processing 2.2.1 32 bit version  
*  
*****/  
  
// used for communication via xbee api  
import processing.serial.*;  
  
// used for text to speech  
import guru.ttslib.*;  
  
// xbee-api library  
import com.rapplogic.xbee.api.ApiId;  
import com.rapplogic.xbee.api.PacketListener;  
import com.rapplogic.xbee.api.XBee;  
import com.rapplogic.xbee.api.XBeeResponse;  
  
String version = "1.4";  
  
// ensure COMPORT is correct: comport XBEE is connected to on computer  
String mySerialPort = "COM4";
```

```

// create and initialize a new xbee object
XBee xbee = new XBee();
int error=0;

// make an array list of sensor objects for display
ArrayList sensordisplay = new ArrayList();
// create a font for display
PFont font;
PImage img;

// table for data storage
Table table;

// location variables of sensors
int xA = 490; //490 //470
int yA = 260; //260 //150
int xB = 575;
int yB = 420;
int xC = 490;
int yC = 680;

// text box size
int xText = 60;
int yText = 80;

// location of the pilot
int xp = 480;
int yp = 393;
// time for average to calculate
int AvgN = 300; //Back
int AvgF = 2 ; //Front average for point

// arrays & ints for wind averaging (Window Averaging)
float [][] Aarray = new float[AvgN][2];
float [][] Barray = new float[AvgN][2];
float [][] Carray = new float[AvgN][2];
float [][] AarrayF = new float[AvgF][2];
float [][] BarrayF = new float[AvgF][2];
float [][] CarrayF = new float[AvgF][2];

int Acount = 0, Bcount = 0, Ccount = 0;
int AF = 0, BF = 0, CF = 0;
float AvgSpd, AvgDir, AvgDirx, AvgDiry, AvgX, AvgY;
float FAvgx, FAvgy;

// use a constant wind for a given day
float uniformW = 3; //mph
float uniformD = 70; //degrees 105 (Azimuth)
boolean useWind = false; //change to "true" to specify constant wind conditions

```

```

boolean iAB, iBC, iAC = false;
float tempX, tempY = 0;
float xx = 0, yy = 0;
float [][] AvgLine = new float[3][2];

TTS tts;

void setup()
{
  size(1000, 800); // screen size
  smooth(); // anti-aliasing for graphic display
  img = loadImage("air_field_1000x800.jpg"); //image of airfield

  tts = new TTS();

  font = loadFont("Arial-BoldMT-12.vlw");
  textFont(font); // use font for text

  PropertyConfigurator.configure(dataPath("log4j.properties"));

  try
  {
    // opens serial port defined above, at specified baud rate
    xbee.open(mySerialPort, 9600);
  }
  catch (XBeeException e)
  {
    println("** Error opening XBee port: " + e + " **");
    println("Is your XBee plugged in to your computer?");
    println("Did you set your COM port in the code near line 20?");
    error=1;
  }

  // save the data in table/file
  table = new Table();
  table.addColumn("Time");
  table.addColumn("Name");
  table.addColumn("Temperature");
  table.addColumn("Humidity");
  table.addColumn("Pressure");
  table.addColumn("Speed");
  table.addColumn("Angle");
}

void draw()
{
  background(img); // draw a background
  WindAverage(); // find average wind speed
  DrawTime(); // draw time
  ellipse(xp,yp, 10,10); // pilot location

```

```

// report any serial port problems in the main window
if (error == 1)
{
    fill(0);
    text("*** Error opening XBee port: **\n"+
        "Is your XBee plugged in to your computer?\n" +
        "Did you set your COM port in the code near line 20?", width/3, height/2);
}

SensorData data = new SensorData(); // create a sensor data object
data = getData(); // put data into the sensor data object

if (data.temp >=0 && data.address != null)
{
    // check to see if a display object already exists for this sensor
    int i;
    boolean foundIt = false;
    for (i=0; i < sensordisplay.size(); i++)
    {
        if ( ((SensorDisplay) sensordisplay.get(i)).address.equals(data.address) )
        {
            foundIt = true;
            break;
        }
    }
}

float temperatureFar = (data.temp);
float relativeHum = (data.humid);
float pressure = (data.press);
float direction = (data.dir);
float windSpeed = (data.spd);
String sensorName = (data.address);

// save the data to csv file
saveAs(sensorName, temperatureFar, relativeHum, pressure, windSpeed, direction);

// update the sensor box if it exists, otherwise create a new one
if (foundIt)
{
    ((SensorDisplay) sensordisplay.get(i)).temp = temperatureFar;
    ((SensorDisplay) sensordisplay.get(i)).humid = relativeHum;
    ((SensorDisplay) sensordisplay.get(i)).press = pressure;
    ((SensorDisplay) sensordisplay.get(i)).dir = direction;
    ((SensorDisplay) sensordisplay.get(i)).spd = windSpeed;
}

```

```

if(sensorName == "Alpha")
{

    Aarray[Acount][0] = AarrayF[AF][0];
    Aarray[Acount][1] = AarrayF[AF][1];
    Acount = Acount + 1;

    AarrayF[AF][0] = windSpeed;
    AarrayF[AF][1] = direction;
    AF = AF + 1;

    if(AF >= AvgF)
    {
        AF = 0;
    }
    if(Acount >= AvgN)
    {
        Acount = 0;
    }

}
if(sensorName == "Bravo")
{

    Barray[Bcount][0] = BarrayF[BF][0];
    Barray[Bcount][1] = BarrayF[BF][1];
    Bcount = Bcount + 1;

    BarrayF[BF][0] = windSpeed;
    BarrayF[BF][1] = direction;
    BF = BF + 1;
    if(Bcount >= AvgN)
    {
        Bcount = 0;
    }
    if(BF >= AvgF)
    {
        BF = 0;
    }

}

```

```

if(sensorName == "Charlie")
{
  Carray[Ccount][0] = CarrayF[CF][0];
  Carray[Ccount][1] = CarrayF[CF][1];
  Ccount = Ccount + 1;
  if(Ccount >= AvgN)
  {
    Ccount = 0;
  }
  CarrayF[CF][0] = windSpeed;
  CarrayF[CF][1] = direction;
  CF = CF + 1;
  if(CF >= AvgF)
  {
    CF = 0;
  }
}
}
else if (sensordisplay.size() < 10)
{
  if (data.address == "Alpha")
  {
    sensordisplay.add(new SensorDisplay(data.address, xText, yText, xA, yA));
    ((SensorDisplay) sensordisplay.get(i)).temp = temperatureFar;
    ((SensorDisplay) sensordisplay.get(i)).humid = relativeHum;
    ((SensorDisplay) sensordisplay.get(i)).press = pressure;
    ((SensorDisplay) sensordisplay.get(i)).dir = direction;
    ((SensorDisplay) sensordisplay.get(i)).spd = windSpeed;

    // fill array with initial values
    for(int j=0;j<AvgN;j++)
    {
      Aarray[j][0] = windSpeed;
      Aarray[j][1] = direction;
    }
    for(int j=0;j<AvgF;j++)
    {
      AarrayF[j][0] = windSpeed;
      AarrayF[j][1] = direction;
    }
  }
}
if (data.address == "Bravo")
{
  sensordisplay.add(new SensorDisplay(data.address, xText, yText, xB, yB));
  ((SensorDisplay) sensordisplay.get(i)).temp = temperatureFar;
  ((SensorDisplay) sensordisplay.get(i)).humid = relativeHum;
  ((SensorDisplay) sensordisplay.get(i)).press = pressure;
  ((SensorDisplay) sensordisplay.get(i)).dir = direction;
  ((SensorDisplay) sensordisplay.get(i)).spd = windSpeed;
}

```

```

// fill array with initial values
for(int j=0;j<AvgN;j++)
{
  Barray[j][0] = windSpeed;
  Barray[j][1] = direction;
}
for(int j=0;j<AvgF;j++)
{
  BarrayF[j][0] = windSpeed;
  BarrayF[j][1] = direction;
}
}
if (data.address == "Charlie")
{
  sensordisplay.add(new SensorDisplay(data.address, xText, yText, xC, yC));
  ((SensorDisplay) sensordisplay.get(i)).temp = temperatureFar;
  ((SensorDisplay) sensordisplay.get(i)).humid = relativeHum;
  ((SensorDisplay) sensordisplay.get(i)).press = pressure;
  ((SensorDisplay) sensordisplay.get(i)).dir = direction;
  ((SensorDisplay) sensordisplay.get(i)).spd = windSpeed;

// fill array with initial values
for(int j=0;j<AvgN;j++)
{
  Carray[j][0] = windSpeed;
  Carray[j][1] = direction;
}
for(int j=0;j<AvgF;j++)
{
  CarrayF[j][0] = windSpeed;
  CarrayF[j][1] = direction;
}
}
}

// render the sensor info on the screen
for (int j =0; j<sensordisplay.size(); j++)
{
  ((SensorDisplay) sensordisplay.get(j)).render();
}

```

```

/*****
// Pseudo code to determine if lines intercept
// If true, code will save screenshot
// Check for thermal line intersect
iAB =
lineLineIntersect(xA,yA,AvgLine[0][0],AvgLine[0][1],xB,yB,AvgLine[1][0],AvgLine[1][1]);
iBC =
lineLineIntersect(xB,yB,AvgLine[1][0],AvgLine[1][1],xC,yC,AvgLine[2][0],AvgLine[2][1]);
iAC =
lineLineIntersect(xC,yC,AvgLine[2][0],AvgLine[2][1],xA,yA,AvgLine[0][0],AvgLine[0][1]);

if ((iAB == true) || (iBC == true) || (iAC == true))
{
//saveFrame("data/2Intersects/Sensor-#####.png");
}

// play noise if there are three intersection
if (iAB == true && iBC == true && iAC == true)
{
tempX = (AvgLine[0][0]+AvgLine[1][0]+AvgLine[2][0])/3 - xp;
tempY = (AvgLine[0][1]+AvgLine[1][1]+AvgLine[2][1])/3 - yp;
saveFrame("data/3Intersects/Sensor-#####.png");
float TDI = (degrees(atan2(tempY, tempX)));

// determine direction of triangulation from pilot's position
// and play audio cue for pilot
if (TDI >= -22.5 && TDI < 22.5) //EAST
tts.speak("East");
if (TDI >= 22.5 && TDI < 67.5) //SOUTHEAST
tts.speak("Southeast");
if (TDI >= 67.5 && TDI < 112.5) //SOUTH
tts.speak("South");
if (TDI >= 112.5 && TDI < 157.5) //SOUTHWEST
tts.speak("Southwest");
if (TDI >= 157.5 || TDI < -157.5) //WEST
tts.speak("West");
if (TDI >= -157.5 && TDI < -112.5) //NORTHWEST
tts.speak("Northwest");
if (TDI >= -112.5 && TDI < -67.5) //NORTH
tts.speak("North");
if (TDI >= -67.5 && TDI < -22.5) //NORTHEAST
tts.speak("Northeast");
}

// reset values
iAB = iBC = iAC = false;

// save the screen shot
saveFrame("data/test/Sensor-#####.png");
*****/
} // end of draw loop

```

```

// defines the sensor data object
class SensorData
{
  float temp; // Temperature
  float humid; // Relative humidity
  float press; // Pressure
  float dir; // Wind direction
  float spd; // Wind speed
  String address;
}

// define sensor display class
class SensorDisplay
{
  int sizeX, sizeY, posX, posY; // size and position of display
  int ALx, ALy, tempLine;
  String address; // name of router
  float temp, humid, press, dir, spd, rad, ThermalDir; // store variables locally

  SensorDisplay(String _address, int _sizeX, int _sizeY, int _posX, int _posY) // initialize
  thermometer object
  {
    address = _address;
    sizeX = _sizeX;
    sizeY = _sizeY;
    posX = _posX;
    posY = _posY;
  }

  void render()
  {
    noStroke(); // remove shape edges

    // circle
    fill(0);
    ellipseMode(CENTER);
    ellipse(posX, posY, 10, 10);

    // black box
    fill(0);
    rect(posX+5, posY+5, sizeX, sizeY);

    // Wind vector
    pushMatrix(); // push this section out of the current loop
    translate(posX, posY);
    rotate(radians(dir+90));
    strokeWeight(3);
    stroke(15,237,255);
    line(0, 0, spd*10, 0); //50 or width
    popMatrix();
  }
}

```

```

//Loop to find average speed and direction for sensor
if (address == "Alpha")
{
  AvgDirx = 0;
  AvgDiry = 0;
  FAvgx = 0;
  FAvgy = 0;
  tempLine = 0;
  //Loop for back average
  for(int j=0;j<AvgN;j++)
  {
    AvgDirx = AvgDirx + Aarray[j][0]*cos(radians(Aarray[j][1]));
    AvgDiry = AvgDiry + Aarray[j][0]*sin(radians(Aarray[j][1]));
  }
  //Loop for front average
  for(int k=0;k<AvgF;k++)
  {
    FAvgx = FAvgx + AarrayF[k][0]*cos(radians(AarrayF[k][1]));
    FAvgy = FAvgy + AarrayF[k][0]*sin(radians(AarrayF[k][1]));
  }
}
if (address == "Bravo")
{
  AvgDirx = 0;
  AvgDiry = 0;
  FAvgx=0;
  FAvgy=0;
  tempLine = 1;
  for(int j=0;j<AvgN;j++)
  {
    AvgDirx = AvgDirx + Barray[j][0]*cos(radians(Barray[j][1]));
    AvgDiry = AvgDiry + Barray[j][0]*sin(radians(Barray[j][1]));
  }
  //Loop for front average
  for(int k=0;k<AvgF;k++)
  {
    FAvgx = FAvgx + BarrayF[k][0]*cos(radians(BarrayF[k][1]));
    FAvgy = FAvgy + BarrayF[k][0]*sin(radians(BarrayF[k][1]));
  }
}
}

```

```

if (address == "Charlie")
{
  AvgDirx = 0;
  AvgDiry = 0;
  FAvgx=0;
  FAvgy=0;
  tempLine = 2;
  for(int j=0;j<AvgN;j++)
  {
    AvgDirx = AvgDirx + Carray[j][0]*cos(radians(Carray[j][1]));
    AvgDiry = AvgDiry + Carray[j][0]*sin(radians(Carray[j][1]));
  }
  //Loop for front average
  for(int k=0;k<AvgF;k++)
  {
    FAvgx = FAvgx + CarrayF[k][0]*cos(radians(CarrayF[k][1]));
    FAvgy = FAvgy + CarrayF[k][0]*sin(radians(CarrayF[k][1]));
  }
}

if(useWind == true)
{
  AvgDirx = uniformW*cos(radians(uniformD));
  AvgDiry = uniformW*sin(radians(uniformD));
  AvgN = 1;
}

// Thermal vector
pushMatrix(); // push this section out of the current loop
ThermalDir = degrees(atan2((FAvgy/AvgF)-(AvgDiry/AvgN),(FAvgx/AvgF)-
(AvgDirx/AvgN)));
translate(posX, posY);
rotate(radians(ThermalDir+90)); //add 90 for display
strokeWeight(1);
stroke(255, 0, 0);
line(0,0,width,0);
popMatrix();

AvgLine[tempLine][0] = Math.round(width*cos(radians(ThermalDir+90)))+posX;
AvgLine[tempLine][1] = Math.round(width*sin(radians(ThermalDir+90)))+posY;

//Reset values
AvgSpd = 0.0;
AvgDir = 0.0;
AvgDirx = 0.0;
AvgDiry = 0.0;

```

```

// text in box
textAlign(LEFT);
textSize(12);
fill(0,252,0); // green
text(address, posX+10, posY+20);
text((temp)+" F", posX+10, posY+35);
text((humid)+" %", posX+10, posY+50);
text((press) + " in Hg", posX+10, posY+65);
text((spd) + " mph", posX+10, posY+80);
}
}

void DrawTime()
{
  String AD;
  fill(0);
  stroke(0);
  rect(5, 5, 85, 55);
  fill(0,252,0); // green
  textSize(16);
  text(hour()+":"+minute()+":"+second(),18,20);
  AD = String.format("%.2f", AvgSpd);
  text((AD) + " mph", 7, 35);
  AD = String.format("%.2f", AvgDir);
  text((AD) + " Deg", 7, 50);
}

// queries the XBee for incoming data frames and puts them into data object
SensorData getData()
{
  SensorData data = new SensorData();
  int value = -1; // returns an impossible value if there's an error
  String address = ""; // returns a null value if there's an error

  try
  {
    // wait until a packet is received.
    XBeeResponse response = xbee.getResponse(3000);
    // print response to see raw packet
    println("Received response " + response.toString());

    // check that this frame is a valid RX
    if (response.getApiId() == ApiId.ZNET_RX_RESPONSE && !response.isError())
    {
      ZNetRxResponse dataSample = (ZNetRxResponse)(XBeeResponse) response;

      // get the sender's 64-bit address
      int[] addressArray = dataSample.getRemoteAddress64().getAddress();

```

```

// put the address int array into a formatted string
String[] hexAddress = new String[addressArray.length];
for (int i=0; i<addressArray.length;i++)
{
    // format each address byte with leading zeros:
    hexAddress[i] = String.format("%02x", addressArray[i]);
}
// join the array together with colons for readability:
String senderAddress = join(hexAddress, ":");

// rename address
if (senderAddress.equals("00:13:a2:00:40:b5:56:05"))
{
    data.address = "Alpha";
} else if (senderAddress.equals("00:13:a2:00:40:b6:29:e9"))
{
    data.address = "Bravo";
} else if (senderAddress.equals("00:13:a2:00:40:b5:56:09"))
{
    data.address = "Charlie";
} else {
    data.address = senderAddress;
}

// grab data from packet
int[] payload = dataSample.getData();
//Convert payload from hex to ascii
StringBuilder outputT = new StringBuilder();
for (int i=0; i<(payload.length-1); i++)
{
    String strComb = String.format("%x", payload[i]);
    outputT.append((char)Integer.parseInt(strComb, 16));
}
String theDataPacket = outputT.toString();
//Split sections into single array
String[] DataParts = splitTokens(theDataPacket, ":");

// Temperature // convert ascii/string to float
try
{
    data.temp = Float.parseFloat(DataParts[0]); // Temperature
}
catch (NumberFormatException e) // catch used incase error is read into packet
{
    data.temp = 40;
}
// Humidity // convert ascii/string to float
try
{
    data.humid = Float.parseFloat(DataParts[1]); // %RH
}

```

```

catch (NumberFormatException e)
{
    data.humid = 0;
}
// Pressure // convert ascii/string to float
try
{
    data.press = Float.parseFloat(DataParts[2]); // Pressure
}
catch (NumberFormatException e)
{
    data.press = 0;
}

// Wind speed // convert ascii/string to float
try
{
    data.spd = Float.parseFloat(DataParts[3]); // Speed
}
catch (NumberFormatException e)
{
    data.spd = 0;
}

// Direction // convert ascii/string to float
try
{
    data.dir = Float.parseFloat(DataParts[4]); // Direction
}
catch (NumberFormatException e)
{
    data.dir = 0;
}
}
else if (!response.isError()){
    println("Got error in data frame");
}
else {
    println("Got non-i/o data frame");
}
}

catch (XBeeException e)
{
    println("Error receiving response: " + e);
}

return data; // sends the data back to the calling function
}

```

```

// save data to table
void saveAs(String name, float temp, float humidity, float pressure, float speed, float direction)
{
    TableRow newRow = table.addRow();
    newRow.setString("Time", hour()+":"+minute()+":"+second());
    newRow.setString("Name", name);
    newRow.setFloat("Temperature", temp);
    newRow.setFloat("Humidity", humidity);
    newRow.setFloat("Pressure", pressure);
    newRow.setFloat("Speed", speed);
    newRow.setFloat("Angle", (direction));
    saveTable(table, "data/Sensor_Data.csv");
}

// used to average the wind speeds of the sensors
void WindAverage()
{
    AvgSpd=0;
    AvgDir=0;
    AvgX = 0;
    AvgY = 0;
    for(int i=0; i<AvgN; i++)
    {
        AvgX =
Aarray[i][0]*cos(radians(Aarray[i][1]))+Barray[i][0]*cos(radians(Barray[i][1]))+Carray[i][0]*co
s(radians(Carray[i][1])) + AvgX;
        AvgY =
Aarray[i][0]*sin(radians(Aarray[i][1]))+Barray[i][0]*sin(radians(Barray[i][1]))+Carray[i][0]*sin(
radians(Carray[i][1])) + AvgY;
    }
    AvgSpd = sqrt(pow((AvgX/(3*AvgN)),2) + pow((AvgY/(3*AvgN)),2));
    AvgDir = degrees(atan2(AvgY/(3*AvgN),AvgX/(3*AvgN)));
}

```

```

boolean lineLineIntersect(float x1, float y1, float x2, float y2, float x3, float y3, float x4, float y4 )
{
    boolean over = false;
    float a1 = y2 - y1;
    float b1 = x1 - x2;
    float c1 = a1*x1 + b1*y1;

    float a2 = y4 - y3;
    float b2 = x3 - x4;
    float c2 = a2*x3 + b2*y3;
    float det = a1*b2 - a2*b1;
    if(det == 0)
    {
        // Lines are parallel
    }
    else
    {
        xx = (b2*c1 - b1*c2)/det;
        yy = (a1*c2 - a2*c1)/det;
        if(xx > min(x1, x2) && xx < max(x1, x2) &&
           xx > min(x3, x4) && xx < max(x3, x4) &&
           yy > min(y1, y2) && yy < max(y1, y2) &&
           yy > min(y3, y4) && yy < max(y3, y4))
        {
            over = true;
        }
    }
    return over;
}

```

## VITA

Cody Wayne Pinkerman

Candidate for the Degree of

Doctor of Philosophy

Thesis: UNDERSTANDING AVIAN SOARING TO EXTEND UAV MISSION  
ENDURANCE THROUGH REMOTE DETECTION OF THERMAL  
UPDRAFTS

Major Field: Mechanical and Aerospace Engineering

Biographical:

Education:

Completed the requirements for the Doctor of Philosophy degree in Mechanical and Aerospace Engineering with emphasis in Unmanned Aircraft Systems at Oklahoma State University, Stillwater, Oklahoma in May, 2016.

Completed the requirements for the Master of Science degree in Mechanical and Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in July, 2010.

Completed the requirements for the Bachelor of Science degree in Aerospace and Mechanical Engineering at Oklahoma State University, Stillwater, Oklahoma in May, 2008.

Experience:

Instructor of Record for MAE 3293 Compressible Fluid Flow for four semesters at Oklahoma State University.

Teaching Assistant for multiple Graduate and Undergraduate courses for eight years at Oklahoma State University.

Professional Memberships:

Academy of Model of Aeronautics, American Institute of Aeronautics and Astronautics