Indiana University-Purdue University Fort Wayne

Department of Civil & Mechanical Engineering

ME 487/488

Four-Cup Anemometer Senior Design Project

Report #2

Project Title: Design, Construction, Testing, Calibration, and Demonstration of a Four-Cup Anemometer

Team Members: Christopher Wright

Grant Cornay

Mark Underwood

Faculty Advisor: Dr. Josué Njock Libii

Date: 12/7/2015

Table of Contents

Contents

Table of Figures

Table of Tables

Table of Equations

Acknowledgements

Thanks are given to Dr. Josué Njock Libii for advising of the team for designing, building, and testing the anemometer. Thanks are given to faculty member John Mitchell for assistance in the construction of the anemometer. Additional thanks go out to the Department of Civil & Mechanical Engineering of IPFW for the funding, resources, and tools contributed to designing and building the anemometer. Neff Engineering in Fort Wayne and Onxx Tool in Avilla deserve thanks for providing services at a discount to the team. Lastly, thanks goes to Purdue's department of engineering and Dr. John Sullivan for use of their Boeing wind tunnel for calibrating the anemometer.

Abstract

The purpose of this document is to report on the design revision, construction, and testing of an anemometer to address the problem statement. The Department of Civil & Mechanical Engineering of IPFW required the team to design, construct, test, calibrate, and demonstrate a four-cup anemometer for the purpose of measuring wind speed. This report is concerned with the construction and testing of the four-cup anemometer in the conclusion of a two semester long process, where the anemometer was conceptualized, designed, planned, and budgeted in the first semester. The report is divided into the sections of design revisions of the anemometer, construction of the apparatus, testing and calibration of its functions, and conclusions and recommendations from the project.

Problem Statement

Section P.1: Problem Statement Overview

The IPFW College of Engineering, Technology, & Computer Science (ETCS) is a division of Purdue's main engineering campus which is located in West Lafayette. Their mission is to provide comprehensive undergraduate and graduate programs that prepare students for a variety of careers in engineering, engineering technology, computer science and related disciplines, and leadership, and to serve the needs of the region and strive to achieve national recognition through experience in teaching and learning, innovation and research, community engagement, economic development, and service.

The IPFW Engineering Department sought out a team of students to create an anemometer to use for the fluids classes and engineering demonstrations. It will be used to illustrate how the speed of air (wind) can be measured through the use of air drag on an object's surface. The primary interest of this project is to create an anemometer that is easy to use, portable, durable, reliable, and can be used in a variety of demonstrations.

Section P.2: Requirements and Specifications

The anemometer must be reliable by yielding reproducible results during demonstrations. This means the anemometer must be accurate when operating at the desired wind speeds for demonstrations. Therefore, the goals of the anemometer are the following:

1. The anemometer must be able to measure wind speeds accurately within ±5% of the steady state wind speed.
2. The anemometer must be able to measure with this accuracy for wind speeds ranging from 5 $\frac{km}{hr}$ to 20 $\frac{km}{hr}$.

Section P.3: Given Parameters

The given, or fixed, parameters are those that are guidelines for the design of the anemometer.

1. The anemometer must be a classic four cup anemometer, that is, an anemometer where four wind catchers (cups) are located around a central axis that "catch" the wind causing the anemometer to rotate.
2. The anemometer must be able to display the wind speed.
3. The anemometer must be calibrated against a known source of wind speed.

Section P.4: Design Variables

The IPFW Department of Civil & Mechanical Engineering placed only a few quantitative requirements for the design. Many of the variables of the anemometer were at the discretion of the team's design. The variables to consider when creating the anemometer relate to the operating conditions of the instrument and the components used to create the instrument. The major design variables of the anemometer are:

1. The method of calculating the wind speed from rotational speed.
2. The method of calibrating the anemometer.
3. How the instrument will display and record the wind speed.
4. The variables of each component, such as dimensions and material.

Section P.5: Limitations and Constraints

The limitations and constraints of the anemometer are restrictions to ensure the instrument fits the needs of the IPFW Department of Civil & Mechanical Engineering. The crucial limitations and constraints are the following:

1. The weight of the anemometer when transporting it must be less than 7lbs.
2. The size of the anemometer must be able to fit in an 18" by 18" by 18" cube.
3. The cost of the anemometer must be under the $300 budget contributed by the IPFW Department of Civil & Mechanical Engineering.

Section P.6: Additional Considerations

Additional considerations for the anemometer encompass goals that are not explicit nor required. These goals are considered not mandatory to define success of the project.

1. The anemometer is to be designed such that it will not fail and cause injury to nearby viewers of any demonstration.
2. The anemometer is to be easy to assemble for operation and disassemble for storage.
3. The anemometer is to be durable enough to last many years of use.
4. The anemometer is to operate and yield results that are simple enough for students without fluids knowledge to understand how it operates.
5. The anemometer is to be able to operate and record data in an outdoors environment unattended for at least a day.
6. The anemometer is to be portable for demonstrations at various locations.

Section 1: Initial Design & Revisions

Section 1.1: Initial Design Overview

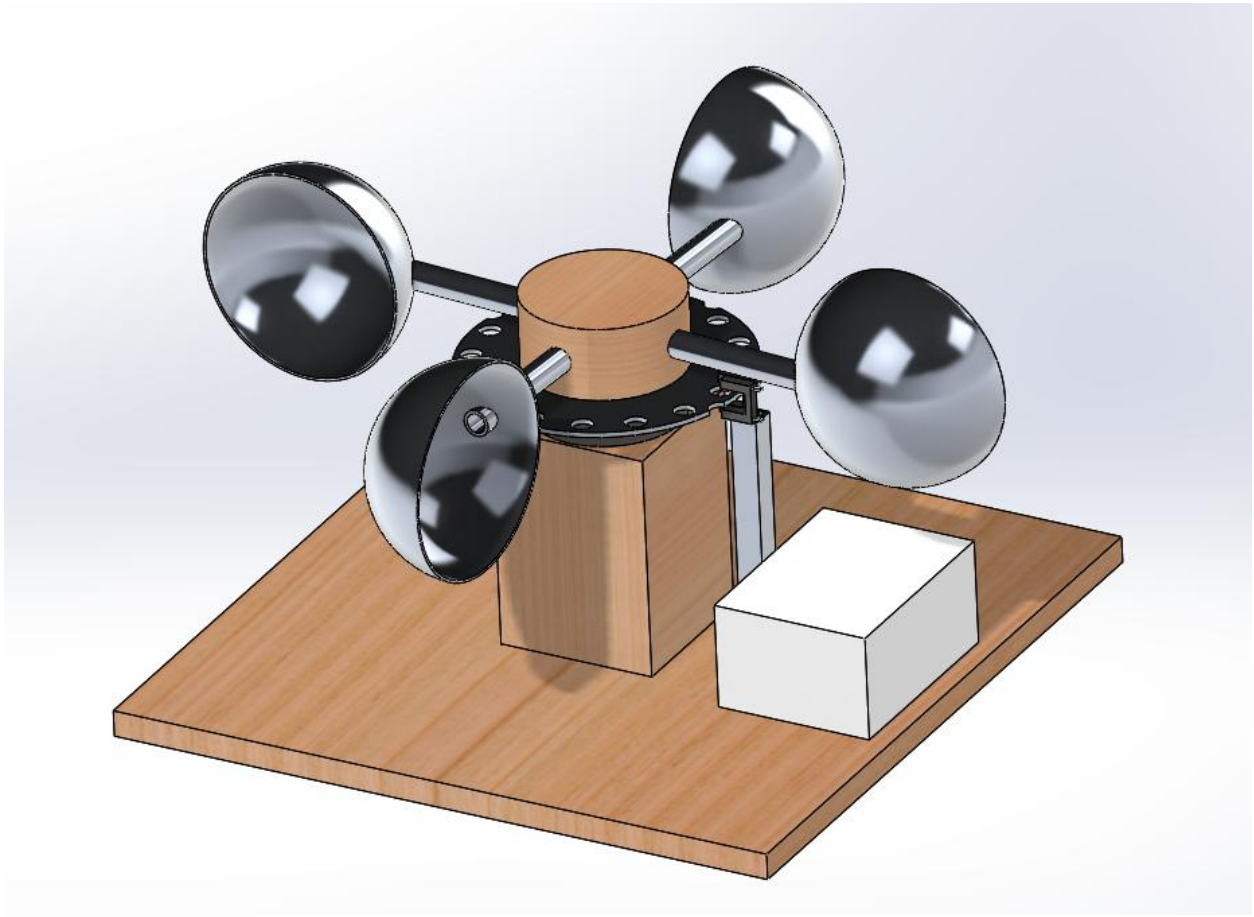


Figure 1.1-x: *Initial Anemometer Design*

Last semester a final detailed design was decided upon, which is seen in Figure 1.1-x. The design followed the simple design for a four-cup anemometer, but with the additions of a measurement disk and an integrated case. The measurement disk had holes drilled so that a photo interrupter sensor could detect the holes using light. The sensor was connected to an Arduino Uno, where the reported breaks of light passing through the measurement disk would yield a rotational speed. The Arduino Uno would then process or export the data to yield wind speed from an equation. With the exception of the disk, all non-structure components were available on the market requiring minimal machining. As seen in Figure 1.1-y, all structural components, the d) hub, g) support shaft, and a) base, were made of wood for easy machining. The integrated case included a wooden lid that would rest on top of the base and use latches to secure the lid to the device. The anemometer when covered by the integrated case's lid is shown in Figure 1.1-z. The proposed design would begin spinning at 2 $\frac{km}{hr}$ and would fail around wind speeds of 96.3 $\frac{km}{hr}$ due to the device toppling over.

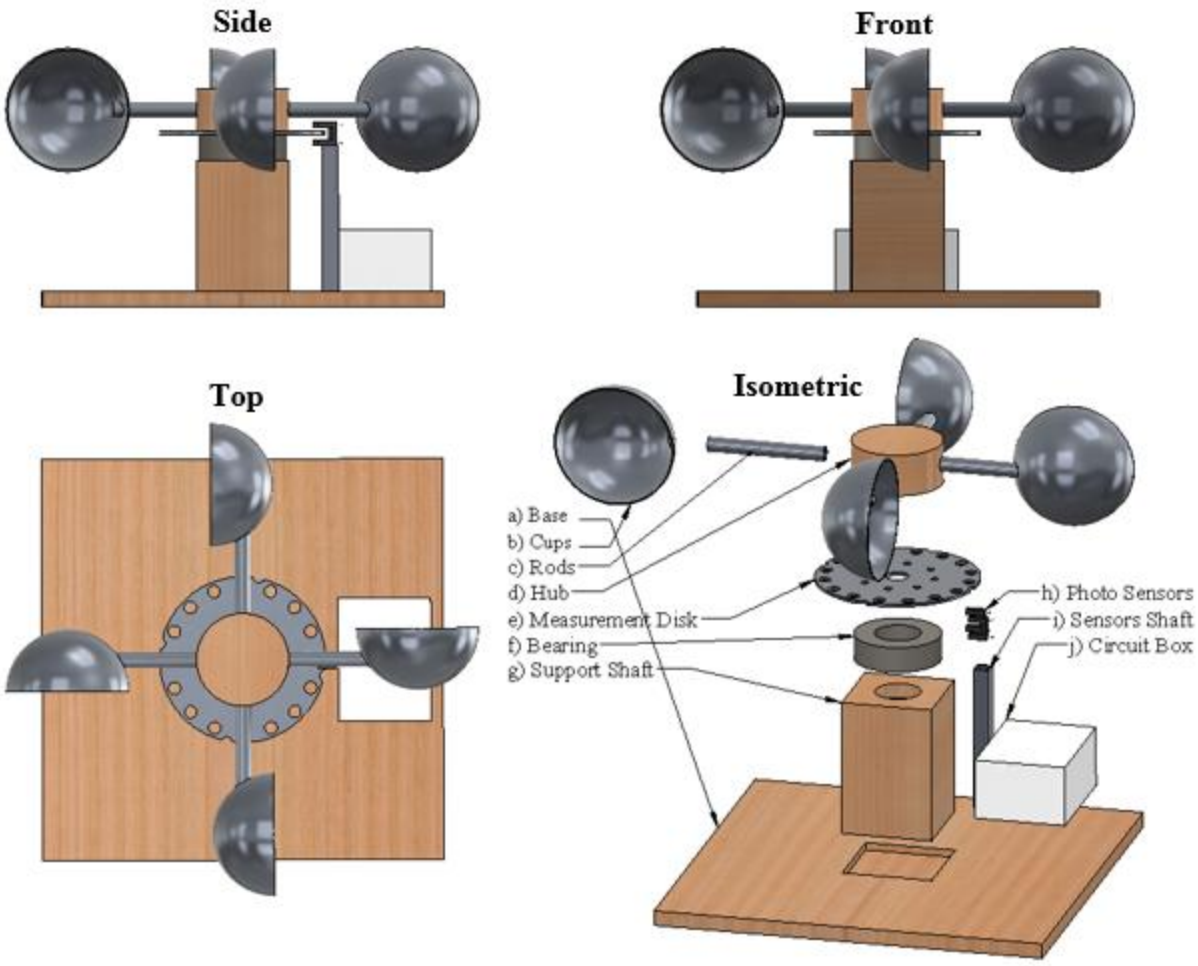


Figure 1.1-y: *Initial Anemometer Design's Components*

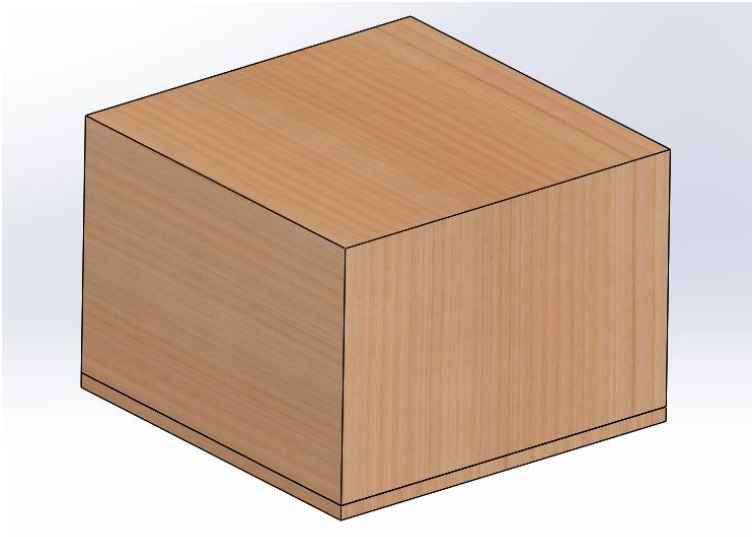


Figure 1.1-z: *Initial Anemometer Design's Integrated Case (Missing Latches)*

Section 1.2: Initial Design Mechanical Details

The parts listed in this section are pointed out on the device in Figure 1.1-y from the previous page.

Section 1.2.a: Base

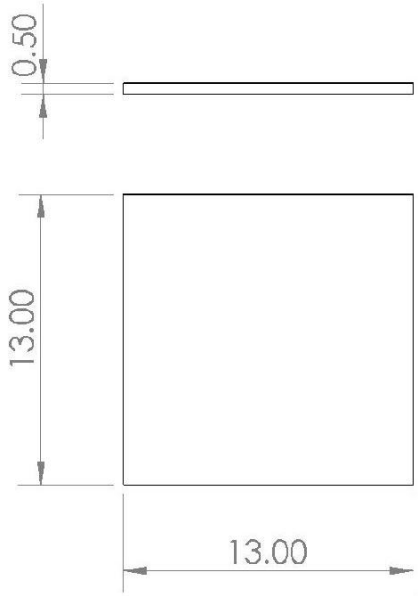
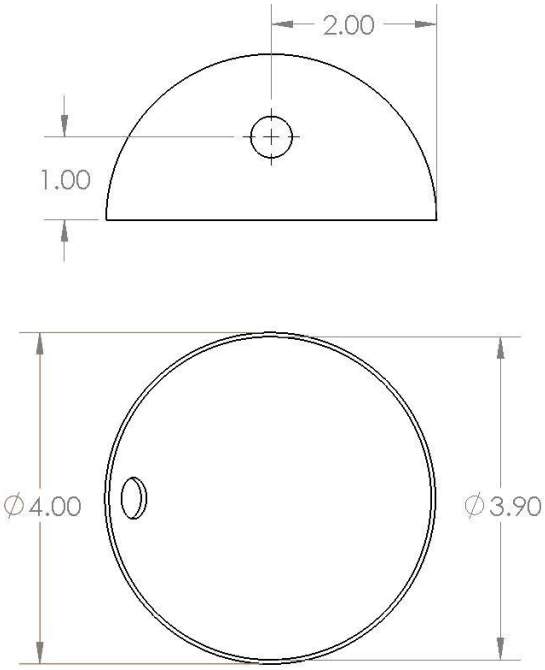


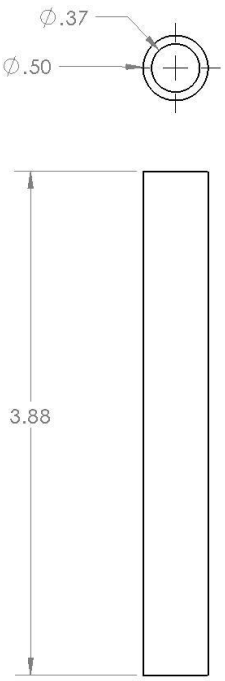
Figure 1.2.a: *Base Diagram*

The base needed to keep the system balanced from tipping over due to the moment produced by the wind. The maximum size of the base was a design limitation of 18" by 18". The variables of thickness and width of the base were determined by what is necessary to prevent tipping and minimize weight and size. Additionally, the base had a minimum width that was great enough to contain the length of the rod plus the cup diameter. The base would weigh 7.583 lb.

Section 1.2.b: Wind Catchers (Cups)

The wind catchers of the anemometers, henceforth referred to as the cups, were chosen to be hollow hemispheres due to the wide availability of information on the coefficient of drag of spheres, specifically hollow hemispheres. The chosen material of the cup was aluminum for the strength and being lightweight. The only variable pertaining to the cup was the diameter of the cup. The weight of each cup before manufacturing was to be .1 lb. The chosen dimensions for the chosen cup are found in Figure 1.2.b on the next page.





(Left) ***Figure 1.2.b:*** *Cups Diagram*
(Right) ***Figure 1.2.c:*** *Rods Diagram*

Section 1.2.c: Crossmembers (Rods)

The crossmembers of the anemometer, which hold the cups onto the central hub, henceforth referred to as rods, were chosen to be hollow aluminum tubes to minimize their drag and weight and maximize their strength. Due to the low forces expected on the system, the smallest diameter was sought after, leaving the only variable the length of the rod. Each rod weighed .02 lb.

Section 1.2.d: Central Hub

The central hub kept the rods perpendicular to each other and connected them to the bearing. The central hub will have large holes drilled into it and will be of significant volume, so cedar wood was chosen as a material. The central hub's diameter was chosen to be the diameter of the bearing chosen, and the height equal to three times the rod radius, leaving no variables to be determined for the central hub. The weight of the central hub was .172 lb. The central hub diagram is in Figure 1.2.d on the next page.

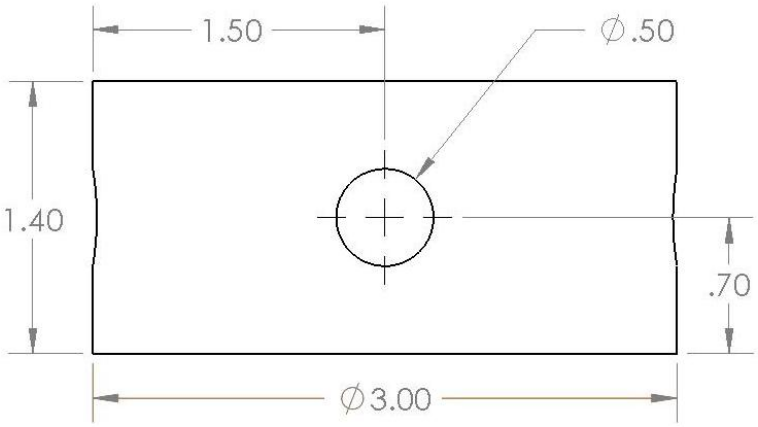


Figure 1.2.d: *Central Hub Diagram*

Section 1.2.e: Measurement Disk

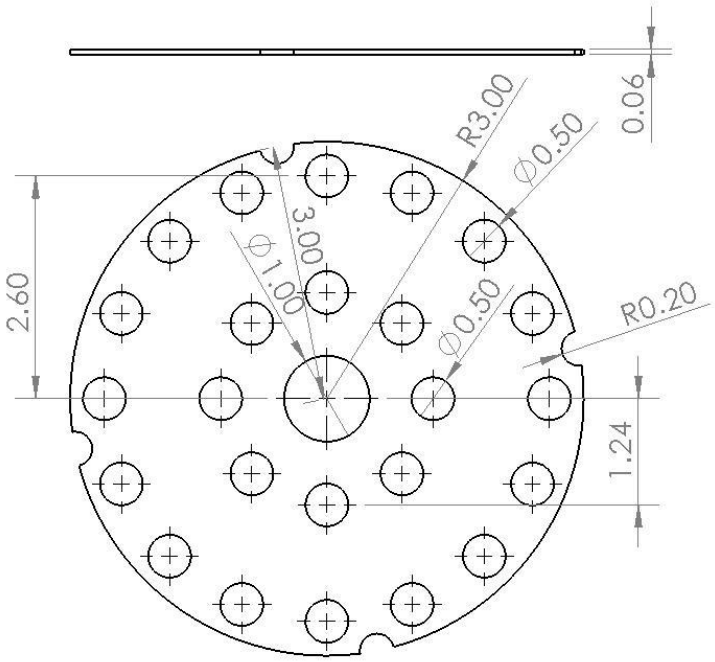


Figure 1.2.e: *Measurement Disk Diagram*

The measurement disk was the means by which the electronics detected rotation. It was an aluminum disk with holes drilled around it at uniform distances apart. The eight center holes were to allow the adhesive to have a stronger hold, and to reduce weight of the disk. The smallest thickness disk was chosen to reduce weight, and the disk only needed to have a diameter greater than the bearing's, so a diameter of 6" was chosen. The material and dimensions of the disk were limited to what was available on the market, since uniform measurements are important for the sensors. The disk weighs 0.021 lbs before manufacturing.

Section 1.2.f: Bearing

The bearing chosen was a banded thrust bearing, and all of its design variables were fixed by the manufacturer. The weight of the bearing was .49 lbs. An excerpt of the data sheet is found in Appendix A.2.

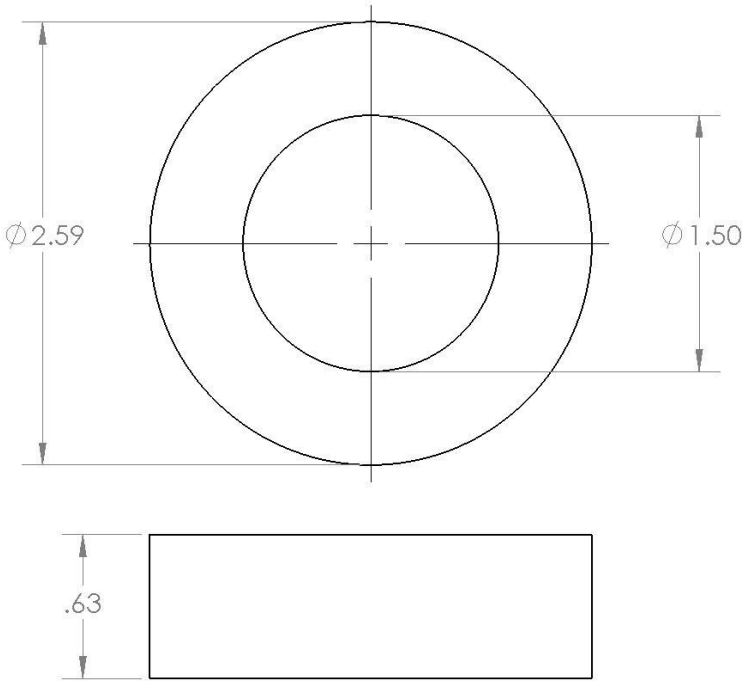


Figure 1.2.f: *Bearing Diagram*

Section 1.2.g: Support Shaft

The support shaft must keep the system elevated so that the system cannot be affected by the boundary layer created by air drag on the base. The shaft material was chosen to be cedar due to the component's size, with a width greater than the bearing's radius and a height that would yield at least an inch distance between the cups and any other boundary layer generated by the other components. This left the only variable to be the thickness of the shaft for the purpose of maintaining enough strength, but reducing the weight. The weight of this component is .207 lbs. Figure 1.2.g shows the support shaft dimensions on the next page.

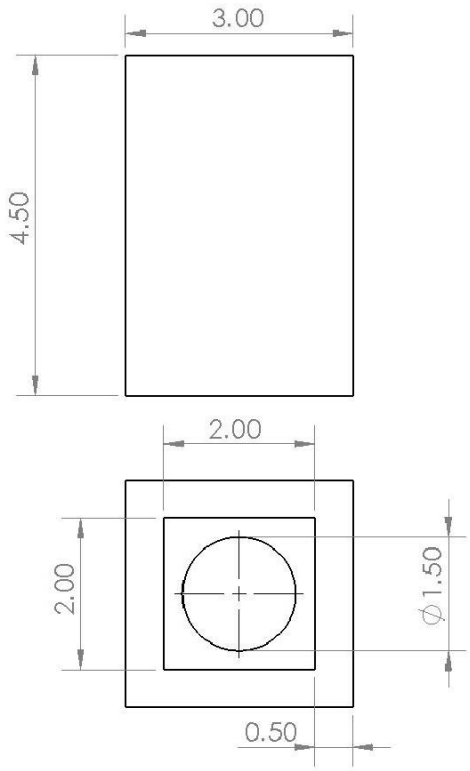


Figure 1.2.g: *Support Shaft Diagram*

Section 1.2.h: Lid

It was planned to use a wooden case that sits over the top of the final design. An example of this component can be seen in Figure 1.2.h-x. The lid of the anemometer was cedar wood with a thickness of .75” with dimensions of 13” x 13” x 8.75”. The total weight of the lid was 7.583 lb.

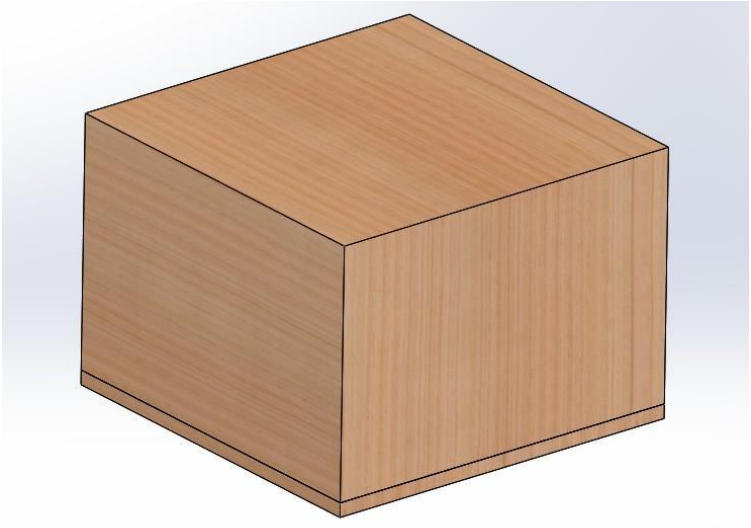


Figure 1.2.h-x: *Example of Lid (Missing the Latches on the Sides)*

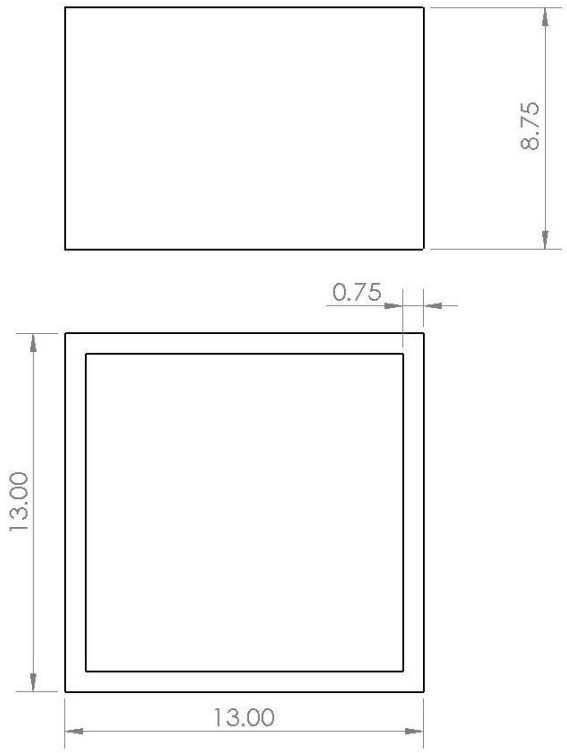


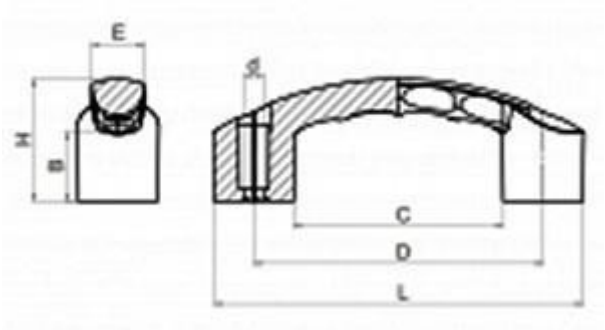
Figure 1.2.h-y: *Lid Diagram*

Section 1.2.i: Latches

Figure 1.2.i: *Image of Latch*

It is planned to use 4 Penn-Elcom L0566K Surface Mount Draw Latches. In this design the screws have 0.3225 pounds of shear force acting on them, and they are rated for up to 2.6 lbs of shear force.

Section 1.2.j: Handle



(Left) ***Figure 1.2.j-x:*** *Image of Handle*
(Right) ***Figure 1.2.j-y:*** *Handle Diagram*

It was planned to use a PGH-1401 plastic top mount handle from Industrial Components Group to carry the weight of the entire system. This would be attached via 2 screws into the top and should hold the 7 lb load applied to them, as they were rated for a strength of 97.66 lbs.

Section 1.2.k: Plans for Manufacturing & Assembly

The manufacturing will be handled using the machining equipment of the Engineering Department. The assembly of the components primarily uses epoxy, and the latches and handle use screws. The assembly and machining of the anemometer will be carried out by the team.

Section 1.3: Initial Design Electrical Details

Section 1.3.a: Circuit Overview

(Left) ***Figure 1.3.a-x:*** *Arduino Uno R3 Circuit Board*
(Right) ***Figure 1.3.a-y:*** *Photo Interrupter*

The rotational speed was to be found using the measurement disk and a photo interrupter sensor with an Arduino Uno R3. The uniform holes in the measurement disk would allow the light of the photo interrupter sensor to pass through, toggling its reading. With the distance between the holes known, the Arduino would be able to calculate the time that has passed between each reading, allowing it to calculate the rotational speed. The photo interrupter sensor would give the Arduino up to 2 times the number of holes in the measurement disk. The measurement disk would have 16 uniform holes, meaning the Arduino will be able to record up to 32 data points per full revolution. At higher rotational speeds, the Arduino would have to skip data points in order to maintain its accuracy of its readings. The multiple data points per rotation would allow the anemometer to be more accurate in outdoor environments, where the wind will not always be in the same direction.

The circuit needed multiple sensors and additions to perform the tasks necessary of the anemometer. The components used with the Arduino Uno are the following: power accessories, a data logging shield, a 16 x 2 character display, and two photo interrupter sensors. The planned wiring for the circuitry is available on the next page as Figure 1.3.a-z.

Section 1.3.b: Arduino Uno

The Arduino Uno, as seen in Figure 1.3.a-x, is designed to work with most 5v and 3.3v logic sensors and components. The Arduino would handle all of the electrical work with the sensors. The Arduino Uno is capable of being programmed to do almost anything, but most of the calculations will be done in post with Excel spreadsheets. The reason for this decision was to allow the Arduino to scan its sensors as fast as possible to improve accuracy.

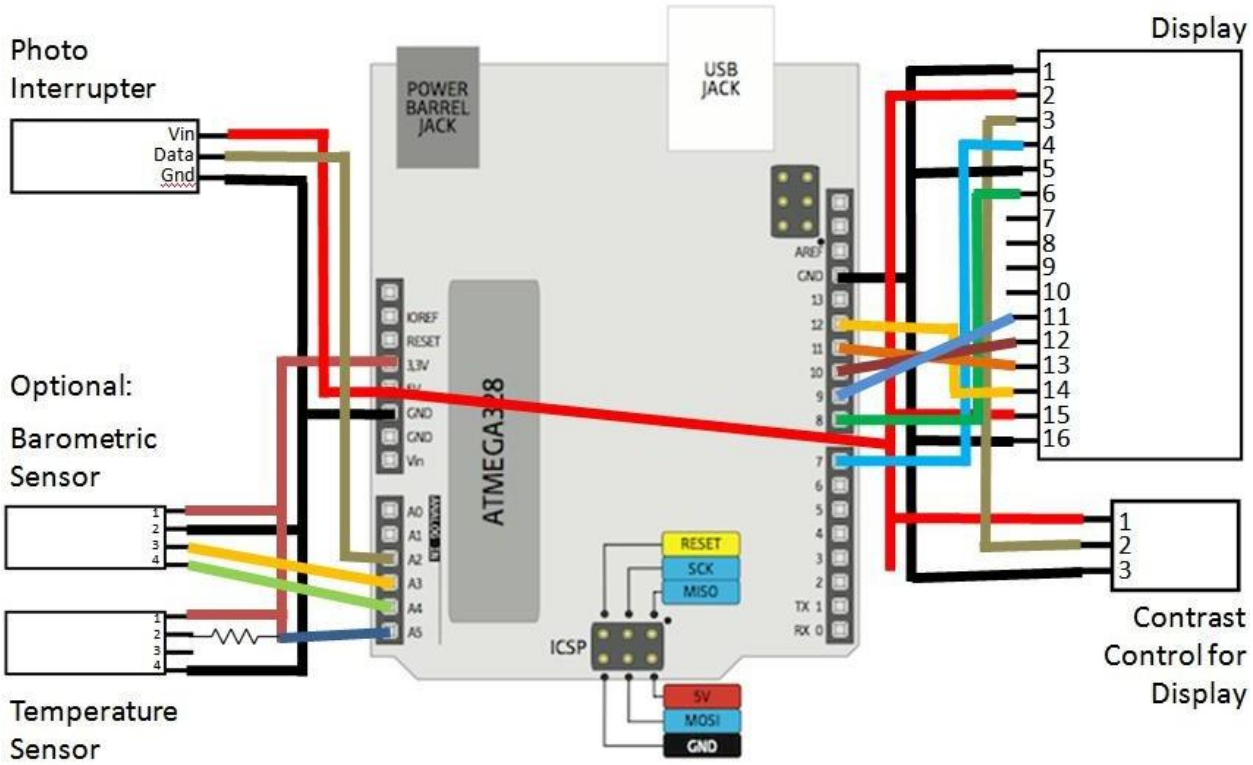


Figure 1.3.a-z: *Circuit Wiring Diagram*

Section 1.3.c: Power Accessories

(Left to Right) ***Figures 1.2.c-x, -y, -z:*** *PC Power, AC Outlet Power, & Battery Power*

The powering accessories for the Arduino Uno and its components were a A-B USB cable, an USB A outlet charger, and a 6 AA battery pack. The USB cable allowed the Arduino to be plugged in to a PC or the outlet charger, allowing it to run. If plugged in to a PC, the Arduino could be programmed and communicate data to the PC from its sensors. The battery pack can also provide power to the Arduino for at least 24 hours minimum. Accurate data on the battery life spans would require building the circuit and testing it.

Section 1.3.d: Data Logging Shield

Figure 1.3.d: *Image of Data Logging Shield*

This shield is a component that mounts on top of the Arduino Uno. It would allow the Arduino to log to any SD card as a text file. The logger also has a built-in clock, which would allow the Arduino to report the time the system was turned on and data recording began.

Section 1.3.e: Character Display

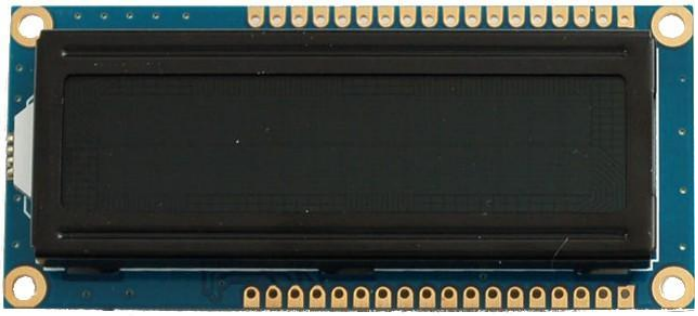


Figure 1.3.e: *Image of 16x2 Character Display*

This display would allow up to 32 characters to be shown on the screen at a time. This display also has variable colors for the backlight, which would be used for user feedback on what the system was currently detecting in addition to wind speed measurements.

Section 1.3.f: Photo Interrupters

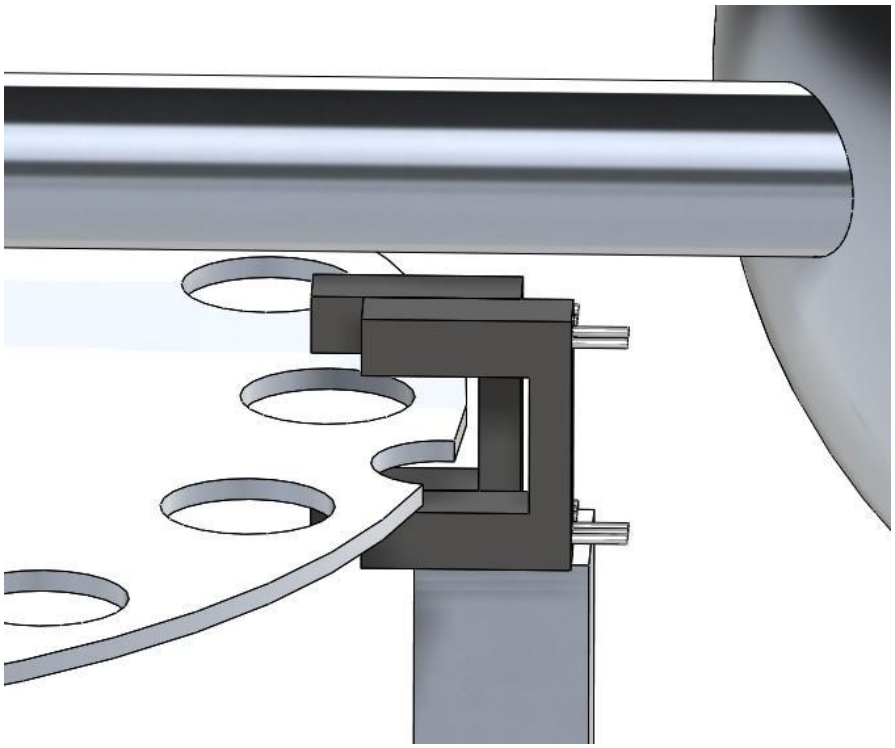


Figure 1.3.f: *Photo Interrupters Arrangement*

The circuit would have two photo interrupters. One photo interrupter would shine and detect its light along the very edge of the measurement disk. The disk would have notches along the edge near the 90 degree intervals. This photo interrupter would be used so that the Arduino Uno can detect where the cups are. An excerpt of the data sheet is found in Appendix A.1.

The second photo interrupter would be used to detect the breaks in its light due to the 16 holes in the measurement disk. The Arduino would find the time difference between the breaks to find the rotational speed. With this data, the Arduino would calculate the wind speed for the display, and would record the rotational speed for the Excel spreadsheets to process more thoroughly.

Section 1.4: Data Acquisition & Calibration

The Arduino would allow three methods of data acquisition. The display would allow the anemometer's user to take manual recordings of the system's measurements. When plugged into a PC, the Arduino would communicate directly with an Excel spreadsheet with the help of a third party program called GoBetwino. The Excel spreadsheet would be set up to automatically plot the data as it was fed to the PC from the Arduino. The third method was by using the data logger. The data logger would record a text file that can later be placed in the Excel spreadsheet. The advantage of the data logger is that it can operate faster than the third party software, and it would allow for automatic data recording in the outdoors.

It was planned to use the wind tunnel in room 343 of the ETCS building in order to calibrate the anemometer. This would be done by setting the wind tunnel to a specific speed and using the data for the speed of the air at the exit of the wind tunnel in order to test if the anemometer would calculate that speed correctly.

Section 1.5: Design Revisions

Some changes were implemented to ensure long term durability, to improve accuracy of wind speed measurements, and to improve the visual appearance of the device as an IPFW display piece. All of the wooden components were replaced, extendable feet and case supports were added, the storage units for the electronics were moved to fit the new structure design, the measurement disk was reconfigured, and the procedure for data logging and calibration was revised.

Section 1.5.a: Central Hub Material Change

Due to the desire to move away from wood to improve durability and appearance, a piece of nylon plastic that was available in the IPFW machining laboratory was used. The nylon component dimensions came out roughly the same as the wooden design.

Section 1.5.b: Structure Changes

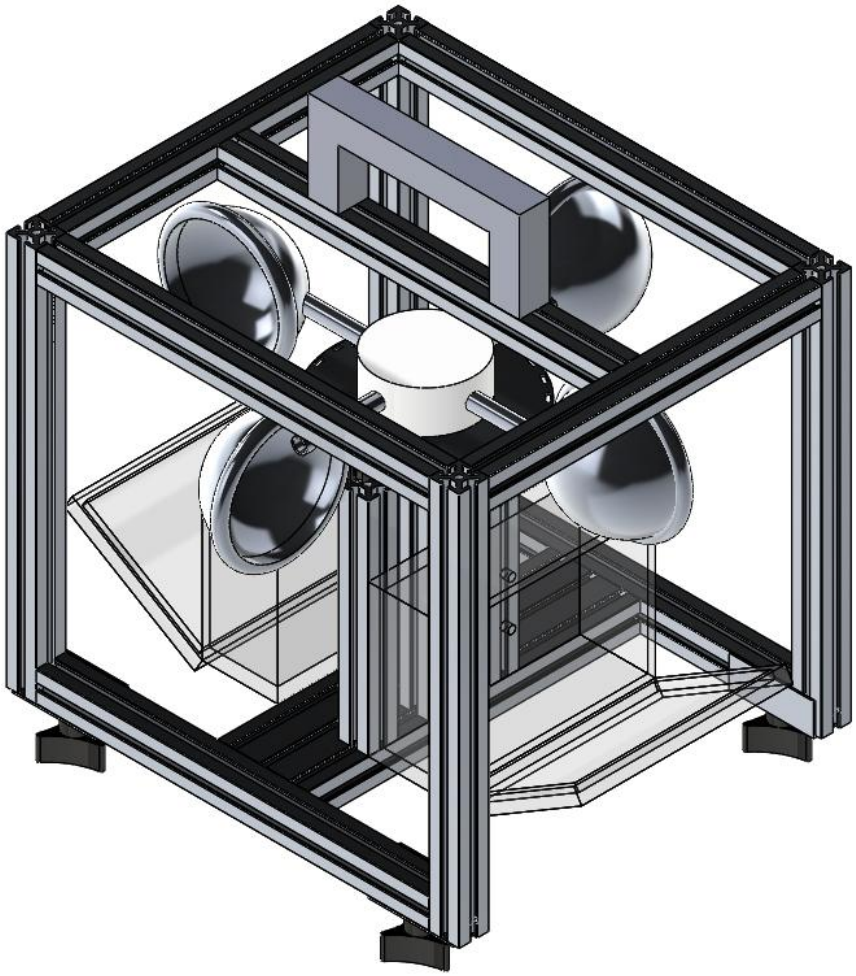


Figure 1.5.b-x: *80-20 Aluminum Extrusion Structure Revision*

The structure and integrated case for the anemometer was redesigned using modular 80-20 aluminum extrusion 10 series. As seen in Figure 1.5.b-x, the 80-20 creates the framework for creating the desired box shape for the integrated case. However, due to differences in the aluminum extrusion and concerns of the electronics storage, many dimensions were slightly altered. The following are the general details of the changes.

Base & Central Shaft:

1. The height of the 80-20 units creating the central shaft were increased to 7.5 inches. The reason being was to give extra clearance above the electronics storage to insure that the air flow around the storage units would be minimal.
2. The top of the central shaft was fitted with a 1/4in thick piece of plastic for the bearing to rest on.
3. The sensor stand on which the sensors are mounted was of height 7.875in.
4. The base was created using one 3x1x11in extrusion with two 1x1x13in extrusions connected on either end with fasteners.
5. Holes for the extendable feet were located .75in back from the edges of extrusions with through diameter of 3/16in and threading of 1/4-20.
6. The hole for the sensor stand was milled +-.5in from 1in away from the central stand with diameter 3/16in to allow for minor adjustments.
7. The holes for all other 80-20 extrusion connections were made to center the extrusion with a hole diameter of 3/16in.
8. The holes for the electronics storage were to be drilled 1in and 3in from the bottom of the central shaft of diameter 3/16in through with 1/4-20 threading.

Lid (Cage):

1. The cage was designed with dimensions of 15x13x13in. The cage was lengthened so that the corner extrusions would be outside the 13x13 base dimensions. The cage was also heightened to compensate for the height increase of the central shaft. Additionally, this gave additional tolerance room for differences from the real components and modeled components.
2. The cage had four vertical corner extrusions of length 13in, each with 1x1in dimensions. Two extrusions of length 11in each connected two of the four corner extrusions. Three extrusions then bridged the two halves with a length of 13in. The third bridging extrusion was where the handle was mounted.
3. The cage connects to and carries the anemometer by using thumbscrews at the bottom of the vertical corner cage extrusions to the base of the anemometer.
4. All holes for 80-20 connections were 3/16in all the way through the 80-20.
5. The holes for the thumbscrews were 1/4in all the way through the 80-20.
6. All holes were located .5in from the edge of the 80-20, with the exception being the handle and center extrusion, which were located to center of the said component.
7. Each extrusion end with a fastener was threaded to 1/4-20.

Section 1.5.c: Extendable Feet & Supports

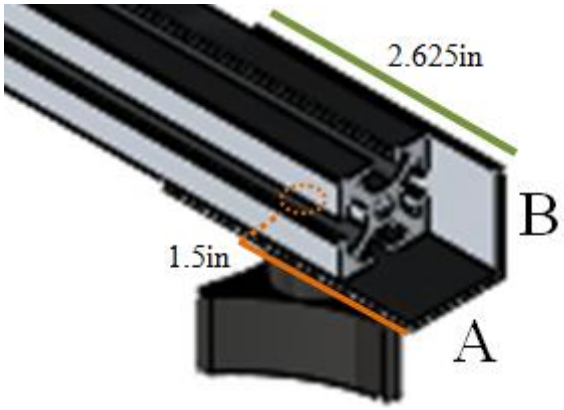


Figure 1.5.c: *Extendable Feet & Supports*

Upon assembling the 80-20 structure, it was noticed that the base was not level. Additionally, it was difficult to screw in the thumbscrews, due to them being close to the table surface. Threaded knobs were added as extendable feet to eliminate both issues.

An additional issue was the height of the holes on the cage and the base were not equal, and the addition of the knobs amplified the issue. The issue was eliminated by using small 90° V bend aluminum pieces. The aluminum pieces were held in place by the extendable feet, and a small amount of epoxy putty applied to the aluminum would completely eliminate the issue altogether.

The dimensions of the 90° V bend aluminum pieces were a length of 2.625in with a hole for the extendable feet of 1/4in diameter located 1.5in back from the end that juts out. A length of 0.75in juts out to catch the cage. The pieces for both sides were not symmetric, as labeled in Figure 1.5.c, for one leg the hole was to be drilled on side A, and for the other side the hole was to be drilled on side B.

Section 1.5.d: Electronics Storage Units

To accommodate all of the electronics, storage units of specific dimensions had to be sought. The primary change in the design only amounted to the orientation of the storage units. The storage units were mounted on to the side of the central shaft rather than to the base.

Section 1.5.e: Measurement Disk Reconfiguration

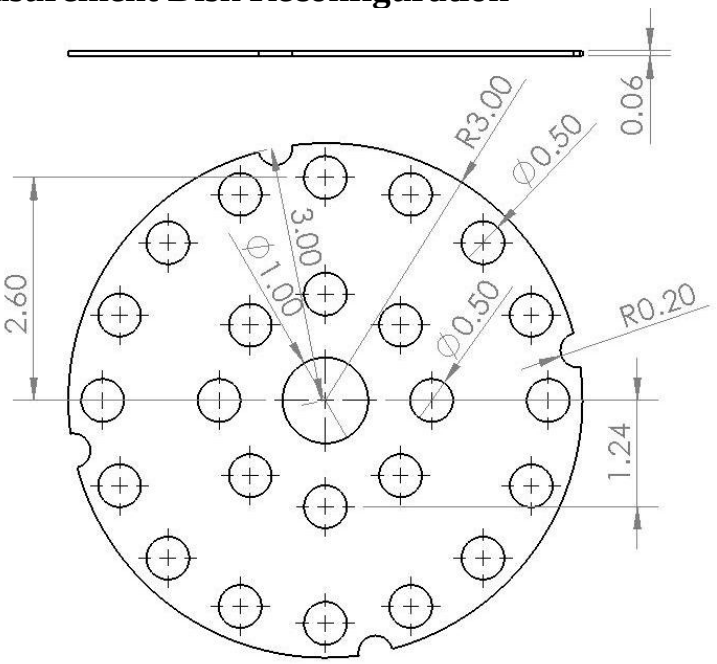


Figure 1.5.e-x: *Original Measurement Disk*

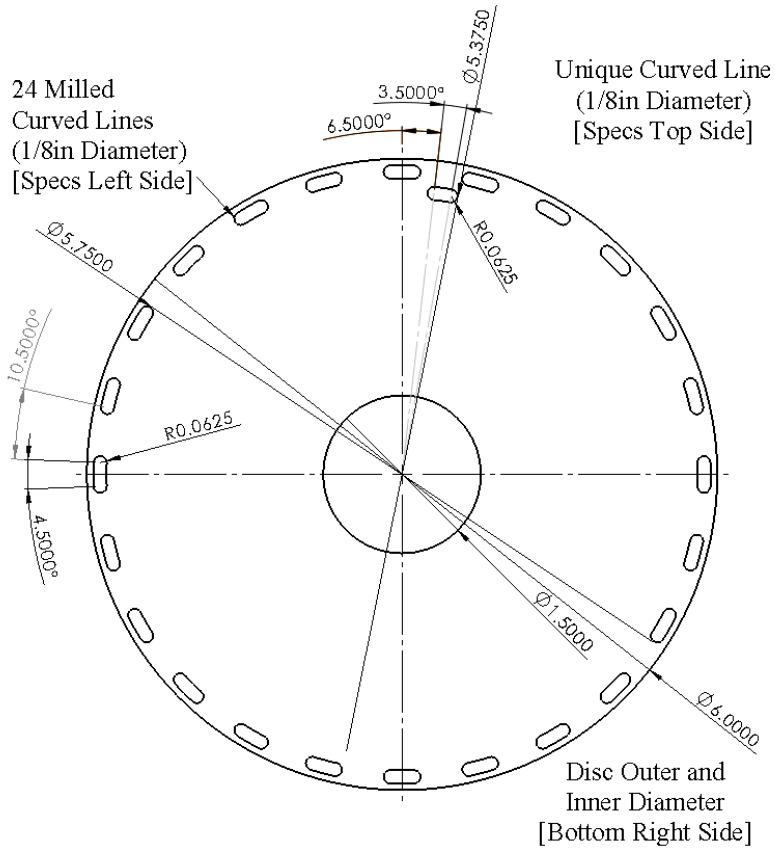


Figure 1.5.e-y: *Revised Measurement Disk*

As seen in Figure 1.5.e-y and Figure 1.5.e-z on the previous page, the measurement disk was redesigned from having holes to milled lines along a radial path. The problem with drilling normal holes was going to be the high likely hood of the position sensor and RPM sensors sharing readings. The only alternative solution was to use smaller holes, but then the Arduino Uno would have trouble keeping up with the time frames it would have to scan the small size of the nodes, possibly effectively reducing the number of functional nodes by half. The new measurement disk had 24 holes producing 48 sample points, whereas the old disk only had 16 holes and 32 sample points. The milled lines allowed the Arduino Uno to have simpler code, more data collection, an increased range of tolerance for functional sensor positioning.

Section 1.5.f: Data Logging & Calibration Revisions

The data logging was changed from using the third party software to custom written Python code that would work automatically. The plan for calibrating the anemometer was also revised. First, the anemometer would be hooked up to a CNC mill to spin it at a constant speed to remove any inaccuracies that may occur from the disk not being centered upon the bearing. Then, instead of the exhaust of IPFW's wind tunnel, the team was to go to Purdue to make use of their large Boeing wind tunnel. The rest of the calibration could be done after collecting the data from Purdue.

Section 2: Construction

Section 2.1: Component Manufacturing

Most of the components only needed minimal manufacturing processes to produce the component. All machined metal components were sanded or ground down to be smooth. Components were made close to their respective design measurements with only minor alterations. The following is how each component was machined in order to be assembled:

Section 2.1.a: Hollow Hemisphere Cups

The cups only needed to have holes machined into them. Using the manual mill in the IPFW machining laboratory, the cups were clamped in the machine equal distance on either side. Holes were drilled into the cups using incremental sizes of drill bites, ranging from 1/8in up to the 1/2in desired size. The cups were slightly bent from the clamp, but after hammering the bend out the cups were ready for assembly. Number of units: 4.

Section 2.1.b: Crossmember Rods

The 1/2in aluminum hollow rods only needed to be cut to their desired length of 3.875in using the horizontal saw of the IPFW manufacturing laboratory. Number of units: 4.

Section 2.1.c: Central Hub

The nylon central hub was cut off a larger chunk of nylon using a manual lathe from the IPFW manufacturing laboratory. The nylon hub was then planed and turned using the lathe. A small stub was left on the hub of diameter 1 1/2in and approximately height of 1/4in. This stub allowed the hub to snap together with the bearing and provide a stronger connection for the epoxy in assembly. Then the hub was marked for 90° from the center by eye, so that the 1/2in holes could be drilled. These holes only required a single pilot hole of 1/8in.

Section 2.1.d: Measurement Disk

The measurement disk was machined by Onxx Tool to specification. Onxx Tool replace the aluminum with stainless steel. The best alternative method for producing this component is using the CNC mill that the IPFW manufacturing laboratory has in possession. Onxx Tool were recommended due to time constraints on the faculty of the CNC machines.

Section 2.1.e: 80-20 Structure

Figure 2.1.e: *Milled Hole for Sensor Stand*

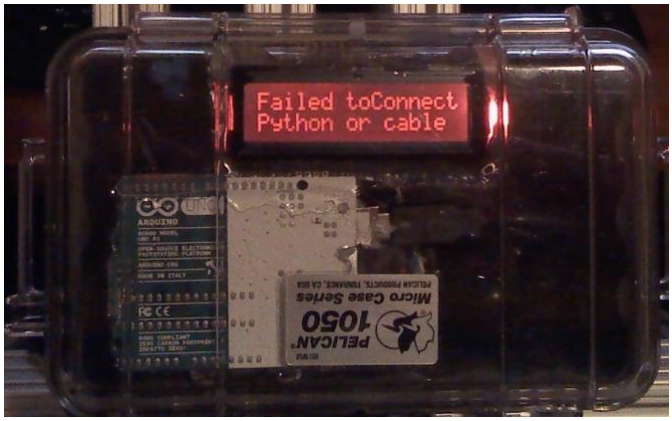
The 80-20 aluminum extrusion 10 series structure was mostly completed through Neff Engineering ordering it from the 80-20 suppliers. The only additional work required was drilling holes and tapping threads for the extendable feet and storage units. The milled hole for the sensor stand was done using a drill press making repetitive holes to avoid scheduling a second day at IPFW's machining laboratory. Total Holes with Threading: 12.

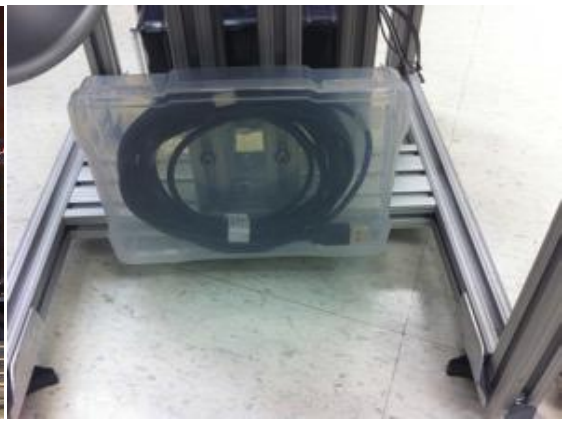
Section 2.1.f: Cage Supports

Figure 2.1.f: *Construction of Cage Supports*

The added cage supports to the base only required being cut at lengths of 2 1/2in from stock 90° bended aluminum. The edges were ground to be smooth. During assembly, the edges were bent into shape using a vice grip to help the cage get into position easier. Holes of diameter 1/4in were then drilled according to the design specifications.

Section 2.1.g: Electronics Storage





(Left) ***Figure 2.1.g-x:*** *Circuit Storage*
(Right) ***Figure 2.1.g-y:*** *Cable Storage*

The only machining for the electronics storage required four precise holes in each storage unit to match the holes on the 80-20 structure's central shaft. Other holes were drilled for cabling, but did not require any specific dimensions. Cabling holes were all drilled at an angle such that water should not drip into the case. Holes were drilled at the bottom of the case for the scenario that the storage unit needs to drain water. Total Holes (precise locations): 8. Total Cable Holes: 6.

Section 2.1.h: Electronics

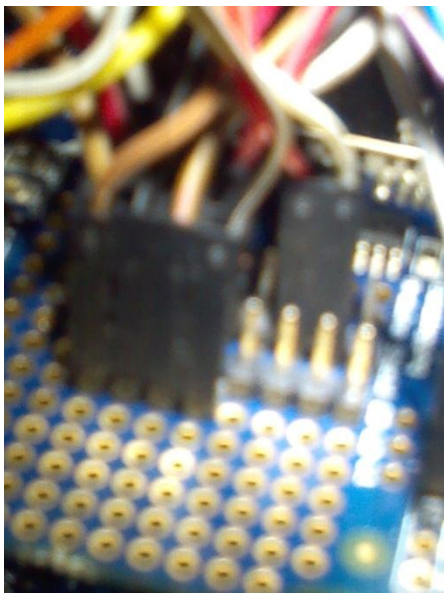


Figure 2.1.h-x: *Common Leads of Adafruit Data Logger*

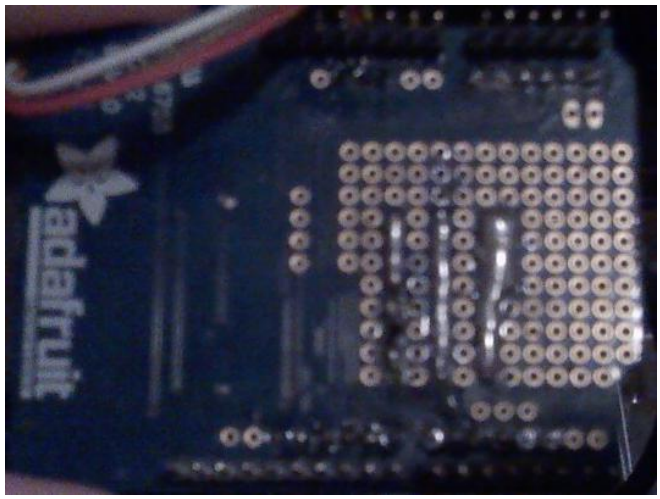


Figure 2.1.h-y: *Underside of Adafruit Data Logger*

The only electronic components needing any kind of assembly was the Adafruit Data Logger and the sensors. The data logger's construction amounted to using header pins to create enough common grounds and 5v sources to make cabling the units together simpler, as seen in Figure 2.1.h-x and in the underside in Figure 2.1.h-y. Additionally, header pins need to be soldered to the data logger to pin into the Arduino Uno. This was done easily by placing the header pins into the Arduino Uno and resting the data logger on top of the Arduino Uno while soldering the pins.

The sensors only needed to be soldered to their breakout boards with a 220 Ohm resistor.

Note: using solder to bridge the header connections adds unnecessary resistance, which could throw off analog readings slightly. Since none of the components are highly sensitive to analog readings, the header pins could be bridged using only solder. If any components were sensitive and needing highly accurate voltage readings, the headers would need to be connected together using a cable.

Section 2.2: Assembly

The assembly was compartmentalized into the spinning portion of the anemometer, the structure, and the electronics. After the three were assembled individually, the spinning portion and structure were assembled together. Then the sensors were adhered to the sensor stand.

Section 2.2.a: Assembling the Spinning Portion

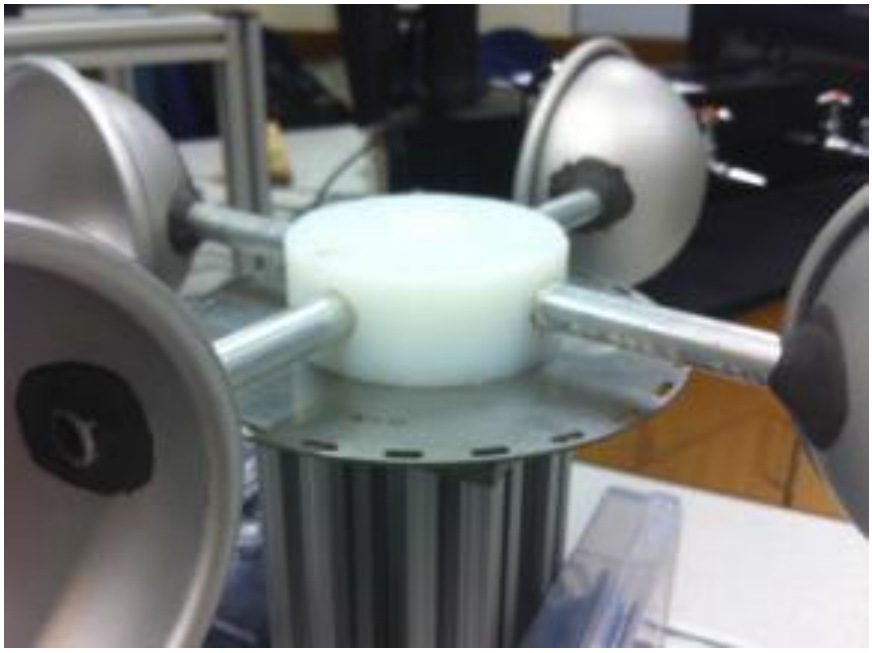


Figure 2.2.a: *Assembled Spinning Portion*

The rods were pressed into the nylon hub using a press in the IPFW machine laboratory. Since the nylon had shrunken around the 1/2in holes, no epoxy was required for holding the aluminum rods in the hub. The cups were then mounted one at a time to get the 90° connections using JB Weld SteelStik as a epoxy putty.

The disk and bearing were then adhered to the hub using JB Weld Kwik. Due to a little excess of adhesive, the bearing was spun while the adhesive cured to ensure the bearing would remain loose. After curing, the edges of the bearing had to be chiseled at a little to remove some excess adhesive.

In order to clean out the bearing of crud and any adhesive, the bearing was soaked in isopropyl rubbing alcohol upside down so that particles could either float to the top or sink out through the bottom. After several soaks and spinning it while soaking, the bearing was free from most particles.

Section 2.2.b: Assembling the Structure

Assembling the cage structure of the 80-20 only required an Allen wrench. The end fasteners were screwed in most of the way into the extrusions. Then the extrusions were slid onto their coupled extrusion and tightened by the fasteners through the holes in the coupled extrusion using the Allen wrench. The handle was attached the same way.

The base and central shaft were assembled in the same fashion, using the plastic plate to keep the central shaft extrusions apart to form a 3x3 cage. One side extrusion of the base has to be loosened and slid off in order to place the sensor stand, which was done once the sensors were ready to be placed permanently. The extendable feet were screwed on with the cage supports. Leftover JB Weld SteelStik from the cup mounting was used to correct the height difference between the holes of the cage and base.

Section 2.2.c: Assembling the Electronics

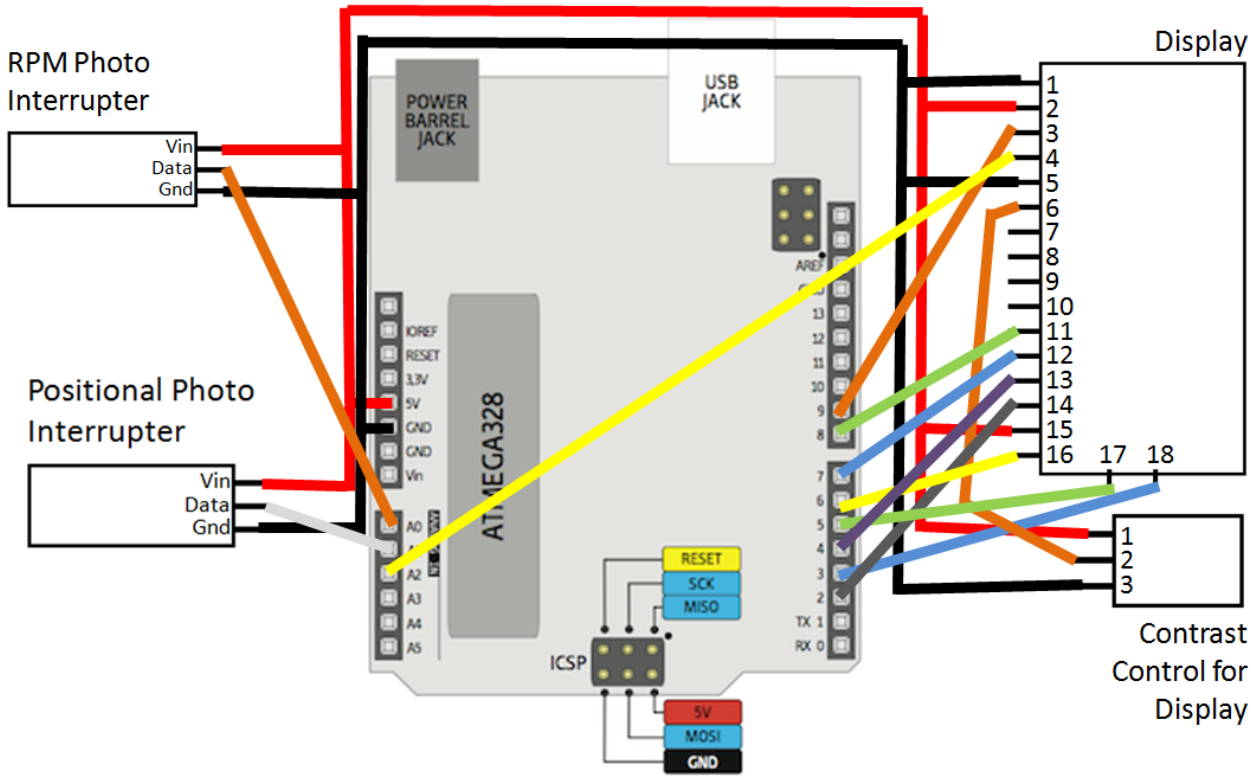


Figure 2.2.c-x: *Wire Diagram*

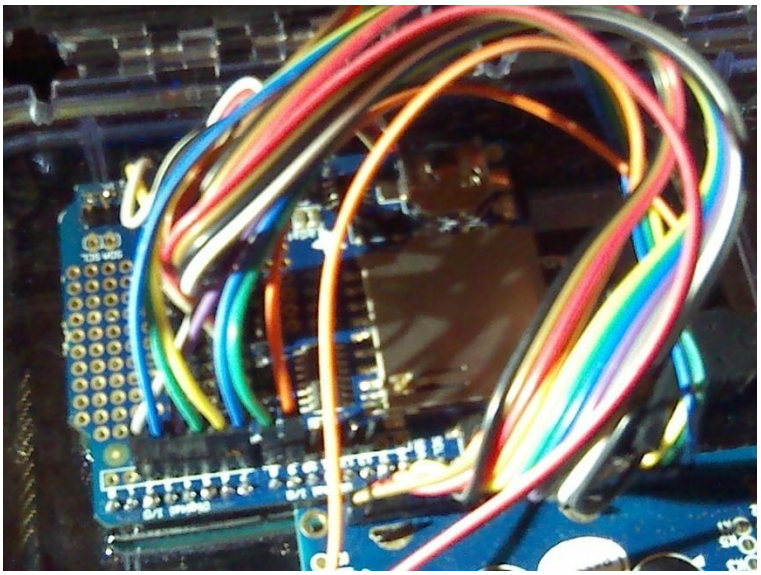


Figure 2.2.c-y: *Electronics Wiring*
Note: White & Red cables were used exclusively for V supply
Black & Brown cables were used exclusively for GND

The cables used were M-M header cables, so in order to create M-F cables the heads had to be bent back and forth to break them off, turning the header into a F connection. The M connection of each cable had to be soldered into the components as such:

1. Three cables had to be soldered to the 3v, 5v, and GND of the Adafruit Data Logger, which are plugged in to their respective common header pins.
2. For each sensor, three cables had to be soldered into each. The heads broken to two-thirds length, but they were also bent 75° so they would not stick out and have the cups bump in to them.
3. Fourteen cables had their heads broken to one-third length and soldered into the display.
4. Three cables were broken to be F-F for the contrast controller of the display.

The last two steps for assembly is sliding the data logger onto the Arduino Uno, and then connecting the F cable connections to the data logger.

Section 2.2.d: Mounting Sensors & Electronics

Mounting the sensors to the sensor stand requires that the spinning portion be adhered to the structure. JB Weld Kwik was used to adhere the bearing to the center of the plastic plate of the structure. The sensor stand was then positioned to give enough lip for the sensors to rest on it. The sensors were then held by hand to find the proper positioning by watching their readouts as the disk was spun back a forth over the single position node. Then Devcon 2 Ton Epoxy was used to adhere the sensors to the sensor stand. This epoxy was slow to cure, so it was left to sit in its dish for nearly 15 minutes before being applied to hold the sensor in the correct position. The sensors were held by hand during the curing process over the next 10 to 20 minutes.

Figure 2.2.d: *Sensor Positioning*

Two problems arose with mounting the sensor during assembly. The first was the sensor stand was too short and had to be lengthened using stock aluminum pieces cut to be 1x1 in pieces. The second was the sensors were first attempted to be adhered using super glue and JB Weld SteelStik. The super glue was intended to coat the electronics to prevent any risks with the SteelStik, but either the super glue failed to adhere to the electronics or the SteelStik was abrasive to the super glue, resulting in the SteelStik effectively shorting the sensors. The adhesive had to be removed and the sensors replaced.

Mounting the Arduino Uno and display in the storage unit was done using the Devcon 2 Ton Epoxy.

The bearing was lubricated using State Chemical's DRI All Purpose Penetrant lubrication after it had been adhered to the structure.

Section 2.3: Software Development

The software was designed to do all the desired tasks with the priority order of: automation, reliability, accurate measurements, error detection, demonstration appearance, ease of data reading, and ease of modification. The software was broken down into functions to reduce memory usage. The software uses approximately 90% of the Arduino's memory, so much optimization had to be done to minimize memory usage and increase functioning speeds. The anemometer can detect reverse motion, so reverse motion is used to set the mode of the code, and a combination of reverse and forward motion sends a confirmation to switch modes, else the current mode is kept when rotating forward. The following is a summary of how the code operates:

Global Variable Initialization

Setup:

Start up display
Define inputs and outputs for pins
Call function **Bootup()**
Calls built-in default function **Loop()**

Bootup():

Check for SD card, give time for user to respond
Disable SD card writing, or continue on to writing the SD file
- Verify that the SD file was written, or report the error on the screen
- - Check and write the time to the SD file, or report an error with the time reporting
Check to see if the Arduino is connected to the PC using Python
Respond with an error screen to give time to the user to set up Python or plugging in the cable, or continue on by attempting to report the time
Disable PC writing if the user failed to set up the PC connection, or report any error with the time reporting

Loop(): runs indefinitely while the device is on

Check if in Mode 1, 2, 3, 4, 5, 6; default is Mode 1

Mode 1: Normal

Calls **ScanNode()**
If position is known, calculates RPM and wind speed
Checks to see if anemometer needs to enter a faster mode or not
If not, writes data as normal
If so, writes data as changing modes

Mode 2: Calibration

Resets calibration variables
Finds node 0 using **ScanNode()**
Begins calibration by scanning one node per revolution, starting with 0

Scans the same node a few times to take the average time
Checks to see if the difference between times and the average is within tolerance
If yes, moves on to the next node
If not, resets the calibration and writes data as a failure
At the end of all the nodes, checks to make sure the speed is the same as when the test began
Writes results

Mode 3: Equipment Test

Changes the display to different colors to let the user verify if the colors are accurate
Writes character on the display to let the user verify if the display is writing properly
Begins showing the voltage readouts of the sensors until the user exits the mode

Mode 4: Reset

Resets variables
Calls **Bootup()**

Mode 5: High Speed Mode

Enters Mode 5 if Mode 1 detects high RPM or if in Mode 6 and slowing down
Only enters Mode 5 on even nodes
Writes results every even node, otherwise functions the same as Mode 1
Currently handles RPMs between 100 and 200
Only exits to Mode 1 when detects below 80 RPM
Only enters Mode 6 when detects above 200 RPM

Mode 6: Ultra Speed Mode

Enters Mode 6 if Mode 1 or Mode 5 detect RPM over 200
Enters Mode 6 at any time
Writes only on node 0, otherwise functions the same as Mode 1
Currently handles RPMs above 200 up to roughly 5000 RPM
Exits to Mode 1 if RPM below 80, and exits to Mode 5 if RPM below 180

ScanNode():

Waits for the next node, knowing which node it is on by the toggling between high and low voltage readings. If in Mode 5, waits for two node passes instead.
Calculates the cycle time between current and previous node reading
If position is not yet known, checks the position sensor to see if it can find position
Else, checks to see if position sensor is reporting its single node
If position sensor is reporting, it checks whether it is reverse, reverse and forward, or forward. Checks reverse & forward by seeing if it has passed exactly 48 nodes
If reverse, stops data writing calls **ReverseDisplay()**
If reverse and forward, stops data writing, sets the variables for **ReverseDisplay()**, and if reverse was detected before reverse and forward, calls **ReverseDisplay()**
If forward, resets all variables for **ReverseDisplay()**
Checks to see if it is in the middle of moving forward and reverse, if so it stops writing results in Mode 1

ReverseDisplay():

Checks variables using a series of if statements to know what mode is currently selected
Updates the display regarding the mode selection
If reverse and forward is detected six times, it switches to the selected mode
If the display has been disabled previously, all results will turn the display back on

Section 3: Calibration & Testing

Section 3.1: Durability Testing

No measurable tests were done for testing durability, but after the construction was completed the strength of the connections and components were tested by hand. These tests were conducted by hitting components with a moderate amount of force to verify that someone bumping into the anemometer would not alter results or damage components. This included testing what would happen if the anemometer came to a complete immediate stop if rotating at high speeds. No components showed signs of looseness or damage.

Section 3.2: Calibrating the Sensors for RPM

Prior to calibrating the anemometer to find the wind speed, the sensors need to be calibrated to measure RPM. This was needed to improve accuracy due to the disk possibly not being centered upon the bearing and due to the possibility of the photo interrupter scanning light not exactly in the center of the sensor. Additionally, due to the previous two reasons, the measurement disk's design did not have the holes and solid portions exactly equal in size.

The calibration was performed using one of the two CNC mills at IPFW's machining laboratory. John Mitchell created a jig that would allow the CNC mill's spindle to rotate the anemometer. The cable box was removed from the anemometer so that the anemometer's base could be clamped in the CNC machine. John then manually positioned the CNC mill's spindle over the anemometer and lowered it on in small increments, checking to see if it was centered between each lowering. The anemometer was hooked up to a laptop through the side window of the CNC mill. The CNC mill was then manually incremented up by 10% increments till it reached 80 RPM. The CNC mill was left running for twenty minutes, producing five calibration results for the anemometer. The anemometer was then removed in a similar fashion and had its cable box replaced. The result of the calibration was finding each node's contribution to measuring RPM with ±.5% accuracy.

Section 3.3: Wind Speed Testing Procedure

The team travelled to Purdue to use the Aerospace Sciences Laboratory's Boeing wind tunnel. Under Dr. John Sullivan's supervision, the team learned how to operate the equipment and prepare the anemometer. Two pieces of equipment were damaged on the wind tunnel. The first was the wind tunnel was unable to attain speeds higher than 50 km/hr. The second was the display for the velocity was not working, so the wind speed was measured using a hand held TSI VelociCalc Air Velocity Meter, pictured in Figure 3.3-w.


(Left) ***Figure 3.3-v:*** *Purdue's Boeing Wind Tunnel*
(Right) ***Figure 3.3-w:*** *TSI VelociCalc Air Velocity Meter*

The anemometer had to be tied down to a circular wooden disk. The team was provided with an older disk that they could freely drill holes into. The anemometer was tied down through the milled hole below the sensor stand, and two extendable legs were tied down. Only one hole was drilled for the sensor stand zip tie, as it looped around the central gap of the disk, seen in Figure 3.3-x. The legs each had two holes drilled and the zip ties were placed around the leg. The zip ties were given a half inch of slack, so that it would be possible to possibly see the anemometer reach its toppling speed as it would begin to lift off and be held by the zip ties. However, it was revealed afterwards of the damaged fan of the wind tunnel, and its incapability of reaching the toppling speed of the device. The cable was fed through the center of the disk, and then the center of the disk was duct taped to reduce interference with the wind flow.

(Left) ***Figure 3.3-x:*** *Zip Tying Down the Anemometer*
(Right) ***Figure 3.3-y:*** *The Anemometer Placed in the Wind Tunnel*

The door to the wind tunnel was raised, and the wooden disk was lifted and placed into its socket in the wind tunnel. The wooden disk was tightened down by placing wedges between the acrylic glass and short stubs that pivots out from under the disk, as seen in Figure 3.3-z. The door to the wind tunnel was then lowered, and the anemometer was ready for testing.

Figure 3.3-z: *Placing the Wedges*

The testing procedure was performed by incrementally increasing the fan speed, waiting for steady state after 10 seconds, letting the anemometer collect data automatically for upwards to a minute at lower speeds, measuring the wind speed with the hand held air velocity meter through a hole in the side, and then repeating the process for the next fan speed. After collecting data over approximately 5 km/hr increments from 5 to 35 km/hr, the team tested the maximum wind speed of 50 km/hr the wind tunnel could produce. The team primarily jumped to 50 km/hr to attempt to verify the toppling speed of the anemometer, but it was far below the skeptical projected 50 mph of the design. After collecting data on 50 km/hr, the fan speed was reduced to 5 km/hr. The fan speed was then slowly reduced until the anemometer stopped spinning. Then the fan speed was slowly increased until the anemometer began spinning.

Section 3.4: Wind Speed Calculations

The data collected from Purdue was then processed to find how to relate the RPM to the wind speed accurately. Trends of the data were used to figure out the best way to approach the data analysis.

Section 3.4.a: Calculating Wind Speed

Data was collected at the wind speeds of 5 km/hr, 10 km/hr, 13.5 km/hr, 15 km/hr, 20 km/hr, 25 km/hr, 30 km/hr, 35 km/hr, and 50 km/hr. Many rotations were taken for each wind speed, and each rotation produced either 48 nodes of collected samples, or 24 nodes if the program was set to high speed mode. It was seen that the RPM reports from the anemometer varied from the beginning to the end of the data collection by small amounts. It is not possible to verify if this was due to the anemometer or the design of the Boeing wind tunnel. Therefore, it was decided to produce an equation using the minimum recorded RPM, the average recorded RPM, and the maximum recorded RPM over the entire data collection for that wind speed. The data for each wind speed's minimum, average, and maximum RPM is summarized in Table 3.4.a-x.

Table 3.4.a-x: *Minimum, Average, and Maximum Measured RPM at Varying Wind Speeds*

Measured RPM			**Measured Wind Speed (kmh)**
Minimum	Average	Maximum	
7.6	8.4	9.2	5.0
31.0	32.1	33.3	10.3
44.0	45.5	46.8	13.4
54.8	56.3	58.1	15.9
73.6	76.6	79.3	20.0
98.3	101.8	105.4	25.6
122.0	126.0	131.0	29.2
142.4	148.4	153.9	33.9
186.0	190.4	194.0	48.2

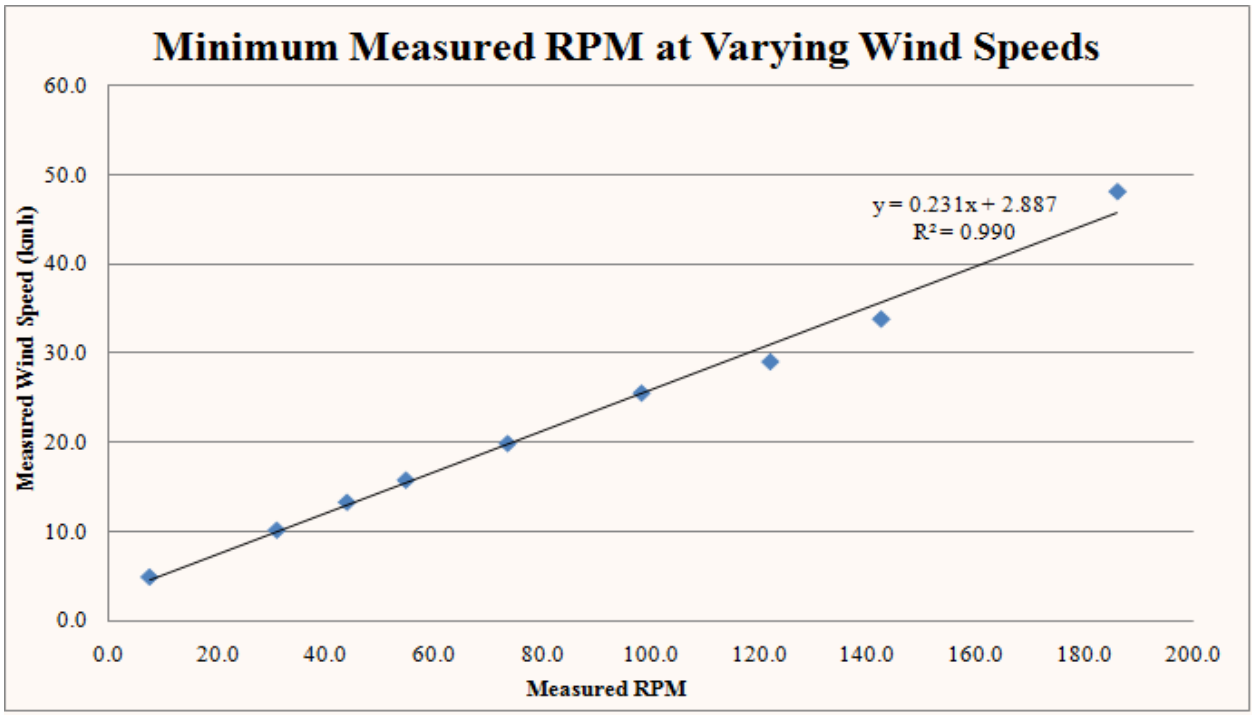


Figure 3.4.a-w: *Plot of Minimum Measured RPM at Varying Wind Speeds*

Upon plotting the methods as seen in the select sample in Figure 3.4.a-w, it was observed from the increasing error that the wind flow was in the state of transitioning from laminar to turbulent at the higher wind speeds. As a result, the data collected at 30 km/hr, 35 km/hr, and 50 km/hr did not fit accurately to a linear line, whereas the laminar region did. Since the wind tunnel was not able to produce wind speeds above 50 km/hr, data could not be collected to observe how the wind speed behaved during and after transition. The priority for accuracy was over the 5 km/hr to 20 km/hr region as defined by the problem statement. Plotting only the laminar region, the results are as shown in Figures 3.4.a-x, 3.4.a-y, and 3.4.a-z.

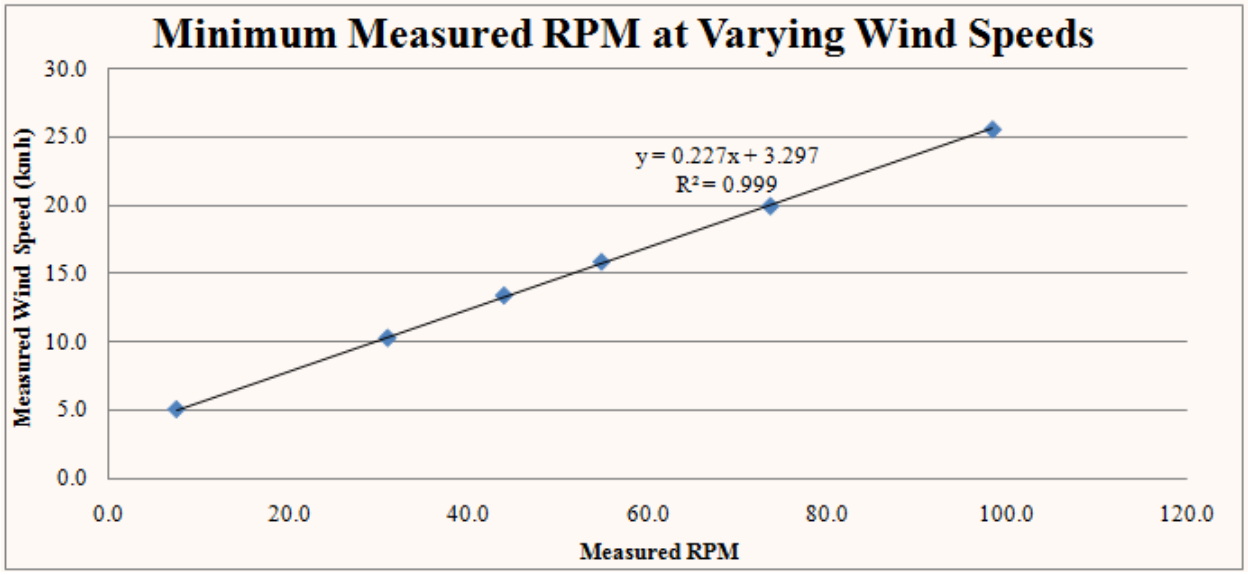


Figure 3.4.a-x: *<u>Minimum</u> Measured RPM at Varying Wind Speeds*

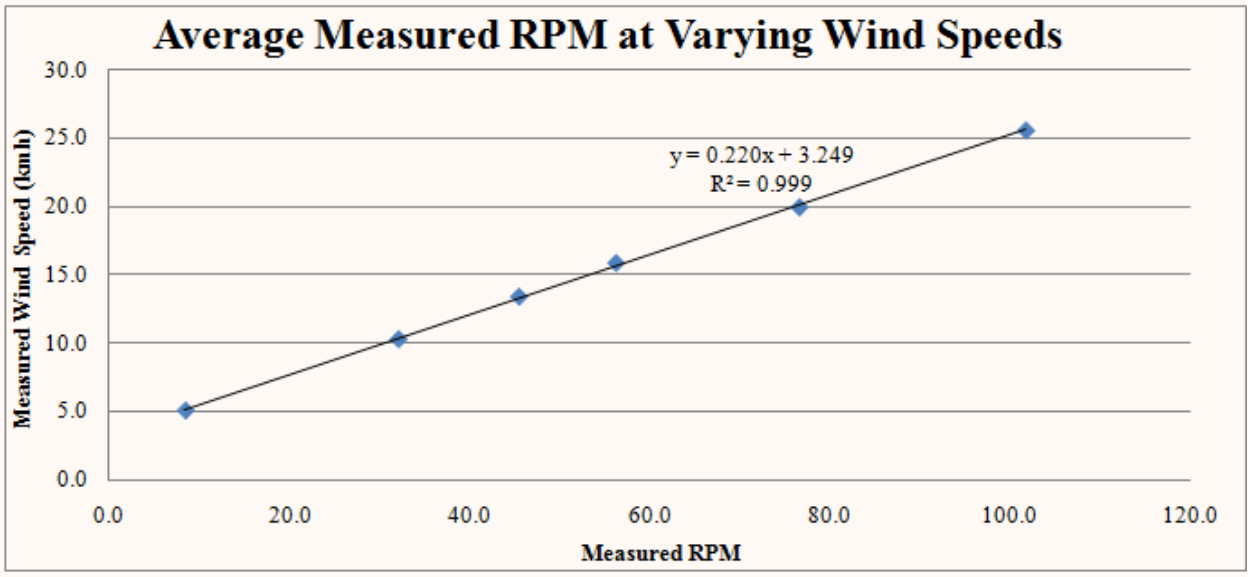


Figure 3.4.a-y: *<u>Average</u> Measured RPM at Varying Wind Speeds*

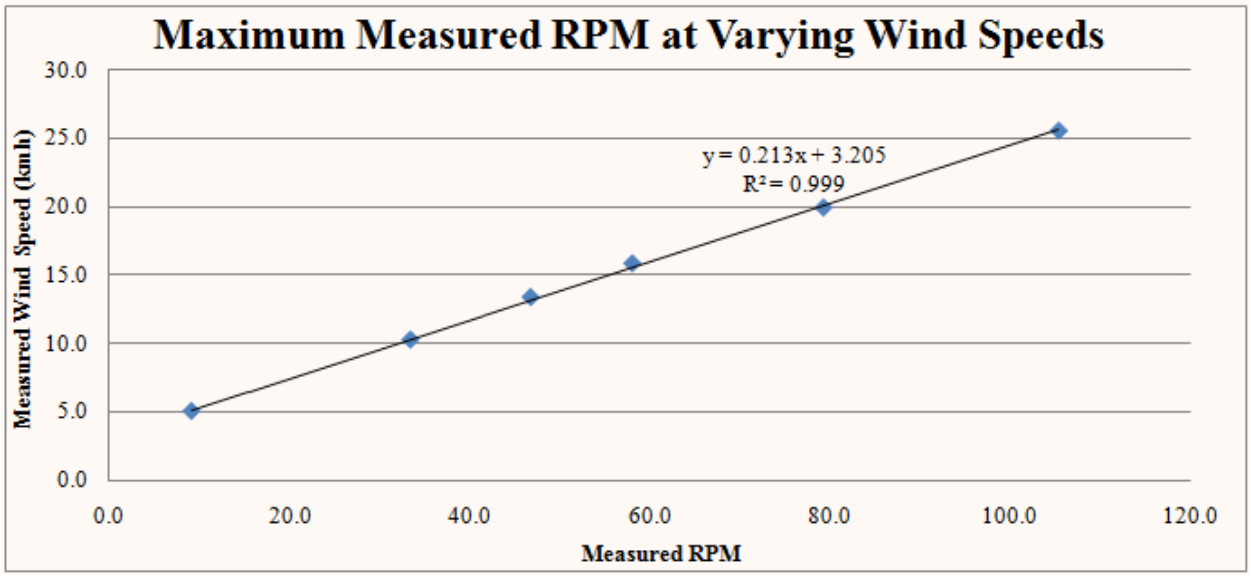


Figure 3.4.a-z: *<u>Maximum</u> Measured RPM at Varying Wind Speeds*

The accuracy of each was then compared in Table 3.4.a-y, revealing that the minimum recorded RPM was the best fit equation to the data for the laminar wind speeds. Therefore, the equation for calculating laminar wind speed accurately was .227*RPM + 3.297.

Table 3.4.a-y: *Accuracy of Minimum, Average, and Maximum RPM Conversion Equations*

Wind Speed (kmh)	Minimum		Average		Maximum	
	Resultant	Error	Resultant	Error	Resultant	Error
5.00	5.02	0.24%	5.11	2.02%	5.16	3.08%
10.26	10.34	0.80%	10.31	0.49%	10.30	0.44%
13.39	13.28	0.86%	13.25	1.03%	13.18	1.61%
15.88	15.74	0.86%	15.63	1.57%	15.58	1.87%
19.98	20.00	0.08%	20.11	0.64%	20.09	0.57%
25.63	25.60	0.11%	25.65	0.05%	25.66	0.10%
29.16	30.99	6.27%	30.97	6.21%	31.10	6.66%
33.95	35.62	4.94%	35.91	5.77%	35.99	6.02%
48.24	45.52	5.65%	45.14	6.42%	44.54	7.68%

Section 3.4.b: Testing Minimum Wind Speed to Spin

Testing the anemometer's minimum wind speed to begin spinning and measuring wind speed was found to be right below 5 km/hr. However, it was observed that the anemometer would always stop at the same position, and would spin faster at other positions. The anemometer would continue to spin at a wind speed of 4.1 km/hr for upwards of a minute before stopping, and sometimes it would restart spinning. Two select samples of the anemometer's RPM reading are provided in Figure 3.4.b-x and Figure 3.4.b-y. The first selected sample is when the anemometer came to a stop and then restarted. The second selected sample is the general pattern at the minimum wind speed when it did not stop. The second selected sample was taken after the first selected sample, with no changes in the wind speed of 4.1 km/hr.

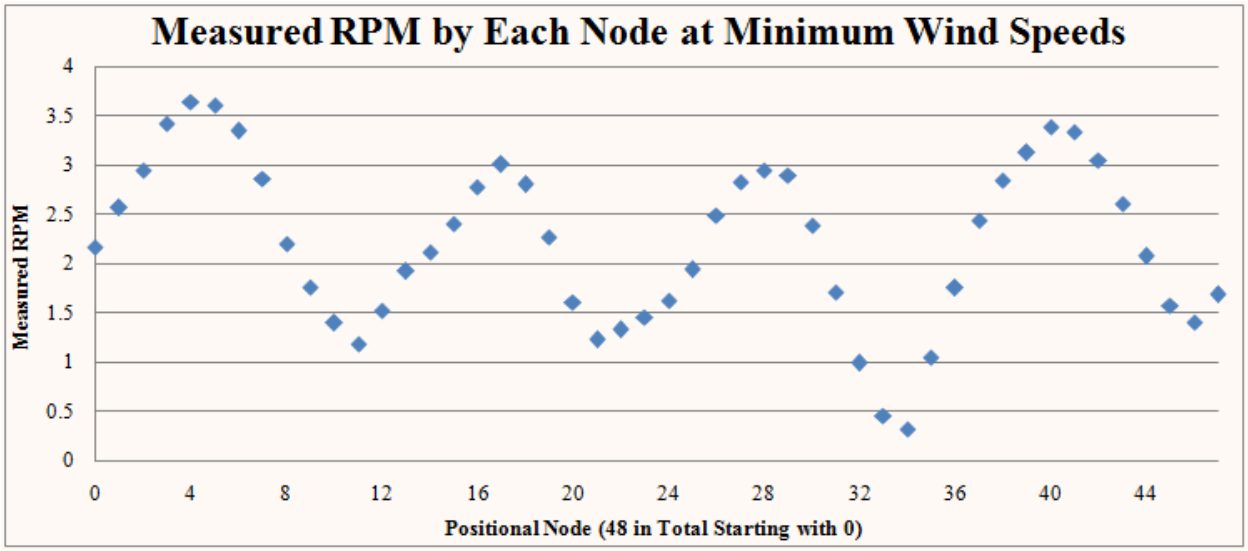


Figure 3.4.b-x: *Plot of Anemometer's RPM at Minimum Wind Speeds (Start/Stop)*

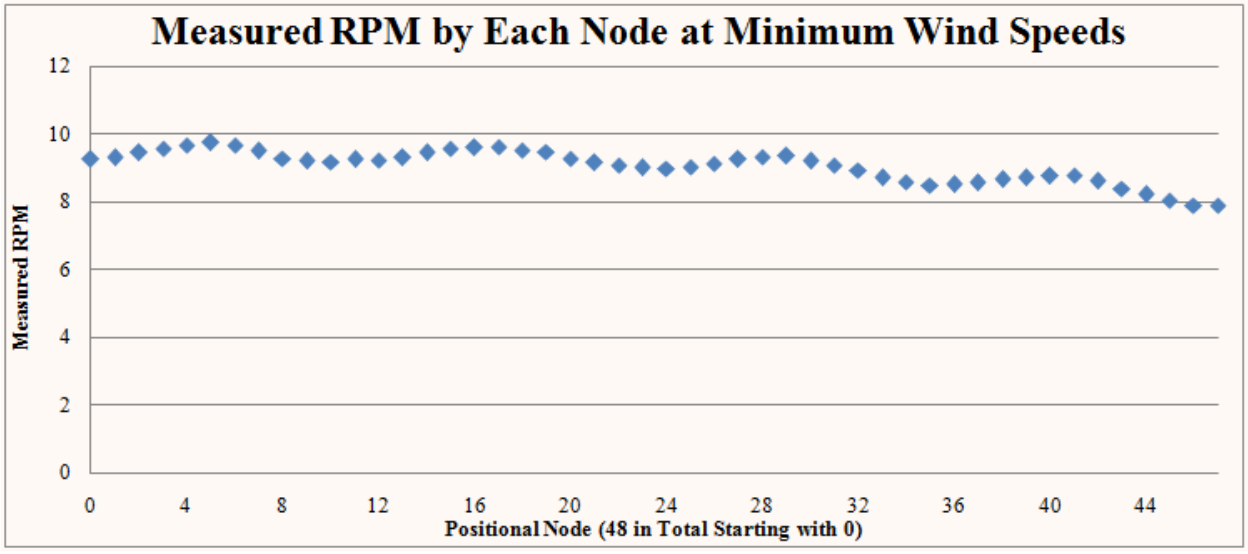


Figure 3.4.b-y: *Plot of Anemometer's RPM at Minimum Wind Speeds (Constant Rotation)*

Using the newly found wind speed equation and the same selected sample in Figure 3.4.b-x, Figure 3.4.b-z shows the calculated wind speed from the anemometer. The anemometer reported the correct wind speed measurement of 4.1 km/hr at the positions where the anemometer's cups are approximately 15 degrees prior to the anemometer reaching 90 degree angles. However, the wind speed equation was calculated based on the minimum peaks of RPM, so the anemometer will report these low speeds inaccurately, with up to 15% error.

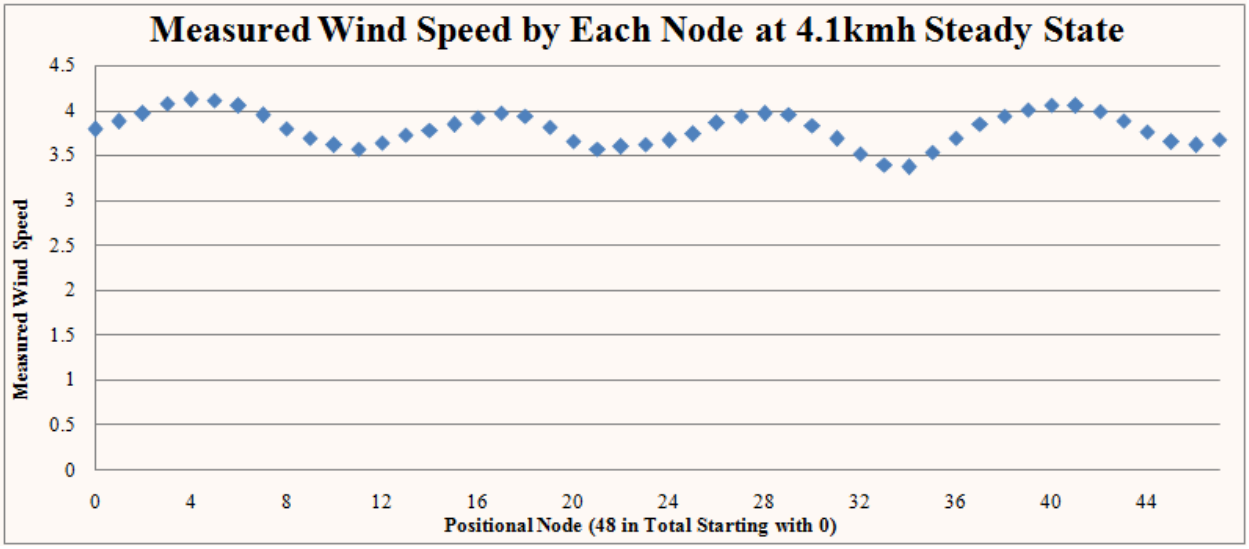


Figure 3.4.b-z: *Plot of Anemometer's Measured Wind Speed at Minimum Speeds*

Section 3.4.c: Testing Maximum Wind Speed

The calculated maximum wind speed before the anemometer would topple over could not be verified using Purdue's Boeing wind tunnel. The only collected data was that the toppling wind speed was over 50 km/hr.

Section 3.4.d: Initial Calibrating the Nodes for Wind Speed

This calibration was performed before finding the wind speed equation that was accepted. The first observation of the data was that all of the odd nodes were reading low compared to the even nodes. The second observation using the 5 km/hr data was that some of the nodes were scattered from their expected symmetric positions, as seen in Figure 3.4.d-t. An additional reason to perform further calibrating was due to the CNC mill's results having possibly up to ±.5% of error, which may have contributed to the scattering a small amount, as the change in RPM between any two nodes was often under 2%.

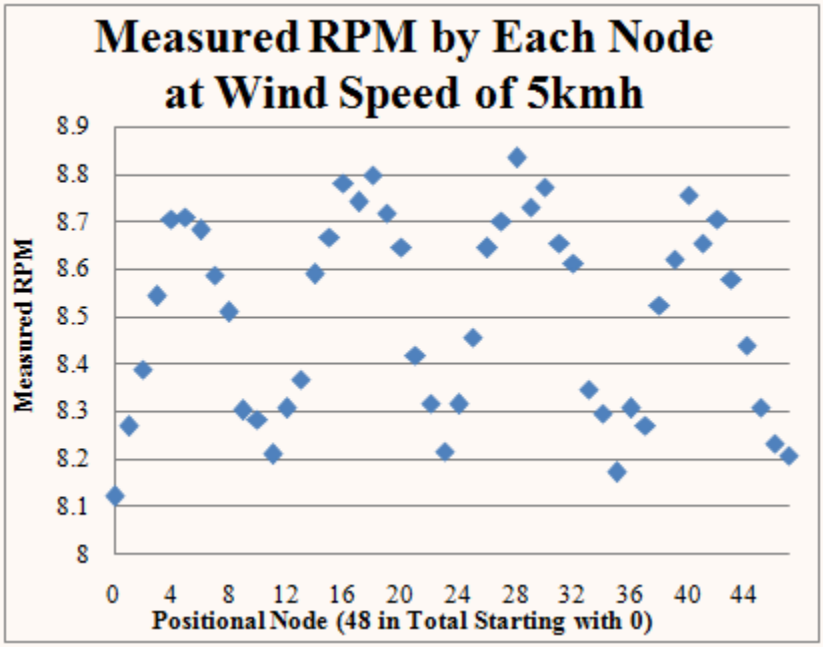


Figure 3.4.d-t: *Plot of Anemometer's Measured Wind Speed at Minimum Speeds*

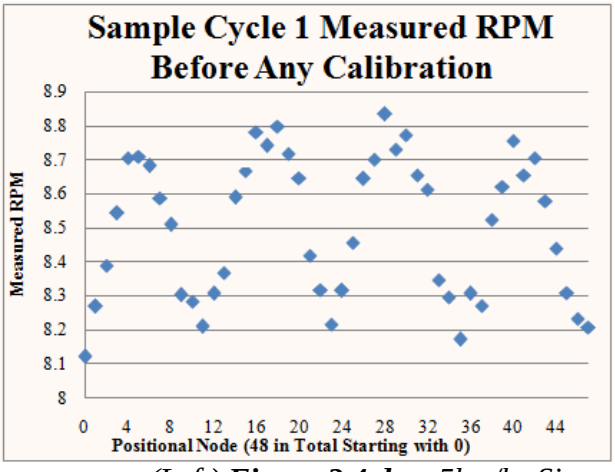
The calibration process began by manually adding or subtracting values from each node in order to make the plot for that cycle appear as a sinusoid, using symmetry of nodes elsewhere on the plot where there was consistency. Then the newly calibrated values would be transferred to a new plot of a different cycle sample. The calibration would continue to be refined, as after each plot less and less nodes appeared inconsistent. After several plots and compromising changes between the different plots, each plot had only one or two inconsistent RPM reports from the patterns common among unadjusted nodes. These new calibrated values were multiplied by a ratio to set their sum to equal 1. The new values were then used in place of the CNC mill's results. An example excerpt of the process is shown in Table 3.4.d-x.

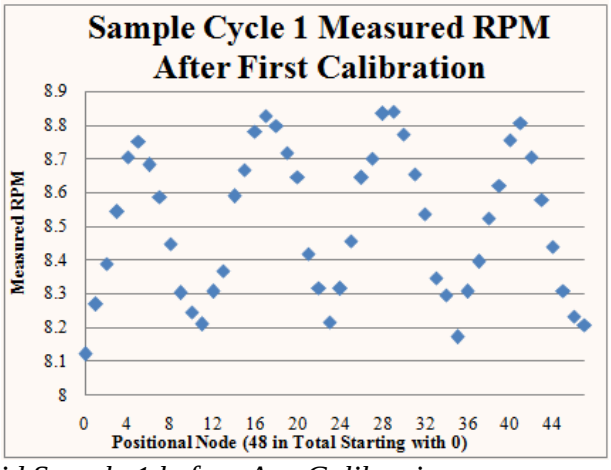
Table 3.4.d-x: *Process of Calibrating Nodes Based on Sinusoid*

Example Table of Calibrating Nodes Based on Sinusoid					
Current Node	Old Node Constant	Measured RPM from	Addition or Subtraction	New Node Constant	New RPM
0	0.02135	8.35		0.02135	8.35
1	0.020177	8.42		0.020177	8.42
2	0.021318	8.61		0.021318	8.61
3	0.020114	8.72		0.020114	8.72
4	0.021305	8.82		0.021305	8.82
5	0.020277	8.98	0.00025	0.020527	9.09
6	0.021431	8.99	0.0002	0.021631	9.07
7	0.020189	8.82		0.020189	8.82
8	0.021255	8.60	-0.00025	0.021005	8.50
(…)					
41	0.019998	8.97	0.00035	0.020348	9.12
42	0.021692	8.88		0.021692	8.88
43	0.02011	8.71	0.00008	0.02019	8.75
44	0.021515	8.57		0.021515	8.57
45	0.020098	8.39		0.020098	8.39
46	0.021339	8.31		0.021339	8.31
47	0.01991	8.33		0.01991	8.33

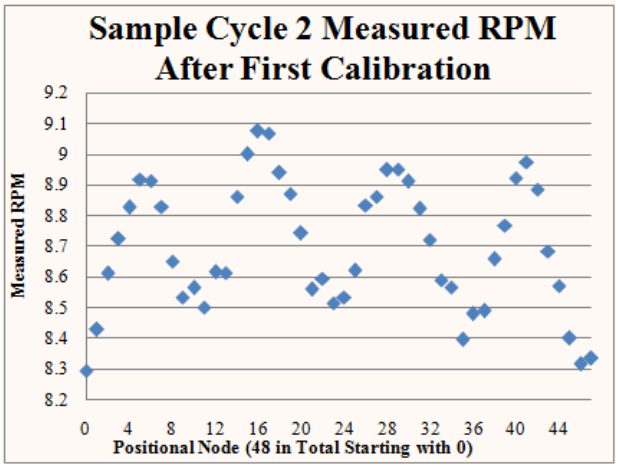
**These values are carried over from previous cycles

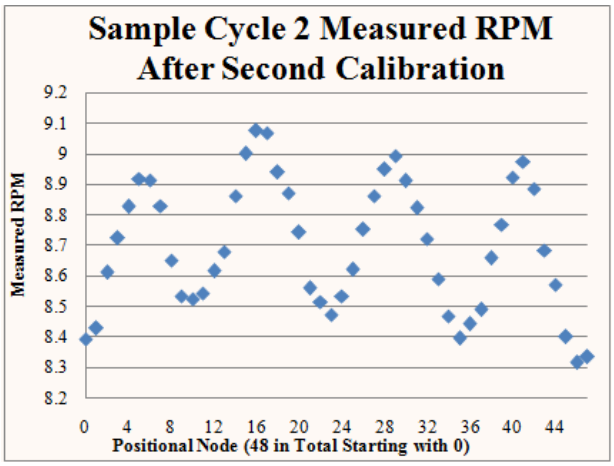
Before and after plots are shown in Figures 3.4.d-u, 3.4.d-v, 3.4.d-w, 3.4.d-x, 3.4.d-y, and 3.4.d-z. Figure 3.4.d-v is after the first calibration process. Figure 3.4.d-x is after the second calibration process.



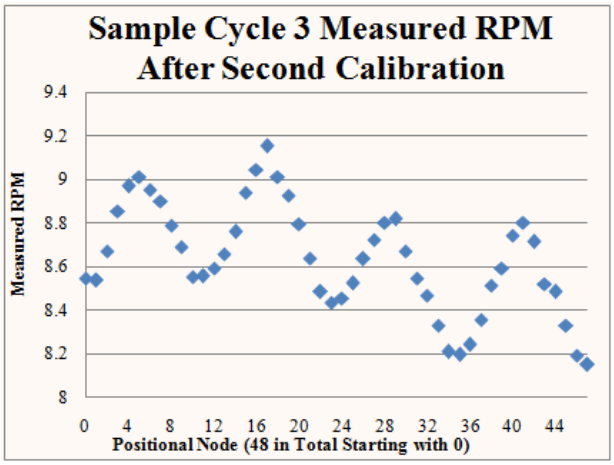


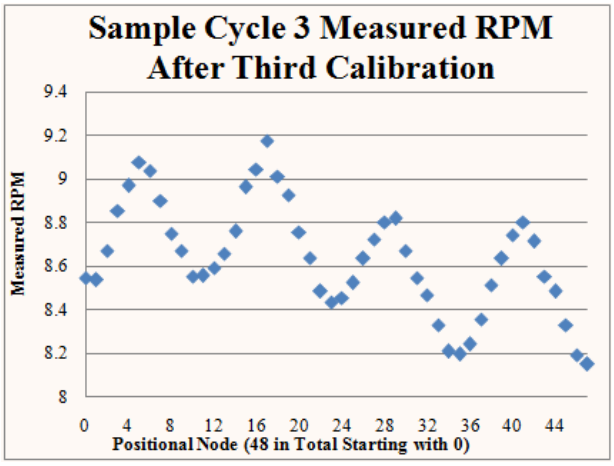
(Left) ***Figure 3.4.d-u:*** *5km/hr Sinusoid Sample 1 before Any Calibration*
(Right) ***Figure 3.4.d-v:*** *5km/hr Sinusoid Sample 1 after First Calibration*





(Left) ***Figure 3.4.d-w:*** *5km/hr Sinusoid Sample 2 after First Calibration*
(Right) ***Figure 3.4.d-x:*** *5km/hr Sinusoid Sample 2 after Second Calibration*





(Left) ***Figure 3.4.d-y:*** *5km/hr Sinusoid Sample 3 after Second Calibration*
(Right) ***Figure 3.4.d-z:*** *5km/hr Sinusoid Sample 3 after Third Calibration*
Note: *Very few changes needed by the third calibration compared to the first two.*
Additional Note: *Nodes 0 & 47 may not match well due to the wind speed varying over time.*

Section 3.4.e: Calibrating the Nodes to Eliminate Sinusoid Nature

The last calibration for the anemometer was calibrating each node based on the minimum measured RPM in a cycle, since the wind speed was to be calculated off of the minimum measured RPM. This would result in the sinusoid nature of the plot being mostly eliminated, and present the measured wind speed closer to being a constant when in steady state wind flow. This was done by finding the minimum reported RPM in a cycle and creating a ratio between each node and that minimum. Then an average of the ratios over several cycles was used to produce a new constant for that node. This was done for each wind speed that was laminar flow, and the average was taken of each wind speed's nodes' constants. Data for the wind speed of 25 km/hr was left out, since it was recorded in high speed mode which only reports on even nodes. Data for the wind speed of 5 km/hr was also left out due to the sinusoid nature beginning to start a

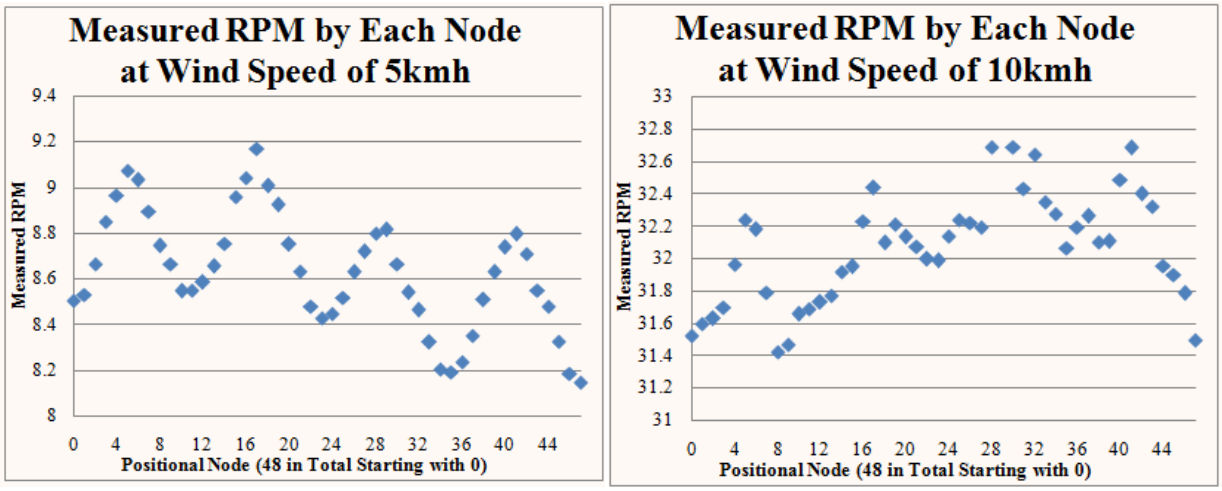
new trend starting at 10kmh. This is shown below in Figure 3.4.e-x and Figure 3.4.e-y. The process is shown in Table 3.4.e-x. The results for each wind speed and each node constant are shown in Table 3.4.e-y.

Table 3.4.e-x: *Process of Calculating S.S. Node Constants at 10 km/hr*

Calibrating New Node Constant from Previous Constant & Minimum RPM							
Sample for One Cycle				Inverse	Average	(10kmh)	New RPM
Current Node	Node Constant	Measured RPM	Cycle Minimum	Ratio of Current	Ratio Across All Cycles	New Node Constant	for Selected Cycle
0	0.02149	32.09	31.79	0.9908	0.98457	0.02115	31.59
1	0.02016	32.07	31.79	0.99142	0.98753	0.01991	31.67
2	0.0213	32.07	31.79	0.99129	0.98540	0.02099	31.60
3	0.0201	32.16	31.79	0.98862	0.98388	0.01978	31.64
4	0.02129	32.32	31.79	0.98353	0.97735	0.02081	31.59
5	0.02051	32.71	31.79	0.97204	0.96757	0.01985	31.64
6	0.02162	32.53	31.79	0.97733	0.96833	0.02093	31.50
7	0.02018	32.07	31.79	0.99125	0.98399	0.01985	31.56
8	0.02099	31.79	31.79	1	0.99269	0.02084	31.56
9	0.02009	31.85	31.79	0.99808	0.99227	0.01993	31.61
(…)					(…)		(…)
45	0.02008	32.17	31.79	0.98815	0.98845	0.01985	31.80
46	0.02132	31.97	31.79	0.99429	0.99322	0.02118	31.76
47	0.0199	31.80	31.79	0.9996	0.99936	0.01988	31.78

Table 3.4.e-y: *Steady-State Wind Speed Node Constants*

Averaging Node Constant between Varying Wind Speeds						Standard
Node	10 kmh	13.5 kmh	15 kmh	20 kmh	**Average**	Deviation
0	0.021153	0.021142	0.021129	0.021086	**0.021128**	0.0000294
1	0.019912	0.019933	0.019912	0.019898	**0.019914**	0.0000146
2	0.020993	0.021011	0.021003	0.020998	**0.021001**	0.0000080
3	0.019776	0.019795	0.019833	0.019815	**0.019805**	0.0000244
4	0.020808	0.020877	0.020882	0.020851	**0.020855**	0.0000339
(…)						
45	0.019852	0.019882	0.019879	0.019781	**0.019849**	0.0000472
46	0.021180	0.021191	0.021214	0.021137	**0.021181**	0.0000324
47	0.019884	0.019855	0.019845	0.019778	**0.019840**	0.0000447



(Left) ***Figure 3.4.e-x:*** *5km/hr Sinusoid Example (Zoomed In)*
(Right) ***Figure 3.4.e-y:*** *10km/hr & Higher Sinusoid Example (Zoomed In)*

Then the equation for finding wind speed needed to be reevaluated using the new constants. The results are shown in Figure 3.4.e-z. Overall, the accuracy slightly improved for most of the laminar wind speeds. It should be noted that these new constants do not accurately find the RPM of the device, but rather accurately define the wind speed relationship to the RPM of the device.

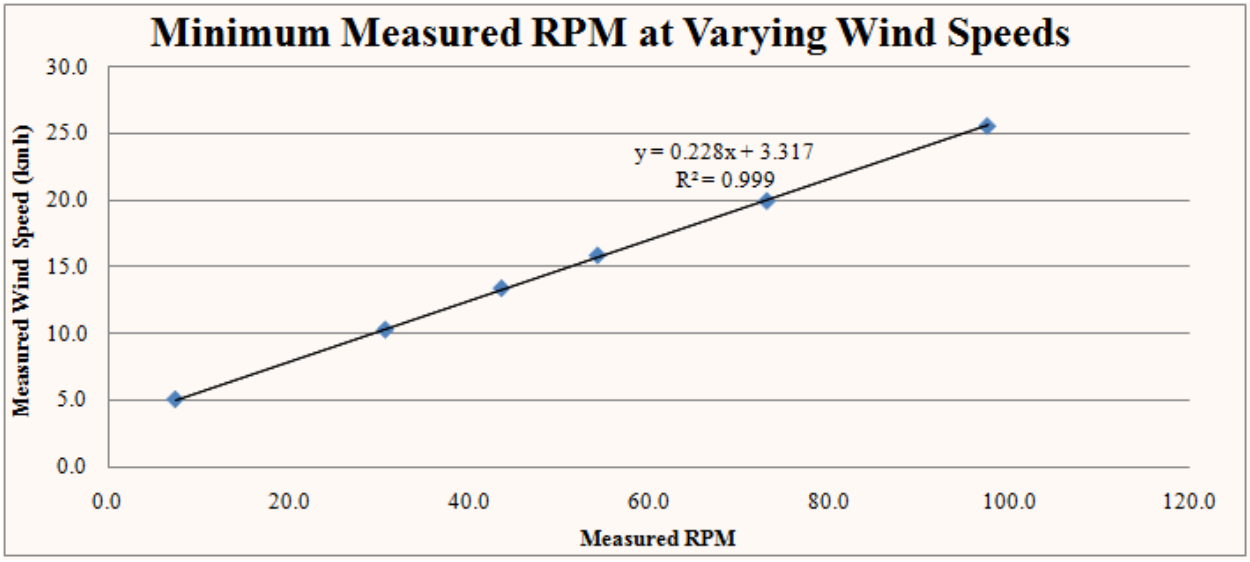


Figure 3.4.e-z: *Calibrated Measurement of RPM to Eliminate Sinusoid Nature*

Section 3.5: Comparing Results to Theoretical

The theoretical equation for comparing RPM to wind speed found during the design process of the anemometer after spinning had begun was:

$$RPM = (9.55)\omega = 9.55\left(59.27v - \sqrt{3299.9779\,v^2 + 62.5032}\right)$$

(Equation 3.5-x)

The final equation determined through testing for accurately detecting wind speed in kmh within 2% error in the laminar region was:

$$v = (.228)RPM + 3.317$$

(Equation 3.5-y)

Which when set equal to RPM and m/s is (when the device is spinning):

$$RPM = (15.79)v - 14.54$$

(Equation 3.5-y)

Table 3.5-x: *Comparing Theoretical Equation to Fitted Equation*

Comparing Theoretical Equation & Fitted Equation			
Measured Wind	Measured RPM	Theoretical RPM	Error
2.85	30.46	47.83	57.0%
3.41	39.30	57.89	47.3%
4.41	55.09	75.66	37.3%
5.62	74.20	96.99	30.7%
7.09	97.41	122.80	26.1%
8.10	113.36	140.49	23.9%
9.16	130.10	159.03	22.2%
10.75	155.20	186.82	20.4%
14.00	206.52	243.56	17.9%

The theoretical equation's error mostly came from the bearing's friction being different from its technical data sheet, as discovered by the minimum wind speed required to begin spinning. Other major sources of error would have come from the cups having large lips and the weight of the spinning portion. The cups having large lips surely played a large role, as they would drastically change the coefficient of drag for the cups.

Section 4: Evaluation, Cost Analysis, Recommendations, & Conclusion

Section 4.1: Evaluation

Section 4.1.a: Specified Requirements

The anemometer was able to meet the specified requirements:

1. The anemometer was able to measure wind speed from 5 $\frac{km}{hr}$ to 25 $\frac{km}{hr}$ with an accuracy of ±2%, compared to the goal of 5 $\frac{km}{hr}$ to 20 $\frac{km}{hr}$ with ±5% accuracy of the steady state wind speed.
2. The anemometer begun spinning and reading results around 4.75 $\frac{km}{hr}$, below the goal of 5 $\frac{km}{hr}$, and did not show any sign of failure up to the wind speed of 50 $\frac{km}{hr}$.

Section 4.1.b: Given Parameters

The anemometer was able to meet the specified given parameters:

1. The anemometer was a four cup anemometer, as was given as a parameter.
2. The anemometer was capable of displaying the wind speed by three methods.
3. The anemometer was calibrated against a accurately known source of wind speed.

Section 4.1.c: Limitations and Constraints

The anemometer was able to meet most of the specified limitations and constraints:

1. The anemometer went over the carrying weight limitation of 7 lbs by weighing around 11 lbs.
2. The size of the anemometer fitted in approximately a 13" by 16" by 14" rectangular cube, below the limitations of 18" by 18" by 18".
3. The required components of the anemometer were able to fit into the $300 budget contributed by the IPFW Department of Civil & Mechanical Engineering with a total required cost of $266.34.

Section 4.1.d: Additional Considerations

The anemometer was able to include the following optional considerations:

1. The anemometer had no sharp edges, had low forces if stopped immediately, and showed no signs of failure which could cause injury to a nearby viewer of the device.
2. The anemometer could be assembled and disassembled for use within 30 seconds.
3. The anemometer was weather resistant and is made of durable materials to last many years.
4. The anemometer was able to operate automatically and display results in a simple fashion that students without fluids knowledge can understand.
5. The anemometer was able to operate in an outdoors environment and collect data unattended.
6. The anemometer was easily portable for demonstration purposes.

Section 4.2: Cost Analysis

The anemometer had several extra features for the purpose of being a better display piece. As such, the components and their costs are categorized and their necessity noted in Table 4.2-x on the next page. The total cost of necessary components and assembly was $266.34. The total cost of the anemometer was $685.45. All of the assembly and machining was included as additional cost in Table 4.2-x, but most labor was donated from the team. Software development was left off of the bill of materials since it could be transferred to future anemometers, and it was done by the team. The calibrating and testing processes were left off of the bill of materials, as it depends on how accurate of a calibration the user desires.

Table 4.2-x: *Bill of Materials*

Anemometer Cost Analysis								
Category	**Required for Base Anemometer?**						**Required Cost**	**Total Cost**
Item	Number of Units	Supplier	Price Per Unit	Labor Rate	Labor Time	Required?	Required Cost	Total Cost
Anemometer Spinning Portion	**Yes**						**$198.86**	**$198.86**
Fat Daddio's 4" x 2" Deep Aluminum Hemisphere Pan	4	Fat Daddio; Amazon.com	$7.88	$25.00	0.25	Yes	$37.78	$37.78
Aluminum Tubing, .375im ID, 1/2in OD, 2ft	1	Metals Depot	$8.11	$25.00	0.1	Yes	$10.61	$10.61
Central Hub, Nylon	1	IPFW	$0.00	$25.00	0.25	Yes	$6.25	$6.25
Measurement Disc, Stainless Steel	1	ONXX Tool Inc	$0.00	$50.00	1	Yes	$50.00	$50.00
Banded Ball Thrust Bearing,	1	Grainger	$59.68	$0.00	0	Yes	$59.68	$59.68
JB Weld SteelStik	1	Hardware Store	$4.27	$0.00	0	Yes	$4.27	$4.27
JB Weld Kwik	1	Hardware Store	$5.27	$0.00	0	Yes	$5.27	$5.27
Assembly	1	IPFW	$0.00	$25.00	1	Yes	$25.00	$25.00
Electronics	**Yes**						**$67.48**	**$136.25**
Arduino Uno R3	1	Adafruit	$13.50	$25.00	0	Yes	$13.50	$13.50
Photo Interrupter with Breakout Board	2	Sparkfun Electronics	$5.46	$25.00	0.25	Yes	$17.16	$17.16
RGB 16x2 Character Display	1	Adafruit	$15.21	$25.00	0.5	Yes	$27.71	$27.71
20 M-M Cables	1	Adafruit	$4.31	$25.00	0	Yes	$4.31	$4.31
Adafruit Data Logger Shield	1	Adafruit	$21.75	$25.00	1	No	$0.00	$46.75
4GB SD Card	1	Adafruit	$8.67	$25.00	0	No	$0.00	$8.67
USB Power Supply	1	Adafruit	$5.40	$25.00	0	No	$0.00	$5.40
6x AA Battery Holder	1	Adafruit	$5.45	$25.00	0.1	No	$0.00	$7.95
2 Ton Epoxy	1	Hardware Store	$4.80	$0.00	0	Yes	$4.80	$4.80
Electronics Storage	**No**						**$0.00**	**$29.88**
Pelican 1050 Micro Dry Case	1	Amazon.com	$20.25	$25.00	0.2	No	$0.00	$25.25
Photograph Case	1	Crafts Store	$2.13	$25.00	0.1	No	$0.00	$4.63
Structure	**No**						**$0.00**	**$320.46**
Neff Engineering 80-20 Purchase	1	Neff Engineering	$289.61	$25.00	0.25	No	$0.00	$295.86
Aluminum 90° Angle 3ft 1"x1" 1/16" Thickness	1	Hardware Store	$6.13	$25.00	0.25	No	$0.00	$12.38
1/4-20x3/4 Stainless Steel Button Head Cap Screws	8	Hardware Store	$0.59	$25.00	0	No	$0.00	$4.72
Square Nut 1/4-20 COA	4	Hardware Store	$0.31	$25.00	0.25	No	$0.00	$7.50
			Totals				**Required**	**Spent**
							$266.34	**$685.45**

Section 4.3: Recommendations

The following are the recommendations if one is to construct the same anemometer, maintain this anemometer, and use this anemometer in demonstrations:

1. When the components are obtained for constructing a new anemometer, it is important to verify that all the components have the same dimensions used as in the design.
2. The size of the case for the anemometer should be increased, especially height, if using the same components to create a new anemometer.
3. The case should be modified such that the anemometer cannot spin while in transportation, and moves to this fixed position as the case descends upon the anemometer.
4. The same lubricant should be used when lubing if it is desired to maintain the anemometer's accuracy.
5. If being used to measure higher wind speeds accurately, further calibration should be done over the transient and turbulent regions.
6. The anemometer should not be left out in harsh weather, weather with high air moisture, or a location with high air moisture. If it is desired to use the anemometer as a weather station, the electronics should be more thoroughly protected using epoxy.
7. The anemometer should not be left unattended with guest students, as playing with the anemometer with sudden acceleration and stops could weaken the strength of the bonds of the adhesives used.
8. The anemometer does spin at high speeds, so young viewers should be kept a small distance away from the anemometer.

Section 4.4: Conclusion

The anemometer met all of the functional requirements, and most of the ergonomic requirements. The only limitation not met was keeping the weight of the anemometer below 7lb to keep it ergonomically friendly. The anemometer was able to measure wind speeds accurately within 2% over the laminar region from 5 $\frac{km}{hr}$ to 25 $\frac{km}{hr}$, and within 10% accuracy up to 50 $\frac{km}{hr}$, only needing further calibration to handle the transient and turbulent regions. The cost for the functional components of the anemometer to measure wind speed was $266.34 with labor. The software of the anemometer allows it to self detect most errors and operate automatically. The device was found to be durable, and should even be safe to use as an outdoor weather station. The anemometer was made to be portable using 80-20 aluminum extrusion that formed a base and a lid that attached to that base.

The completed anemometer satisfied all the following criteria:

- Is a classic four cup anemometer.
- Measures wind speeds accurately as low as 5 $\frac{km}{hr}$
- Measures wind speeds accurately of at least 20 $\frac{km}{hr}$
- Displays the wind speeds via the LCD display or on a computer screen
- Allows recording of wind speeds three ways: manually off of the LCD display, through a PC, or onto a SD card for later viewing
- Fits in a 18" by 18" by 18" cube
- Fits the $300 budget for required components
- Can be packed into its case in less than a minute
- Can operate outdoors

The completed anemometer failed one criteria:

- Weighs less than 7 lb for ergonomic friendliness (weighed around 11 lb)

Appendix A

Parts Data Sheets

Appendix A.1: Photo Interrupter Data Sheet

SHARP GP1A57HRJ00F

GP1A57HRJ00F

Gap : 10mm, Slit : 1.8mm
***OPIC Output**
Case package Transmissive Photointerrupter

■ Description

GP1A57HRJ00F is a standard, OPIC output, transmissive photointerrupter with opposing emitter and detector in a case, providing non-contact sensing. For this family of devices, the emitter and detector are inserted in a case, resulting in a through-hole design.

This device has a wide gap.

■ Features

1. Transmissive with OPIC output
2. Highlights :
   - Vertical Slit for alternate motion detection
   - Output Low Level at intercepting optical path
   - Wide gap width (10mm)
   - Positioning Pin to prevent misalignment
3. Key Parameters :
   - Gap Width : 10mm
   - Slit Width (detector side) : 1.8mm
   - Package : 18.6×15.2×5mm
4. Lead free and RoHS directive compliant

■ Agency approvals/Compliance

1. Compliant with RoHS directive

■ Applications

1. General purpose detection of object presence or motion.
2. Example : Printer, FAX, Optical storage unit

* "OPIC"(Optical IC) is a trademark of the SHARP Corporation. An OPIC consists of a light-detecting element and a signal-processing

Sheet No.: D3-A03901EN
Date Oct. 3, 2005

Figure A.1: *Excerpt of Photo Interrupter Data Sheet*
Retrieved from https://www.sparkfun.com/datasheets/Components/GP1A57HRJ00F.pdf

Appendix A.2: Bearing Data Sheet

Technical Specs

Item	Thrust Bearing	Dynamic Load Capacity (Lb.)	7700
Type	Banded Ball	Static Load Capacity (Lb.)	18300
Direction	Axial	Max. RPM	2800
Bore Dia. (In.)	1.500	Cage Material	Steel
Outside Dia. (In.)	2.594	Temp. Range (F)	-20 - + 250
Height (In.)	0.625	Lubrication	OIL

***Figure A.2**: Excerpt of Bearing Data Sheet*
Retrieved from: http://www.grainger.com/product/INA-Banded-Ball-Thrust-Bearing-4ZZT1?s_pp=false&picUrl=//static.grainger.com/rp/s/is/image/Grainger/4ZZP5_AS01?$smthumb$

Appendix B

Arduino Code

Appendix B: Arduino Code

Note: Some code was commented out due to minor issues getting the libraries working

```
#include <Time.h>
#include <math.h>
#include <SPI.h>
#include <SD.h>              //logger
#include <Wire.h>            //logger //display  //Pressure Sensor
#include <RTClib.h>
#include <DS1307RTC.h>
#include <LiquidCrystal.h>       //display

//VARIABLES
// Display Pins
#define DisplayRed 6
#define DisplayBlue 3
#define DisplayGreen 5


// Data Pins
#define SensorPin1 A0
#define SensorPin2 A1
#define DataLoggerPin 10


// Calculation Vars
const float SensorSpacing[52] =
{0.021350,0.020177,0.021318,0.020114,0.021305,0.020277,0.021431,0.020189,0.021255,0.020
152,0.021418,0.020161,0.021352,0.020161,0.021302,0.020138,0.021389,0.020073,0.021490,0.0
20060,0.021666,0.019985,0.021793,0.020073,0.021929,0.020010,0.021954,0.019860,0.022043,
0.019974,0.022068,0.019836,0.022167,0.019784,0.022031,0.020022,0.021954,0.019785,0.0216
65,0.019947,0.021682,0.019998,0.021692,0.020110,0.021515,0.020098,0.021339,0.019910,0.02
1350,.0001,.0001,.0001};
const float SensorSinusoidRPM[52] =
{0.0214850,0.0201633,0.0213035,0.0201003,0.0212905,0.0205130,0.0216163,0.0201753,0.0209
907,0.0200883,0.0212035,0.0202472,0.0213375,0.0202972,0.0212875,0.0201743,0.0213744,0.
0203091,0.0214753,0.0200463,0.0215513,0.0199714,0.0215783,0.0199594,0.0219141,0.019996
4,0.0217392,0.0198465,0.0220280,0.0203101,0.0220529,0.0198225,0.0219521,0.0197705,0.021
7661,0.0200084,0.0218391,0.0200713,0.0216503,0.0200333,0.0216672,0.0203341,0.0216772,0.
0201763,0.0215003,0.0200843,0.0213245,0.0198965,  0.0214850, 0.0201633, 0, 0};
const float SensorWindRPM[52] =
{0.021127652,0.019913579,0.021001202,0.019804569,0.020854763,0.019926609,0.020989628,
0.019920735,0.020886257,0.019980971,0.020977259,0.021003157,0.019986811,0.020053997,0
.020928177,0.019826177,0.020888863,0.019722465,0.021038753,0.019645071,0.021147750,0.
019627462,0.021184730,0.019573102,0.021349881,0.019493662,0.021202228,0.019350328,0.0
```

```
21265617,0.019449552,0.021317038,0.019282391,0.021348559,0.019308777,0.021307626,0.01
9664973,0.021319875,0.019650010,0.021229753,0.019674004,0.021097004,0.019713108,0.021
062034,0.019818492,0.021136259,0.019848666,0.021180775,0.019840348,  0.021127652,
0.019913579, 0, 0};
long Peak = 0;
const float RatioRadsToSpeed = .0637;
float TrueRPM = 0;
float SinusoidRPM = 0;
float WindRPM = 0;
float WindSpeed = 0;
float Constant1 = 0.227;
float Constant2 = 3.297;


// Timing Vars
unsigned long TimeStamp = 0;
unsigned long PreviousTimeStamp = 0;
unsigned long CycleTime = 0;


// Logic Vars
byte KnowsPosition = 0;
int CurrentNode = 0;
boolean LastReading = 1;
boolean WritePC = 1;
byte ReverseCount = 0;
byte ConfirmCount = 0;
byte Mode = 1;

// Calibration Vars
boolean NodeMoved = 0;
byte CalibrationNode = 0;
unsigned long CalibrationStart = 0;
unsigned long CalibrationChange = 0;
unsigned long CalibrationVariance = 0;
unsigned long CalibrationMax = 0;
unsigned long CalibrationMin = 0;
unsigned long CalibrationTolerance = 10000;
unsigned long CalibrationSum = 0;
boolean CalibrationBreak = 0;
unsigned long CalibrationAverage[52] =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0};


// Display Vars
```

```
boolean WriteDisplay = 1;
byte LastDisplayNode = 0;


// Logging Vars
boolean WriteLogger = 1;
byte DateError = 0;




//SETUP
LiquidCrystal lcd(A2, 9, 8, 7, 4, 2);      //Display Startup

RTC_DS1307 Clock;                          //Logger Startup
File LoggerTextFile;

void error(char *str)       //99 Questionable
{
  Serial.print(F("error: "));
  Serial.println(str);
  LCDColor(250,250,250);
}                           //99 Questionable                                   //END LOGGER

void setup()
{
  //99RTC.begin(DateTime(F(__DATE__), F(__TIME__)));
  //99DateTime now = RTC.now();
  delay(500);
  Serial.begin(9600);
  pinMode(DisplayRed, OUTPUT);
  pinMode(DisplayBlue, OUTPUT);
  pinMode(DisplayGreen, OUTPUT);
  pinMode(DataLoggerPin, OUTPUT);
  digitalWrite(DataLoggerPin, HIGH);
  lcd.begin(16, 2);                  //display
  lcd.clear();

  Bootup();
}

void Bootup()
{
  lcd.setCursor(0,0);
  lcd.print(F("Booting Up"));
```

```
 LCDColor(25,0,0);

 Serial.flush();
 delay(2000);

 if (!SD.begin(DataLoggerPin))
 {
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(F("SD Card Missing"));
  lcd.setCursor(0,1);
  lcd.print(F("10sec to insert"));
  BlinkLCD(0,255,240,0,255,220);
  delay(4000);
  if (!SD.begin(DataLoggerPin))
  {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print(F("SD Data Logging "));
    lcd.setCursor(0,1);
    lcd.print(F("    Disabled    "));
    WriteLogger = 0;
    LCDColor(0,255,255);
  }
 }
 delay(4000);

 if(WriteLogger)
 {
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(F("Writing File    "));
  lcd.setCursor(0,1);
  lcd.print(F(" To SD Card     "));
  LCDColor(25,0,0);
  delay(1500);/*99
  char filename[] = "WindSpeed000.txt";
  for (uint8_t i = 0; i < 100; i++)
  {
    filename[9] = i/10 + '0';
    filename[10] = i%10 + '0';
    filename[11] = i%10 + '0';
    if (! SD.exists(filename))
    {
      LoggerTextFile = SD.open(filename, FILE_WRITE);
      break; // leave the loop!
```

```
    }
  }99*/

  LoggerTextFile = SD.open("CalbTest.txt", FILE_WRITE);
  if (!LoggerTextFile)
  {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print(F("File Error:     "));
    lcd.setCursor(0,1);
    lcd.print(F("Card may be full"));
    BlinkLCD(0,255,240,0,255,220);
    delay(2750);
    lcd.setCursor(0,0);
    lcd.print(F("SD Data Logging "));
    lcd.setCursor(0,1);
    lcd.print(F("    Disabled    "));
    WriteLogger = 0;
    LCDColor(0,255,255);
    delay(3000);
 }
 else
 {
  lcd.setCursor(0,0);
  lcd.print(F("File Created    "));
  lcd.setCursor(0,1);
  lcd.print(LoggerTextFile);
  LCDColor(175,230,0);
  delay(3000);
  Wire.begin();
  if (!Clock.begin())
  {
    DateError = 2;
    lcd.setCursor(0,0);
    lcd.print(F("Date Error       "));
    lcd.setCursor(0,1);
    lcd.print(F("Replace Battery  "));
    BlinkLCD(0,255,240,0,255,220);
    delay(2750);
    LoggerTextFile.print(F("Error")); // seconds since 2000
    LoggerTextFile.print("\t");
    LoggerTextFile.print(F("Error"));
    LoggerTextFile.print("\t");
    LoggerTextFile.print(F("Error"));
  }
  else
```

```
  {/*99
    DateError = 1;
    LoggerTextFile.print(now.get()); // seconds since 2000
    LoggerTextFile.print(F("\t"));
    LoggerTextFile.print(now.month(), DEC);
    LoggerTextFile.print(F("/"));
    LoggerTextFile.print(now.day(), DEC);
    LoggerTextFile.print(F("/"));
    LoggerTextFile.print(now.year(), DEC);
    LoggerTextFile.print(F("\t"));
    LoggerTextFile.print(now.hour(), DEC);
    LoggerTextFile.print(F(": "));
    LoggerTextFile.print(now.minute(), DEC);
    LoggerTextFile.print(F(": "));
    LoggerTextFile.println(now.second(), DEC);
    LoggerTextFile.println("Time (Milliseconds)");99*/
  }
}/*99
if (LoggerTextFile.writeError() || !LoggerTextFile.sync())
{
    WriteLogger = 0;
    lcd.setCursor(0,0);
    lcd.print(F("Logger Error       "));
    lcd.setCursor(0,1);
    lcd.print(F("Refer Manual       "));
    BlinkLCD(1023,523,523,1023,1023,1023);
    delay(2750);
}99*/
}

lcd.setCursor(0,0);
lcd.print(F("Connecting to   "));
lcd.setCursor(0,1);
lcd.print(F("PC & Python.exe "));

LCDColor(25,0,0);
delay(200);
Serial.println(F("s"));
delay(1500);
if (!Serial.available())
{
  lcd.setCursor(0,0);
  lcd.print(F("Failed toConnect"));
  lcd.setCursor(0,1);
  lcd.print(F("Python or cable "));
  BlinkLCD(0,255,240,0,255,220);
```

```
  delay(8750);
  Serial.println(F("s"));
  delay(500);
  if (!Serial.available())
  {
    lcd.setCursor(0,0);
    lcd.print(F("PC Disconnected "));
    lcd.setCursor(0,1);
    lcd.print(F("or Python fail  "));
    WritePC = 0;
    LCDColor(0,255,255);
    delay(3000);
  }
 }
 else
 {
  lcd.setCursor(0,0);
  lcd.print(F("PC Connected    "));
  lcd.setCursor(0,1);
  lcd.print(F("Python Running  "));
  LCDColor(175,230,0);
  delay(3000);
  if (DateError == 1)
  {/*99
    Serial.print(now.get()); // seconds since 2000
    Serial.print(F("\t"));
    Serial.print(now.month(), DEC);
    Serial.print("/"));
    Serial.print(now.day(), DEC);
    Serial.print(F("/"));
    Serial.print(now.year(), DEC);
    Serial.print(F("\t"));
    Serial.print(now.hour(), DEC);
    Serial.print(F(": "));
    Serial.print(now.minute(), DEC);
    Serial.print(F(": "));
    Serial.println(now.second(), DEC);99*/
  }
  else if (!WriteLogger)
  {
    Wire.begin();
  if (!Clock.begin())
  {
    DateError = 2;
    lcd.setCursor(0,0);
    lcd.print(F("Date Error       "));
```

```
    lcd.setCursor(0,1);
    lcd.print(F("Replace Battery  "));
    BlinkLCD(0,255,240,0,255,220);
    delay(2750);
    Serial.println(F("Time"));
   }
   else
   {/*99
    Serial.print(now.get()); // seconds since 2000
    Serial.print(F("\t"));
    Serial.print(now.month(), DEC);
    Serial.print(F("/"));
    Serial.print(now.day(), DEC);
    Serial.print(F("/"));
    Serial.print(now.year(), DEC);
    Serial.print(F("\t"));
    Serial.print(now.hour(), DEC);
    Serial.print(F(": "));
    Serial.print(now.minute(), DEC);
    Serial.print(F(": "));
    Serial.println(now.second(), DEC);99*/
  }
  }
  }
}

//SENSOR READINGS/Calculations/Logging
void loop()
{

if (Mode == 1)
{
  ScanNode();

  if (KnowsPosition == 1)
  {
   TrueRPM = SensorSpacing[CurrentNode]/CycleTime*60000000;

   WindRPM = SensorWindRPM[CurrentNode]/CycleTime*60000000;

   WindSpeed = Constant1*WindRPM + Constant2;

   //Direction Stuff 99

   //Accel Stuff 99
```

```
if(WindRPM > 200)
{
  Mode = 6;
  if (WriteDisplay)
  {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print(F("Entering"));
    lcd.setCursor(0,1);
    lcd.print(F("Ultra Speed Mode"));
  }
  if (WritePC)
  {
    Serial.println(F("I0"));
  }
  if (WriteLogger)
  {
    LoggerTextFile.println(F("I0"));
  }
}
else if ((WindRPM > 100) && (CurrentNode%2 == 0))
{
  Mode = 5;
  if (WriteDisplay)
  {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print(F("Entering"));
    lcd.setCursor(0,1);
    lcd.print(F("High Speed Mode"));
  }
  if (WritePC)
  {
    Serial.print(F("H"));
    Serial.println(CurrentNode);
  }
  if (WriteLogger)
  {
    LoggerTextFile.print(F("H"));
    LoggerTextFile.println(CurrentNode);
  }
}
else
{
  if ((WriteDisplay) && (LastDisplayNode > (WindRPM/1.5)))
  {
```

```
      lcd.clear();
      lcd.setCursor(8,0);
      lcd.print(F("kmh"));
      lcd.setCursor(0,0);
      lcd.print(WindSpeed);
      lcd.setCursor(0,1);
      lcd.print(F("Node:"));
      lcd.setCursor(6,1);
      lcd.print(CurrentNode);

      LastDisplayNode = 0;

      if(WindRPM < 1)
      {
        LCDColor(235,0,85);
      }
      else
      {
        LCDColor(255,200,0);
      }
    }
    else
    {
      LastDisplayNode++;
    }

    if (WritePC)
    {
      Serial.println(CycleTime);
    }

    if (WriteLogger)
    {
      LoggerTextFile.println(CycleTime);
    }
   }
  }
}

else if (Mode == 2)
{
  lcd.setCursor(0,0);
  lcd.print(F("Calibration      "));
  lcd.setCursor(0,1);
  lcd.print(F("Running          "));
  LCDColor(255,150,0);
```

```
CalibrationNode = 0;
CalibrationStart = 0;
CalibrationChange = 0;
CalibrationVariance = 0;
CalibrationSum = 0;
CalibrationMax = 0;
CalibrationMin = 2000000;
while(CalibrationBreak)
{
  ScanNode();
}
memset(CalibrationAverage, 0, 50);
ScanNode();
NodeMoved = 0;
while (((CurrentNode != CalibrationNode) || !NodeMoved) && !CalibrationBreak)
{
  NodeMoved = 1;
  ScanNode();
}
CalibrationStart = CycleTime;
delay(1000);
for (byte i = 0; ((i < 48) && !CalibrationBreak); i++)
{
  for (byte k = 0; ((k < 3) && !CalibrationBreak); k++)
  {
    while (((CurrentNode != CalibrationNode) || !NodeMoved) && !CalibrationBreak)
    {
      ScanNode();
      NodeMoved = 1;
    }
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print(CurrentNode);
    lcd.setCursor(0,1);
    lcd.print(k);
    NodeMoved = 0;
    if (CycleTime > CalibrationMax)
    {
      CalibrationMax = CycleTime;
    }
    if (CycleTime < CalibrationMin)
    {
      CalibrationMin = CycleTime;
    }
    CalibrationAverage[i] = (CycleTime + CalibrationAverage[i]*k) / (k + 1);
  }
```

```
    CalibrationNode++;
    if (CalibrationVariance < (CalibrationMax - CalibrationMin))
    {
      CalibrationVariance = CalibrationMax - CalibrationMin;
      if (CalibrationVariance > CalibrationTolerance)
      {
        CalibrationBreak = 1;
      }
    }
    CalibrationMax = 0;
    CalibrationMin = 2000000;
  }

  while ((CurrentNode != 0) && !CalibrationBreak)
      {
        ScanNode();
      }

      if (CycleTime > CalibrationStart)
      {
        CalibrationChange = CycleTime - CalibrationStart;
      }
      else
      {
        CalibrationChange = CalibrationStart - CycleTime;
      }

  if ((CalibrationChange < CalibrationTolerance) && (CalibrationVariance <
CalibrationTolerance))
  {
    if (WritePC)
    {
      Serial.println(F("Success\t\tChange Start-
End:\tVariance:\tTolerance:\tNodes:\t1\t2\t3\t4\t5\t6\t7\t8\t9\t10\t11\t12\t13\t14\t15\t16\t17\t18\t
19\t20\t21\t22\t23\t24\t25\t26\t27\t28\t29\t30\t31\t32\t33\t34\t35\t36\t37\t38\t39\t40\t41\t42\t43\t
44\t45\t46\t47\t48\tSum:"));
      Serial.print(F("\t\t\t"));
      Serial.print(CalibrationChange);
      Serial.print(F("\t\t"));
      Serial.print(CalibrationVariance);
      Serial.print(F("\t\t"));
      Serial.print(CalibrationTolerance);
      Serial.print(F("\t"));
      for (byte i = 0; i < 48; i++)
      {
        Serial.print(F("\t"));
```

```
      Serial.print(CalibrationAverage[i]);
      CalibrationSum += CalibrationAverage[i];
      //delay();
     }
     Serial.print(F("\t"));
     Serial.println(CalibrationSum);
    }

    if (WriteLogger)
    {
     LoggerTextFile.print(F("Success\t"));
     LoggerTextFile.print(CalibrationChange);
     LoggerTextFile.print(F("\t"));
     LoggerTextFile.print(CalibrationVariance);
     for (byte i = 0; i < 48; i++)
     {
      LoggerTextFile.print(F("\t"));
      LoggerTextFile.print(CalibrationAverage[i]);
      CalibrationSum += CalibrationAverage[i];
      //delay();
     }
     LoggerTextFile.print(F("\tTolerance:\t"));
     LoggerTextFile.print(CalibrationTolerance);
     LoggerTextFile.print(F("\t"));
     LoggerTextFile.println(CalibrationSum);
    }

    if (WriteDisplay)
    {
     lcd.setCursor(0,0);
     lcd.print(F("Calibration      "));
     lcd.setCursor(0,1);
     lcd.print(F("Successful       "));
     LCDColor(255,200,0);
     delay(2000);
     lcd.setCursor(0,0);
     lcd.print(F("Variance:        "));
     lcd.setCursor(0,1);
     lcd.print(CalibrationVariance);
    }
   }
   else
   {
    if (WritePC)
    {
     Serial.print(F("Failure: Beginning & End Cycle Time:\t"));
```

```
  Serial.print(CalibrationChange);
  Serial.print(F("\tVariance:"));
  Serial.print(CalibrationVariance);
  Serial.print(F("\tTolerance:\t"));
  Serial.print(CalibrationTolerance);
 }

 if (WriteLogger)
 {
  LoggerTextFile.print(F("Failure\t"));
  LoggerTextFile.print(CalibrationChange);
  LoggerTextFile.print(F("\t"));
  LoggerTextFile.print(CalibrationVariance);
  LoggerTextFile.print(F("\tTolerance:\t"));
  LoggerTextFile.println(CalibrationTolerance);
 }

 if (WriteDisplay)
 {
  lcd.setCursor(0,0);
  lcd.print(F("Calibration        "));
  lcd.setCursor(0,1);
  lcd.print(F("Failure        "));
  LCDColor(0,255,240);
  delay(3000);
  if (CalibrationChange > CalibrationVariance)
  {
   lcd.setCursor(0,0);
   lcd.print(F("Not Yet        "));
   lcd.setCursor(0,1);
   lcd.print(F("Steady State        "));
  }
  else
  {
   lcd.setCursor(0,0);
   lcd.print(F("Too Much        "));
   lcd.setCursor(0,1);
   lcd.print(F("Variance        "));
   delay(2000);
   lcd.setCursor(0,0);
   lcd.print(F("Variance:       "));
   lcd.setCursor(0,1);
   lcd.print(CalibrationVariance);
  }
  delay(2000);
 }
```

```
  }
}



else if (Mode == 3)
{
  Mode = 1;




}


else if (Mode == 4)
{
  Mode = 1;
  WriteDisplay = 1;
  WriteLogger = 1;
  //LoggerTextFile = "";
  DateError = 0;
  // Date
  WritePC = 1;
  CurrentNode = 0;
  LastReading = 1;
  KnowsPosition = 0;
  ReverseCount = 0;
  ConfirmCount = 0;

  Bootup();
}

else if (Mode == 5)
{
  NodeMoved = 0;
  while((!(CurrentNode%2 == 0) || !NodeMoved) && (CycleTime < 100000))
  {
    NodeMoved = 1;
    ScanNode();
  }
  NodeMoved = 0;

  if (KnowsPosition == 1)
  {
```

```
  TrueRPM = (SensorSpacing[CurrentNode] + SensorSpacing[CurrentNode +
1])/CycleTime*60000000;

  WindRPM = (SensorWindRPM[CurrentNode] + SensorWindRPM[CurrentNode +
1])/CycleTime*60000000;

  WindSpeed = Constant1*WindRPM + Constant2;

  //Direction Stuff 99

  //Accel Stuff 99
  if(WindRPM > 200)
  {
    Mode = 6;
    if (WriteDisplay)
    {
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print(F("Entering"));
      lcd.setCursor(0,1);
      lcd.print(F("Ultra Speed Mode"));
    }
    if (WritePC)
    {
      Serial.println(F("I0"));
    }
    if (WriteLogger)
    {
      LoggerTextFile.println(F("I0"));
    }
  }
  else if (WindRPM < 80)
  {
    Mode = 1;
    if (WriteDisplay)
    {
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print(F("Entering"));
      lcd.setCursor(0,1);
      lcd.print(F("Normal Mode"));
    }
    if (WritePC)
    {
      Serial.print(F("N"));
      Serial.println(CurrentNode);
```

```
  }
  if (WriteLogger)
  {
   LoggerTextFile.print(F("N"));
   LoggerTextFile.println(CurrentNode);
  }
 }
 else
 {
  if ((WriteDisplay) && (LastDisplayNode > (WindRPM/1.5)))
  {
   lcd.clear();
   lcd.setCursor(8,0);
   lcd.print(F("kmh-H"));
   lcd.setCursor(0,0);
   lcd.print(WindSpeed);
   lcd.setCursor(0,1);
   lcd.print(F("H-Node:"));
   lcd.setCursor(8,1);
   lcd.print(CurrentNode);

   LastDisplayNode = 0;

   if(WindRPM < 1)
   {
    LCDColor(235,0,85);
   }
   else
   {
    LCDColor(255,200,0);
   }
  }
  else
  {
   LastDisplayNode++;
  }

  if (WritePC)
  {
   Serial.println(CycleTime);
  }

  if (WriteLogger)
  {
   LoggerTextFile.println(CycleTime);
  }
```

```
    }
  }
}


else if (Mode == 6)
{
  while((CurrentNode != 0) && (CycleTime < 50000))
  {
    ScanNode();
  }
  CurrentNode = 1;

  if (KnowsPosition == 1)
  {
    TrueRPM = SensorSpacing[CurrentNode]/CycleTime*60000000;

    WindRPM = SensorWindRPM[CurrentNode]/CycleTime*60000000;

    WindSpeed = Constant1*WindRPM + Constant2;

    //Direction Stuff 99

    //Accel Stuff 99

    if (WindRPM < 80)
    {
      Mode = 1;
      if (WriteDisplay)
      {
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print(F("Entering"));
        lcd.setCursor(0,1);
        lcd.print(F("Normal Mode"));
      }
      if (WritePC)
      {
        Serial.print(F("N"));
        Serial.println(CurrentNode);
      }
      if (WriteLogger)
      {
        LoggerTextFile.print(F("N"));
        LoggerTextFile.println(CurrentNode);
      }
```

```
}
else if ((WindRPM < 180))
{
  Mode = 5;
  if (WriteDisplay)
  {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print(F("Entering"));
    lcd.setCursor(0,1);
    lcd.print(F("High Speed Mode"));
  }
  if (WritePC)
  {
    Serial.print(F("H"));
    Serial.println(CurrentNode);
  }
  if (WriteLogger)
  {
    LoggerTextFile.print(F("H"));
    LoggerTextFile.println(CurrentNode);
  }
}
else
{
 if ((WriteDisplay))
  {
    lcd.clear();
    lcd.setCursor(8,0);
    lcd.print(F("kmh-I"));
    lcd.setCursor(0,0);
    lcd.print(WindSpeed);
    lcd.setCursor(0,1);
    lcd.print(F("I-Node:"));
    lcd.setCursor(8,1);
    lcd.print(CurrentNode);

    LastDisplayNode = 0;

    if(WindRPM < 1)
    {
      LCDColor(235,0,85);
    }
    else
    {
      LCDColor(255,200,0);
```

```
      }
     }

     if (WritePC)
     {
      Serial.println(CycleTime);
     }

     if (WriteLogger)
     {
      LoggerTextFile.println(CycleTime);
     }
    }
  }
}



}


void ScanNode()
{
  unsigned long LoopTimeINI = millis();

  if(Mode == 5)
  {
    LastReading = 1;
    while (analogRead(SensorPin1) < 700)
    {
      if ((millis() - LoopTimeINI) > 1000)
      {
        //return;
      }
    }
    CurrentNode++;
    while (analogRead(SensorPin1) > 700)
    {
      if ((millis() - LoopTimeINI) > 1000)
      {
        //return;
      }
    }
  }
  else
  {
```

```
  if (!LastReading)
  {
   LastReading = 1;
   while (analogRead(SensorPin1) > 700)
   {
    if ((millis() - LoopTimeINI) > 1000)
    {
     //return;
    }
   }
  }
  else
  {
   LastReading = 0;
   while (analogRead(SensorPin1) < 700)
   {
    if ((millis() - LoopTimeINI) > 1000)
    {
       //return;
    }
   }
  }
}

TimeStamp = micros(); //69
if (TimeStamp > PreviousTimeStamp)
{
 CycleTime = TimeStamp - PreviousTimeStamp;
}
else
{
 CycleTime = TimeStamp + PreviousTimeStamp;
}
PreviousTimeStamp = TimeStamp;

if (!KnowsPosition)
{
 if ((analogRead(SensorPin2) > 700))
 {
  if (!LastReading)
  {
   KnowsPosition = 1;
   CurrentNode = 1;
  }
  else
  {
```

```
      KnowsPosition = 1;
      CurrentNode = 0;
    }
  Serial.println(CurrentNode);
  }
  else
  {
   CurrentNode++;
  }
}
else
{
  CurrentNode++;
  if (analogRead(SensorPin2) > 700)
  {
    if ((CurrentNode != 48) && (CurrentNode != 49))
    {
      if (Mode < 5)
      {
        if(ReverseCount >= 1)
        {
          ConfirmCount++;
          if (LastReading)
          {
            CurrentNode = 0;
          }
          else
          {
            CurrentNode = 1;
          }
          ReverseOptions();
        }
        else if(ReverseCount == 0)
        {
          if (WritePC)
          {
            Serial.println(F("C"));
          }
          if (WriteLogger)
          {
            LoggerTextFile.println(F("C"));
          }
          KnowsPosition = 2;
          if (!LastReading)
          {
            CurrentNode = 0;
```

```
      }
      else
      {
       CurrentNode = 1;
      }
     }
    }
    else
    {
     if (WritePC)
     {
      Serial.println(F("C"));
     }
     if (WriteLogger)
     {
      LoggerTextFile.println(F("C"));
     }
     if (!LastReading)
     {
      CurrentNode = 0;
     }
     else
     {
      CurrentNode = 1;
     }
    }
   }
   else if (!LastReading && (Mode < 5))
   {
    if (WritePC)
    {
     Serial.println(F("R"));
    }
    if (WriteLogger)
    {
     LoggerTextFile.println(F("R"));
    }
    CurrentNode = 0;
    ReverseCount++;
    KnowsPosition = 2;
    ConfirmCount = 0;
    ReverseOptions();
   }
   else
   {
    CurrentNode = 0;
```

```
      KnowsPosition = 1;
      ReverseCount = 0;
      ConfirmCount = 0;
      CalibrationBreak = 0;
     }
   }
  }

  if (CurrentNode > 48)
  {
   if (WritePC && (KnowsPosition == 1))
   {
    Serial.println(F("C"));
   }
   if (WriteLogger && (KnowsPosition == 1))
   {
    LoggerTextFile.println(F("C"));
   }
   KnowsPosition = 2;
  }

  if (CurrentNode > 52)
  {
   CurrentNode = 51;
   LCDColor(235,0,85);
  }
}


void ReverseDisplay()
{
        lcd.setCursor(0,1);
        lcd.print(F("Reverse Count:  "));
        lcd.setCursor(14,1);
        lcd.print(ReverseCount);
        LCDColor(0,0,230);
}

void ReverseOptions()
{
  if (WriteDisplay)
  {
  if (ReverseCount > 0)
  {
   CalibrationBreak = 1;
```

```
if (ReverseCount >= 15)
{
   ReverseCount = 1;
   ConfirmCount = 0;
}
if (ReverseCount >= 10)
{
 if (ConfirmCount >= 6)
 {
   if (WritePC)
   {
    Serial.println(F("E"));
   }
   if (WriteLogger)
   {
    LoggerTextFile.println(F("E"));
   }
   Mode = 4;
   lcd.setCursor(0,1);
   lcd.print(F("Mode Confirmed    "));
   delay(1000);
   ReverseCount = 0;
   ConfirmCount = 0;
 }
 else if (ConfirmCount >= 1)
 {
    lcd.setCursor(0,1);
    lcd.print(F("Confirm x    "));
    lcd.setCursor(9,1);
    lcd.print(ConfirmCount);
 }
 else
 {
    lcd.setCursor(0,0);
    lcd.print(F("Mode: Reboot     "));
    ReverseDisplay();
 }
}
else if (ReverseCount >= 8)
{
 if (ConfirmCount >= 6)
 {
   if (WritePC)
   {
    Serial.println(F("T"));
   }
```

```
    if (WriteLogger)
    {
      LoggerTextFile.println(F("T"));
    }
    Mode = 3;
    lcd.setCursor(0,1);
    lcd.print(F("Mode Confirmed    "));
    delay(1000);
      ReverseCount = 0;
      ConfirmCount = 0;
   }
   else if (ConfirmCount >= 1)
   {
    if(WriteDisplay)
      lcd.setCursor(0,1);
      lcd.print(F("Confirm x    "));
      lcd.setCursor(9,1);
      lcd.print(ConfirmCount);
   }
   else
   {
      lcd.setCursor(0,0);
      lcd.print(F("M:Equipment Test"));
      ReverseDisplay();
   }
 }
 else if (ReverseCount >= 6)
 {
  if (ConfirmCount >= 6)
  {
    if (WritePC)
    {
      Serial.println(F("A"));
    }
    if (WriteLogger)
    {
      LoggerTextFile.println(F("A"));
    }
    Mode = 2;
    CalibrationBreak = 0;
    lcd.setCursor(0,1);
    lcd.print(F("Mode Confirmed    "));
    delay(1000);
    ReverseCount = 0;
    ConfirmCount = 0;
   }
```

```
  else if (ConfirmCount >= 1)
  {
     lcd.setCursor(0,1);
     lcd.print(F("Confirm x    "));
     lcd.setCursor(9,1);
     lcd.print(ConfirmCount);
  }
  else
  {
     lcd.setCursor(0,0);
     lcd.print(F("Mode: Calibrate  "));
     ReverseDisplay();
  }
 }
 else if (ReverseCount >= 4)
 {
  if (ConfirmCount >= 6)
  {
   if (WritePC)
   {
    Serial.println(F("N"));
   }
   if (WriteLogger)
   {
    LoggerTextFile.println(F("N"));
   }
   Mode = 1;
   lcd.setCursor(0,1);
   lcd.print(F("Mode Confirmed    "));
   delay(1000);
   ReverseCount = 0;
   ConfirmCount = 0;
  }
  else if (ConfirmCount >= 1)
  {
     lcd.setCursor(0,1);
     lcd.print(F("Confirm x    "));
     lcd.setCursor(9,1);
     lcd.print(ConfirmCount);
  }
  else
  {
     lcd.setCursor(0,0);
     lcd.print(F("Mode: Normal    "));
     ReverseDisplay();
  }
```

```
    }
    else if (ReverseCount >= 2)
    {
     if (ConfirmCount >= 6)
     {
       Mode = 1;
       lcd.setCursor(0,1);
       lcd.print(F("Mode Confirmed     "));
       delay(1000);
       ReverseCount = 0;
       ConfirmCount = 0;
       WriteDisplay = 0;
     }
     else if (ConfirmCount >= 1)
     {
        lcd.setCursor(0,1);
        lcd.print(F("Confirm x        "));
        lcd.setCursor(9,1);
        lcd.print(ConfirmCount);
     }
     else
     {
        lcd.setCursor(0,0);
        lcd.print(F("Disable Display?  "));
        lcd.setCursor(0,1);
        lcd.print(F("Rotate CW/CCW x2   "));
        ReverseDisplay();
     }
    }
    else
     {
        lcd.setCursor(0,0);
        lcd.print(F("Reverse Detected   "));
        ReverseDisplay();
     }
  }
}
else
{
  if (ReverseCount >= 2)
    {
     if (ConfirmCount >= 6)
     {
       Mode = 1;
       ReverseCount = 0;
       ConfirmCount = 0;
```

```
      WriteDisplay = 1;
      lcd.setCursor(0,1);
      lcd.print(F("Display On       "));
     }
    }
}
}

void BlinkLCD(byte a, byte b, byte c, byte d, byte e, byte f)
{
 if(WriteDisplay)
 {
   LCDColor(a,b,c);
   delay(250);
   LCDColor(d,e,f);
   delay(250);
   LCDColor(a,b,c);
   delay(250);
   LCDColor(d,e,f);
   delay(250);
   LCDColor(a,b,c);
   delay(250);
   LCDColor(d,e,f);
 }
}

void LCDColor(byte a, byte b, byte c)
{
 if(WriteDisplay)
 {
   analogWrite(DisplayRed, a);
   analogWrite(DisplayBlue, b);
   analogWrite(DisplayGreen, c);
 }
}
```