Weather Station

Final Report

Javier Salazar, Anthony Giuntini, Boris Dostinov, Damian Hoffman
Department of Electrical Engineering, The University of Texas at Arlington
Junior Design
Arlington, TX 76010

***Abstract*— For the Junior Design project, we have built a weather station that operates outdoors and measures temperature, barometric pressure, ultraviolet radiation levels, humidity, wind direction, and wind speed. The data is transmitted wirelessly to the ThingSpeak website. Additionally, the weather station has an LCD for current readings viewing. This report describes the progress of the design and construction of the weather station since the second report. After the second report, we updated the master code file and developed the case for the electrical systems. This report expands on the results, difficulties, and design changes during the ongoing development process.**

Keywords—Arduino Yun microcontroller (Seeeduino Cloud), humidity sensor (DHT11), ultraviolet sensor (UV Index), temperature sensor (TMP36), pressure sensor (BMP280), Hall effect sensor for wind speed, wind vane with a rotary encoder (potentiometer), LCD interface, magnets, Birdhouse casing, Matlab data analysis, ThingSpeak, and Anemometer.

I. Introduction

The first report laid out a basic outline of the project and each member's responsibilities, as well as the preliminary design of the system. The second report examined the design with further detail, lists changes to the team responsibilities, issues with design, and additional components used since the first report. In this final report, we have finalized the weather station by designing a case, fixing code bug issues, implementing the wind vane and anemometer to the case design, and shifting circuitry from a breadboard to a protoboard.

A) System Requirements

There were three basic system requirements. The first stated that the system must be able to measure the temperature, pressure, wind speed, and wind direction. Secondly, the system must upload the measured data to the Thingspeak website. Finally, the system must be capable of surviving outdoors in standard weather conditions.

B) Project Responsibilities

We have outlined a project schedule to facilitate the development process (see Appendix section and Figure 1 and 2). Each team member has different responsibilities that have changed since the first initial report due to different areas of expertise. Javier oversees the main programming of the microcontroller and analysis program. He will be responsible for allowing the controller to collect from the sensors, upload, and analyze the data. Additionally, Javier is responsible for the wind vane design and protoboard assembly (soldering). Damian will take responsibility for the sensors which include: temperature, pressure, ultraviolet, and humidity. This includes designing the sensors, and the basic programming that allows the sensors to communicate to the Arduino. Moreover, Damian constructed, decorated, and designed the case of the weather station. Boris will take the leading role in designing the wind speed Anemometer and coding to measure wind speed as well as wind direction code. We will place the electronics in a birdhouse, but it is important to ensure that the board and the sensors remain water-resistant during rainy weather conditions. Finally, Anthony, the group leader, is responsible for determining a viable way to power the system as well as designing a user interface for the device. This includes any coding associated with making the user interface. Additionally, Anthony greatly optimized the code and ensured that the code didn't go over the program storage space on the device. Each individual is responsible for their respective parts, but since we are a team, nothing is done completely independently. The project timeline shown in the Gantt chart is the updated design schedule from the first initial report. The project roles have shifted since the creation of the Gantt chart. Boris is still responsible for the anemometer design, wind speed code, and wind direction code. Damian's responsibilities have shifted due to Javier having previous experience with 3D CAD modeling. Damian is responsible for the assembly of the weather station components in the case, and Javier is responsible for the 3D-printed wind vane design. Moreover, two extra sensors (uv level and humidity) were added to the design and will be integrated by Damian. Javier is still responsible for the matlab data, the main programming for the microcontroller, and internet connectivity. Anthony's responsibilities have shifted as well. Initially, he was tasked with writing the second report, however, now the project was split up among all of the team members. Anthony remains in charge of the LCD display's implementation and device's power. Moreover, Anthony optimized the code and greatly helped to make sure the code worked as designed. Since timelines and projected schedules have changed, the

timeline and Gantt chart have been updated (shown in the Appendices and Figure 1 and 2).

The basic software and hardware development tools (including measurement equipment) includes Fritzing Schematic Image Software, Microsoft Visio, Microsoft Office suite, SolidWorks CAD modeling software, and myDAQ software suite with student myDAQ system (multimeter, oscilloscope). Additionally, when we are at the lab, we can use the oscilloscope, function generator, power supplies, and multimeters available instead of the myDAQ. We need the myDAQ to debug the hardware, SolidWorks to construct 3d model components, Fritzing to design visual diagrams of our schematic, Multisim to simulate the circuits and Microsoft Visio and Microsoft Office to compile parts list, make presentations, write reports, and other organizational elements.

Task Name	Start	End	Duration (days)
Brainstorming Ideas	1/17/2017	1/31/2017	14
Parts List	1/31/2017	2/7/2017	7
Lab Report #1	2/7/2017	2/21/2017	14
Order Parts/Research Programming	2/20/2017	2/28/2017	8
Anemometer Design/Wind Speed Code/Wind Direction Code/ Hall Effect Circuit	2/28/2017	4/12/2017	43
Case Design & Assembly/Sensors Programming/Components-Case Integration	2/28/2017	4/12/2017	43
Main Programming (WiFi) /Wind Vane Design / Protoboard Assembly	2/28/2017	4/12/2017	43
Power System/LCD GUI/Main Systems Debugging&Optimization	2/28/2017	4/12/2017	43
Report #2	3/28/2017	4/4/2017	7
Final Testing/Evaluation	4/12/2017	5/1/2017	19
Final Report	4/30/2017	5/4/2017	4
Project Presentation	5/1/2017	5/2/2017	1
Legend			
Color	Team Members		
	All Team Members		
	Javier Salazar		
	Damien Hoffman		
	Boris Dostinov		
	Anthony Giuntini		

Figure 1. Updated timeline schedule after second report (see Appendix).

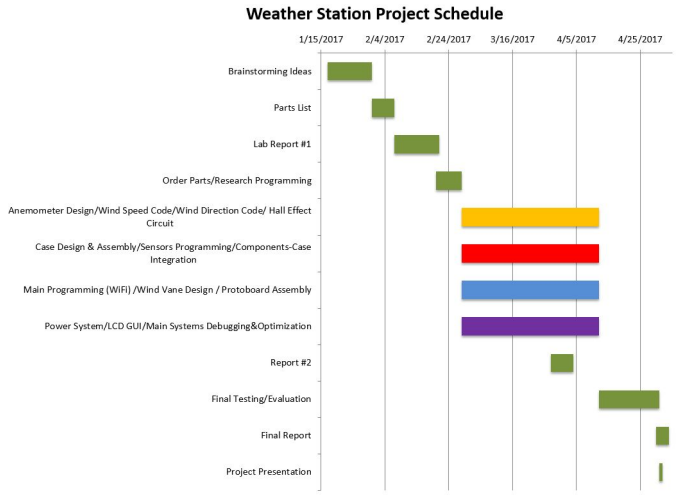

Figure 2. Updated Gantt chart with shifted responsibilities after second report (see Appendix).

II. Design Approaches, Alternatives & Problems

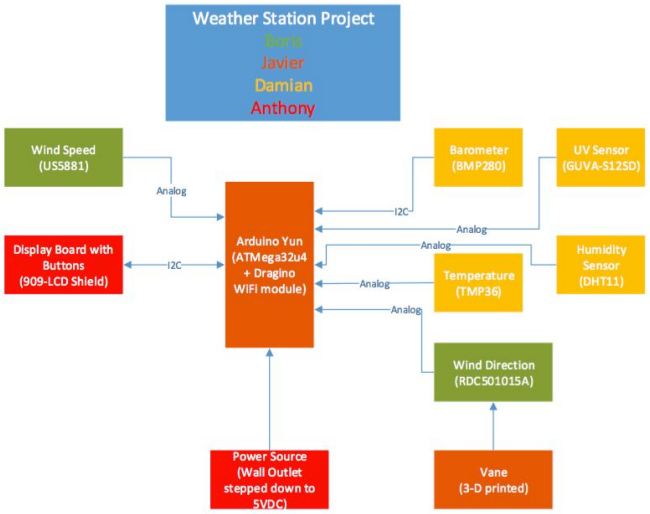

Figure 3. Complete block diagram of weather station (see Appendix).

Figure 1 shows the components of the weather station and the interaction between the blocks. The blocks are color-coded according to the team member assigned to work on that block. Below, team members describe the approaches explored for their respective blocks and provide a cost-performance trade-off analysis for each approach. The wind speed, wind direction, and temperature sensors have analog based outputs. The Arduino Yun microcontroller has sufficient analog pins to process the data from these sensors. On the other hand, the LCD module and barometer have digital outputs that are sufficiently covered in the microcontroller's digital I/O pins. Many of the components will communicate with the microcontroller serially using the I2C protocol. This protocol was chosen, for the sake of uniformity and simplicity. Shown below in figure 4, we have the complete wiring of the weather station. This shows how the sensors and different components connect to the microcontroller. A protoboard is used since the LCD shield covers some pins needed for the sensors and any power systems can be utilized here.

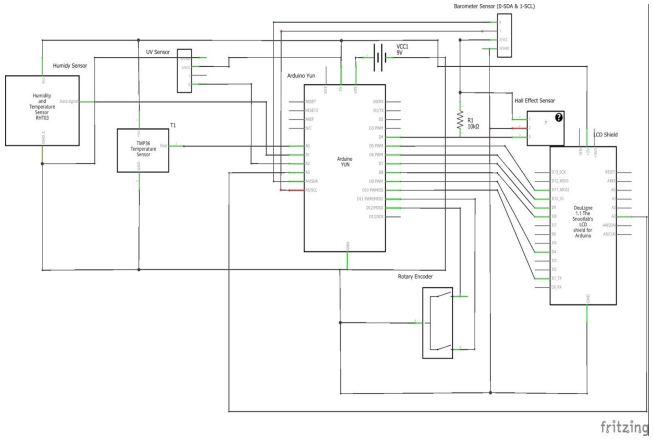

Figure 4. Schematic layout of weather station components (see appendix).

A. Controller

The Arduino Yun combines the power of Linux OpenWrt and the ease of the Arduino C language in one board without requiring additional shields to be added to the device. The microcontroller offers on-board Wi-Fi connectivity (Atheros AR9331) and the processor is based on the ATmega32u4 which offers 20 digital I/O pins and 12 analog I/O pins. This device has digital IO, analog IO, and I2C which is all of the communication protocols required for the sensors and LCD interface. Mouser offers a similar device "clone" made by a different company called Seeed Studio Seeeduino for $50. This device offers enough I/O pins to satisfy the minimum requirements for the project and there is sufficient documentation online on how to work with this device. An alternative for the project previously considered is a simpler microcontroller such as the Arduino Uno ($20) and a Wi-Fi shield like the Xbee shield ($30) but the Seeeduino costs the same and comes in one package for ease of use so this integrated controller is the better option.

TABLE 1

Part Selection For Microcontroller			
	Required	Solution #1 Seeeduino Cloud	Solution #2 Raspberry Pi
Digital I/O Pins	8	20	40 GPIO
Analog I/O Pins	4	12	40 GPIO
Flash Memory	5 MB	16 MB	1 GB
Connectivity	Wi-Fi	Wi-Fi	Ethernet + Wi-Fi (additional adapter)
Cost	Least	$50	$60
Power Consumption	Least	240mA	400mA

Here is the table where we laid out the pros and cons of the two solutions we considered for the microcontroller. After consideration, we decided to use solution #1 for the microcontroller which is the Seeeduino (Arduino Yun clone) mouser part number 713-102010021. It is the cheaper solution that also happens to be the most convenient. It contains Wi-Fi connectivity on board so it's less of a hassle since it has integrated support. The first drawback is storage space as opposed to solution #2 but this should not be a problem since we only need a couple of megabytes at most. The second drawback is the number of GPIO pins but this isn't a problem since we still have plenty of pins available to connect the blocks to the system. Additionally, I have experience working with the Arduino language since I helped code the robot in the EE1104 project so this is my preference since I will do most of the coding. I don't have any experience with the Raspberry Pi so it would take a longer timeframe to make the coding work with the parts. For these reasons, we have decided to use solution #1 for the microcontroller.

The main code of the system is shown in the appendix section of the report. Moreover, this report will explain certain aspects of the code that are imperative to connectivity to the ThingSpeak server. The Arduino Yun has its own library that it uses to initialize ethernet/WiFi connectivity. The problem is that the client is outdated for the specific model of the device we are using, Searching online for solutions, we found a similar library that allows for internet connectivity that is updated and the model of the device works well under. This is the Bridgeclient.h at line 6. The ThingSpeak website also has its own API for Arduino devices that it uses to send/receive data and that is the ThingSpeak.h library at line 5. To communicate with ThingSpeak, we will need a couple of things to write to a channel. We need the IP address of the server, the channel number associated with my account, and the write API key to give permission to write data in. These values are stored as strings and integers within the program as we will need them later on. We initialize a request to open wireless communication at line 10. Within the setup class, we tell it to begin internet connectivity at line 27. This command will allow the Arduino device to connect according to its stored wifi or ethernet settings. At line 28, the Arduino uses the ThingSpeak library with the wireless connectivity protocol to establish a connection with the server. An important consideration for data upload is the 15 seconds update delay that ThingSpeak has for free accounts. We are not allowed to upload all data collected at the end of every hour for example. We can only upload 1 piece of data per field in a channel and no more per update cycle. In other words, the weather station will require a constant internet connection if the user wishes to have data uploaded to the ThingSpeak server. Within the loop class, we use the delay function to make sure data isn't sent under the minimum time. Following that line, we have a local variable that stores the return value of a function. The system is designed so that every sensor has a function and it just returns the value of the sensor. That is why we need a local variable to store that because ThingSpeak has a certain rule of data upload. For some reason, ThingSpeak only really accepts data in a string form and nothing else. If data is sent that happens to be an integer then it will upload one data value but stop taking in other future values per session connect. This behavior can be solved by using strings. This was a serious problem that

is not well documented online for some mysterious reason. We managed to find a forum topic thread that discussed this data uploading issue and implemented a similar design to avoid this issue. The important lines (39-41) are the commands from the library that will send the strings to the server. An efficient way to upload the data is to use the ThingSpeak.setField() command to set a variable with a field on the channel. We are allowed maximum 8 fields per channel so this is enough since we have below 8 sensors. We set the variables to the specific channel and then use the ThingSpeak.writeFields() command using the channel number and API key to provide authentication and upload our data. The writeFields is the command that uploads that data according to whatever fields were set to the variables used. This is the basic process of how data recorded from our sensors is sent to the ThingSpeak server. In the code, there are various functions that record temperature, uptime, and hall sensor revolutions. Damian has the functions built for the rest of the sensors and will explain his progress and code further in the report. After running some tests, the data is successfully sent to the ThingSpeak server. As shown below, here is a graph of the temperature in a room over time during a data collection test. Another issue we experienced in order to upload the data to the ThingSpeak server is the wifi connectivity. On the device itself, it only supports networks that have a key phrase that is WPA/WEP on any standard 802.11 a/b/g/b/n. Unfortunately, the UTA network requires a username and password in order to connect. There is no universal keyphrase so this feature is not supported by the Arduino Yun. In order to workaround this issue, we need to create our own wifi hotspot since this weather unit will be outside. Using a mobile device, or a laptop that is connected to the internet, we can create a hotspot that uses only a keyphrase password and that would take care of the issue.

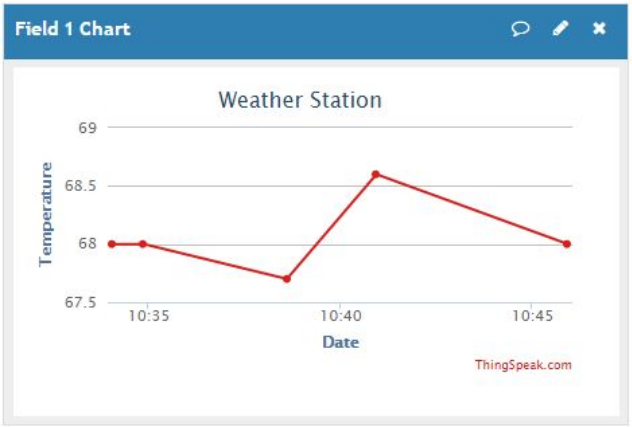

Figure 5. Matlab Data Plot of TMP36 Sensor.

One aspect of code design that needed to be revisited is code allocation. The libraries needed for internet connectivity and ThingSpeak communication took up a certain percentage of the Arduino memory. Any variables that we create that store the sensor values are taking up space on the device as well. If we compile the code we get the following information: (91%) of program storage space - Max is 32kB - Global variables use (54%) of dynamic memory - Max is 2.5kB. This tells us how much program space is available and how much dynamic memory (SRAM) is free to have multiple variables. The amount of code we have isn't enough to justify a large space requirement. That is why the libraries take up a significant amount of space on the device. Taking that into consideration, there is not a reason why we would go over the memory limits of the microcontroller but in the event that we are close, then we will need to consider memory allocation. Changing large memory data types to small memory data types, reducing global variables to local variables, reduce code, and other things to optimize the space on the Arduino. An alternative solution is to store strings in flash instead of SRAM since SRAM is limited. Strings can take up quite a bit of SRAM so forcing it to stay in flash memory, meaning it won't get copied to SRAM at program start, will substantially improve memory space on the limited SRAM.

B. Wind Vane

TABLE 2

Parts Selection for Wind Vane			
	Required	Solution #1	Solution #2
Cost	N/A	$2.50	~$5
Material (Quality)	N/A	ABS	Plastic/Cardboard/Misc.
Weight	Least	50g	Unknown
Vane Cross Area (Power)	Most	~60 cm^2	Unknown

After consideration, we decided to use solution #1 for the wind vane design which is to 3d print the part using UTA's Fablab and the SolidWorks program. The SolidWorks program is offered to students for free so it is a good option. Additionally, Autodesk Inventor is also offered to students for free so this is another program to use. However, I have experience using SolidWorks, not Inventor so this is my preference for the program. Printing the design leads to a high-quality solid wind vane that can be optimized precisely using the computer. Creating the wind vane using common materials and parts is not something that we want to do because the design might be finicky and difficult to adapt to fit the rotary encoder that will record the data. Printing the design is a higher quality construction that is easier to modify in the computer if a change is required while

solution #2 does not have that luxury. For these reasons, we will use the Fablab to print the wind vane that will be used for the project so we are going with solution #1.

We designed a wind vane to fit on top of the rotary sensor. The program of choice is SolidWorks since the school offers a free student license. First, in order to design a wind vane we needed to make measurements of the rotary sensor since the wind vane will fit on top of the sensor. After making the measurement, we designed the model as shown below.

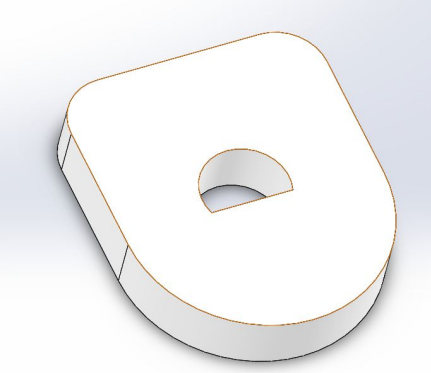

Figure 6. Rotary Sensor 3D model.

After knowing the size of the input on the rotary sensor, we needed to know the tolerance level of the PolyPrinters located in the fabrication lab at UTA. I designed a test part that would fit into the sensor with a tight fit but loose enough that it would fit the sensor perfectly. I printed the part below in figure 7.

Figure 7. Wind Vane Adapter Test Part.

The adapter part fit perfectly so all that was needed at this point is to print the 3D wind vane so that we can test the full wind vane system and check how accurate it is. The testing still needs to be done, but the wind vane is printed. Looking at some online designs and taking the maximum dimensions of the PolyPrinters into consideration, I designed the vane as illustrated below in figure 8.

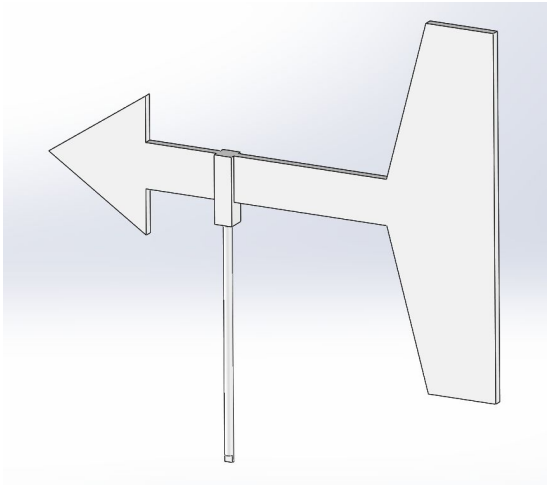

Figure 8. 3D Wind Vane Design.

The cross area on the back side of the wind vane should be enough to allow the vane to freely rotated. The rotary sensor itself is a low-torque device. That is, the sensor does not require excessive force to rotate which makes it ideal for the wind vane design that generates low force due to the "wingspan". The rotary sensor, essentially, is a potentiometer that will output a specific voltage depending on the rotation from an axis.

A. User Interface

In addition to the displayed data on the website, the weather station also has a LCD display and buttons to provide the user with a direct interface to the device. The interface (an Olimex Shield-lcd-16x2) has four buttons as well as a LCD display that can show two rows of 16 characters each. The 16x2 display limits the amount of data shown on one page, but the use of a menu and indexing system allows for more user controlled options as well as reducing the complexity of the interface, which is desired. See figure 9 for the profile of the display and buttons.

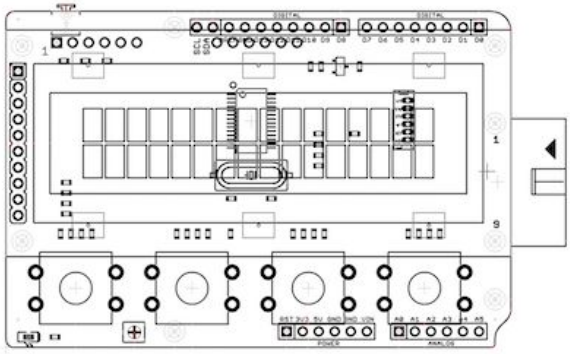

Figure 9. Profile of the Olimex 16x2 Display Shield.

The interface follows a general theme where the top 16 characters are used to display information, while the bottom 16 characters indicate the actions of each button. Since the

bottom row lies directly above the buttons, there are three characters that can clearly relate to each button. There are two options in the main menu: Current readings (labelled by CR), and Settings (labelled by SET). See Figure 10 for an image of the home menu.

Figure 10. Image of the display with the current home menu.

The current readings option from the main menu, takes you to a screen that displays readings from the sensors. Data from a single sensor is shown in the first row of characters. The user can use the two middle buttons (labelled in the second row by "<-" and "->") to scroll between the data from the other sensors. The sensor readings are taken when the current readings screen is entered. Like the System Info screen, the user can use the "EX" button to return to the main menu. Figure 12 shows the current design of the current readings screen.

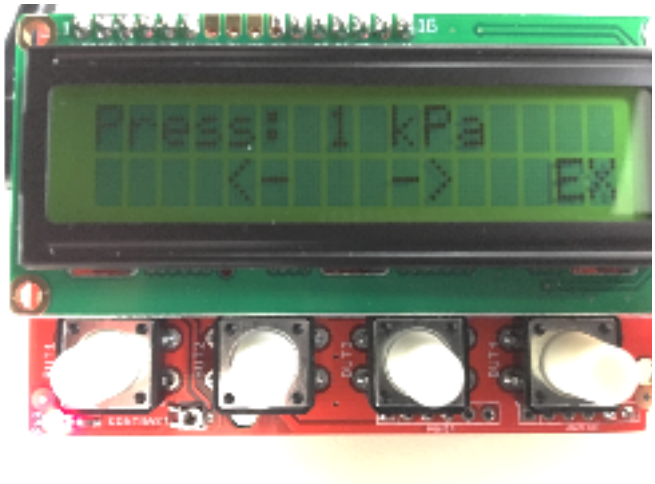

Figure 12. Image of the current readings screen.

The third and final main menu option takes you to the settings menu. In settings, the user can switch between several options using the same method as the Current Readings screen. When the user finds the setting that they want to change, the select button ("SEL") will choose that option and change the screen to one that will allow the user to change the setting. The only setting is the "calibrate north" setting. This setting provides a way for the user to indicate which direction is North for purposes of an accurate wind direction reading. The settings submenu and options also have exit buttons.

In the original menu design, there was a system info submenu (visible in figure 10). However, due to a shortage of memory on the device, we determined that it would be beneficial to remove this submenu.

B. Temperature, Pressure, Humidity and UV

The Grove Barometer sensor is a breakout board featuring the BMP280 barometer, designed for easy connection to the

Arduino Yun via the I2C interface which transfers information serially. The breakout board is connected to the Arduino Yun using the grove connecter, which has four pins: The first being Vcc-5volts, the next ground, and the final two are SDA (serial data) and SCL (serial clock) which are used to transfer information via I2C. The original temperature sensor for the weather station was discarded because of its redundancy with a temperature sensor already being built into the Grove Barometer. There are five different power settings available for this sensor: Ultra low power, Low power, Standard resolution, High resolution, and Ultra high resolution. The recommended setting for the sensor when being used for a weather station is the ultra-low power mode, since the accuracy does not need to be that high. The following chart shows the different power modes and their corresponding pressure resolutions.

Oversampling setting	Pressure oversampling	Typical pressure resolution	Recommended temperature oversampling
Pressure measurement skipped	Skipped (output set to 0x80000)	–	As needed
Ultra low power	×1	16 bit / 2.62 Pa	×1
Low power	×2	17 bit / 1.31 Pa	×1
Standard resolution	×4	18 bit / 0.66 Pa	×1
High resolution	×8	19 bit / 0.33 Pa	×1
Ultra high resolution	×16	20 bit / 0.16 Pa	×2

Figure 13. Power settings and their corresponding pressure resolutions.

The next chart shows the different power modes as they correspond to resolution of the temperature measurement.

osrs_t[2:0]	Temperature oversampling	Typical temperature resolution
000	Skipped (output set to 0x80000)	–
001	×1	16 bit / 0.0050 °C
010	×2	17 bit / 0.0025 °C
011	×4	18 bit / 0.0012 °C
100	×8	19 bit / 0.0006 °C
101, 110, 111	×16	20 bit / 0.0003 °C

Figure 14. Power settings and their corresponding temperature resolutions.

And finally, the following chart shows the typical current consumption for the device.

Oversampling setting	Pressure oversampling	Temperature oversampling	I_{DD} [µA] @ 1 Hz forced mode	
			Typ	Max
Ultra low power	×1	×1	2.74	4.16
Low power	×2	×1	4.17	6.27
Standard resolution	×4	×1	7.02	10.50
High resolution	×8	×1	12.7	18.95
Ultra high resolution	×16	×2	24.8	36.85

Figure 15. Power settings and their corresponding current consumption.

For our weather station we have decided not to use batteries but we still are setting the BMP280 to ultra-low power mode to conserve energy, so that we are not wasteful. The one we decided not to go with is the Grove Barometer (High Accuracy). it was priced at $19.90. With our choice of the BMP280 costing $8.90, that's a savings of $11.

Parameter	Symbol	Conditions		Min	Typ	Max	Unit
Operation Supply Voltage	V_{DD}			1.8	3.3	3.6	V
Operation Temperature	T_{OP}			-40	25	85	℃
Average Operation Current (Pressure Measurement under One Conversion per Second)	$I_{DD(P)}$	DSR*	4096		85.2		uA
			2048		42.6		
			1024		21.3		
			512		10.7		
			256		5.3		
			128		2.7		
Average Operation Current (Temperature Measurement under One Conversion per Second)	$I_{DD(T)}$	DSR*	4096		68.8		uA
			2048		34.4		
			1024		17.2		
			512		8.6		
			256		4.3		
			128		2.2		
Conversion Time of Pressure or Temperature	t_{CONV}	DSR*	4096		65.6		ms
			2048		32.8		
			1024		16.4		
			512		8.2		
			256		4.1		
			128		2.1		
Peak Current	$I_{DD(R)}$	During conversion			1.3		mA
Standby Supply Current	$I_{DD(ST)}$	At 25℃				0.1	uA
Serial Data Clock Frequency	f_{SCL}	I²C protocol, pull-up resistor of 10k				400	kHz
Digital Input High Voltage	V_{IH}			0.8			V_{DD}
Digital Input Low Voltage	V_{IL}					0.2	V_{DD}
Digital Output High Voltage	V_{OH}	I_O = 0.5 mA		0.9			V_{DD}
Digital Output Low Voltage	V_{OL}	I_O = 0.5 mA				0.1	V_{DD}
Input Capacitance	C_{IN}					10	pF

*DSR stands for Down Sampling Rate.

Figure 16: Power settings and their correspondent settings.

The high accuracy sensor offers higher accuracy for the temperature and pressure readings and it uses slightly less current than the BMP280, while in the lowest resolution as shown in the figure below. The current consumption is lower by 0.3 microAmps/0.8 microAmps for the pressure reading/temperature reading. Also, the input voltage for the high accuracy barometer is 3.3 V. Since we're using the measurements for a weather station we are not concerned with high accuracy results; a variance of +/-1 degree Celsius is acceptable. Also, for the weather station power consumption is not a priority since it will be plugged into an outlet, and the difference between the two parts is from 0.3 microAmps to 0.8 microAmps so it is not significant enough to justify the price difference.

The Grove UV sensor is a breakout board featuring the GUVA-S12D UV sensor, designed for easy connection to the Arduino Yun. The UV sensor is a three-terminal device, one pin for a supply voltage of 5 volts, and another for the ground, and the final for the output voltage to be measured by the A0 input pin. The Arduino takes analog measurement of the resulting output voltage and converts it to digital. The ADC is a 10-bit converter that uses the input voltage (5 volts) as Vref. Using the following equation:

Voltage(per bit)= $\frac{Vref}{2^N}$ **, N=10**

We get 5V/1024 which gives the value of 4.88 mV per digital bit. For the purpose of our project we desired the measurements to be tested at a frequency of once per minute. Given that UV level does not change that rapidly there is no need for frequent UV checks. Below is the chart showing the relationship between the wavelength of light received and Responsivity output given in the unit of Area/Watt.

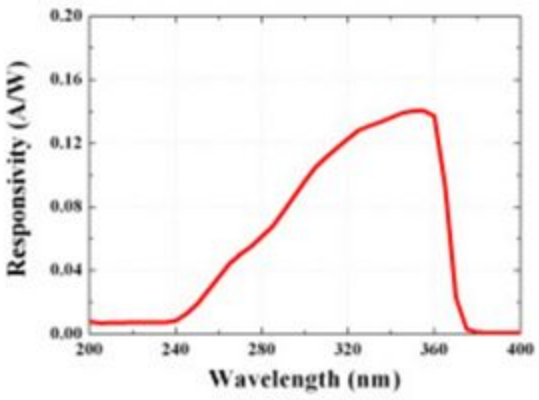

Figure 17. Responsivity vs Wavelength.

This makes sense with detection of range of the UV sensor being 240nm-370nm. Based upon the output voltages we get intensity of light and from there we can estimate the UV index level. The UV Index is based upon the international guidelines of the World Health Organization. Each level of severity comes with risk from unprotected sun exposure. For example, in UV index, 5 is a moderate risk level. This sensor was chosen for added functionality of the weather station; it was not part of the original design but was added later because it was chosen by its price $9.90 and its availability to ship immediately.

The next sensor is the humidity sensor, the original choice was Honeywell HIH7120-021-001, but this sensor was broken while in testing (snapped off a connector pin). Fortunately, a member of the team happened to have a

humidity sensor of his own, the DHT11. So this part was used because of ease of access being owned by a member of the group already and the price being zero.

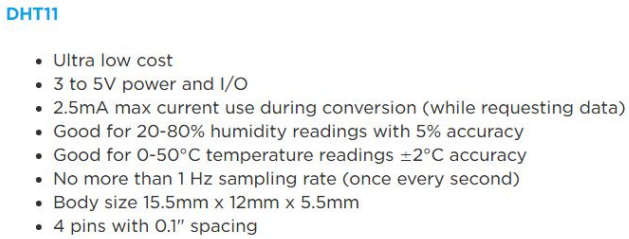

Figure 18. DHT11 Humidity Sensor

For the Temperature sensor we went with the, TMP36GT9Z Temperature sensor. The other option being the 595-LMT01ELPGQ1 Temperature sensor. Both of the sensors were designed to work with 2.7-5.5 volts which works perfect for the output choices of 3.3 volts and 5 volts that the Arduino Yun offers as output voltages. The accuracy rating for the TMP36 is +/-1°C at 25°C (+/-1.8°F@77°F) while the 595-LMT has an accuracy rating of +/-0.125°C at 25°C (+/- .225°F@77°F). We decided that for a weather station the accuracy of +/-1°C would be fine for a weather station. So with the price being the final factor the TMP36 costs $1.39 and the 595-LMT costs $2.72. So with the only real difference being accuracy and not needed the higher accuracy that the 595-LMT sensor the lower priced TMP36 was the clear choice giving us a savings of $1.33.

C. Wind Sensors

The anemometer used was the US5881 hall-effect sensor. I first tried the circuit diagram from an online diy-hacking tutorial (Figure 18). The circuit was very simple, with only a 10k ohm resistor connect to +5V to the digital output. Looking at the pin layout on the US5881 data sheet, the sensor was tested on a breadboard. The output was connected to the Tektronix oscilloscope and a magnet was passed next to the sensor. There was no change in the signal.

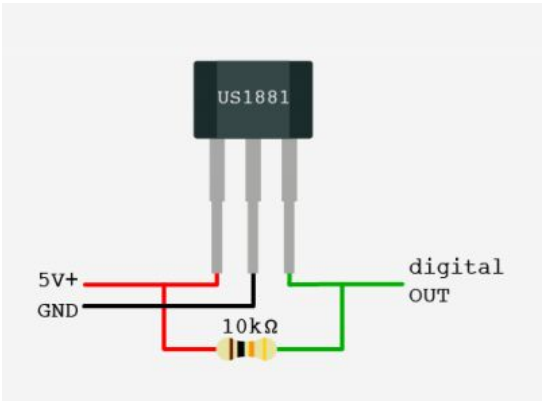

Figure 19. Hall-effect circuit diagram from diy-hacking.

The next attempt was to use the circuit diagram from the US5881 data sheet (Figure 20). The circuit was tested again on a breadboard with the output connected to a Tektronix oscilloscope. There circuit did not work. When the magnet was passed near the hall-effect sensor there was no change on the oscilloscope. When looking back at the data sheet I noticed that I looked at the wrong column for the pit layout. I mixed up the +5V and the output since there was two types of chips, a SE and UA package. This might have shorted the chip and destroyed it. When a new US5881 hall-effect sensor arrived, this proved to be true.

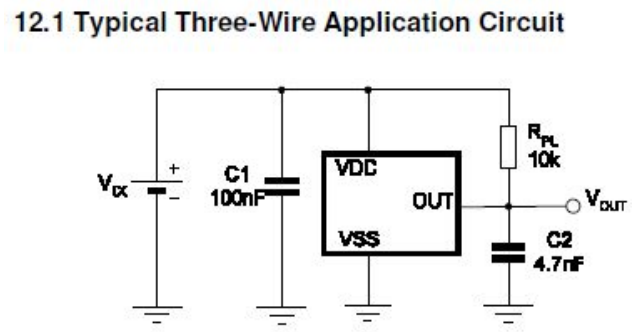

Figure 20. Hall-effect circuit diagram from the data sheet

The new US5881 hall-effect sensor was properly wired on the breadboard and double check to the pin layout on the data sheet. The output V_{out} was connected to the oscilloscope with a +5V connected to the VDC. The test was a success. When the magnet came close to the sensor the output voltage V_{out} drop to 0 Volts as shown in Figure 15.

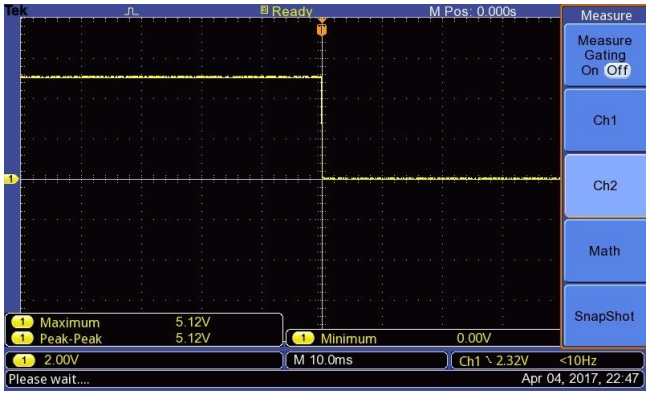

Figure 21. The Tektronix oscilloscope output of the hall-effect sensor.

Testing with the hall-effect sensor with the oscilloscope verified that the sensor works and operates to the data sheet specifications. The current draw of the part is 2mA with a 5 volt supply, which is a typical power draw as it is listed on the data sheet.

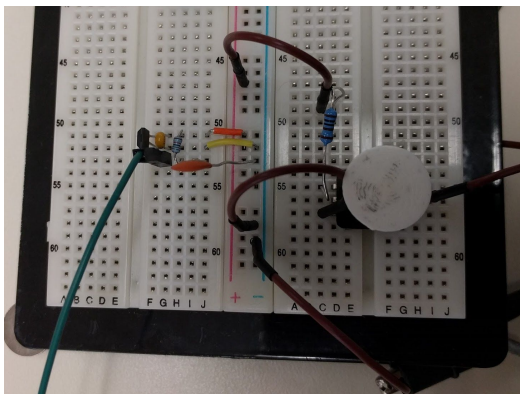

Figure 22. Breadboard testing of the hall-effect and position sensors.

The approach to build an anemometer with plastic egg cups had to be designed. Dustin at IEEE donated to the project with salvaged parts. Various shafts, bearings, and miscellaneous hardware was acquired. Plastic egg cups and wooden skewers were found at home.

The half hemispheres of the plastic egg cups would be attached to the wooden skewers. Then the wooden skewers would be attached to a 3D printed base (Figure 23). The base the with the cups would be inserted on top of a metal shaft with bearings. Another cylindrical base (Figure 24) would be printed in order to hold the magnet and would be inserted at the midpoint of the base .A semi-complete model without the cup hemispheres is shown in Figure 25. In Order to properly design the parts in Solidworks, measurements of the metal shaft, wooden skewers, and magnet were made using digital calipers. The complete models would then to taken to the Fablab in order to be printed with ABS plastic.

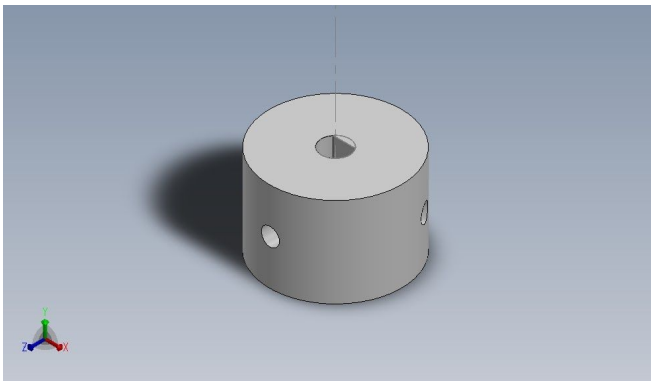

Figure 23. 3D Solidworks model of the anemometer base.

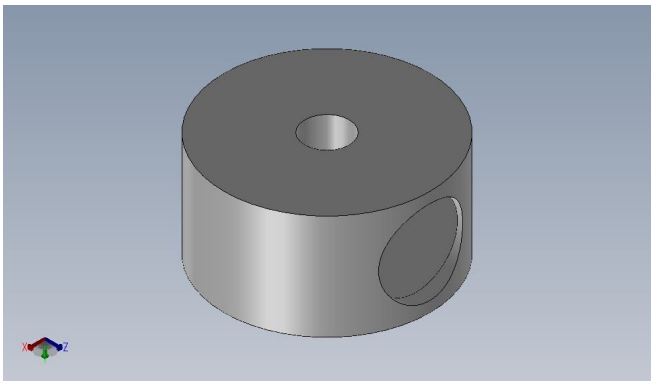

Figure 24. 3D Solidworks model to hold the magnet on the shaft.

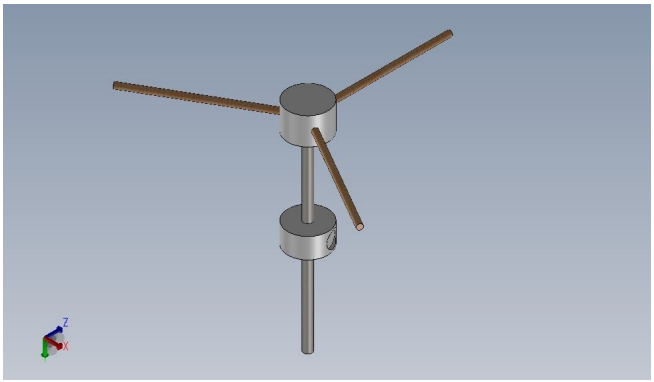

Figure 25. 3D Solidworks model of the partial assembly of the anemometer.

When the rotary encoder arrived from Mouser it was discovered that there were detent and a push switch. This was incompatible with our design. Another part had to be researched. Three different kinds of parts were ordered from Mouser. A rotary encoder without detents and push switch, a rotary encoder with a digital output, and a rotary position sensor that acts like a potentiometer (Figure 26).

Figure 26. Resistive Position Sensor.

The position sensor seem to be the best part. The part is essentially a potentiometer. A +5 volt source can be provided by the arduino and the output would vary between 0 to 5 volts. When the sensor completes a turn, the voltage will jump back up to 5 volts from 0 volts. The sensor would initially be be pointed north. As the wind vane changes direction, the vane will rotate the position sensor. Specified voltage ranges would be coded into the arduino, representing North, South, East, West. The output would then be connected to an analog input of the arduino.

III. Current Parts & Results

A. Wind Vane Design

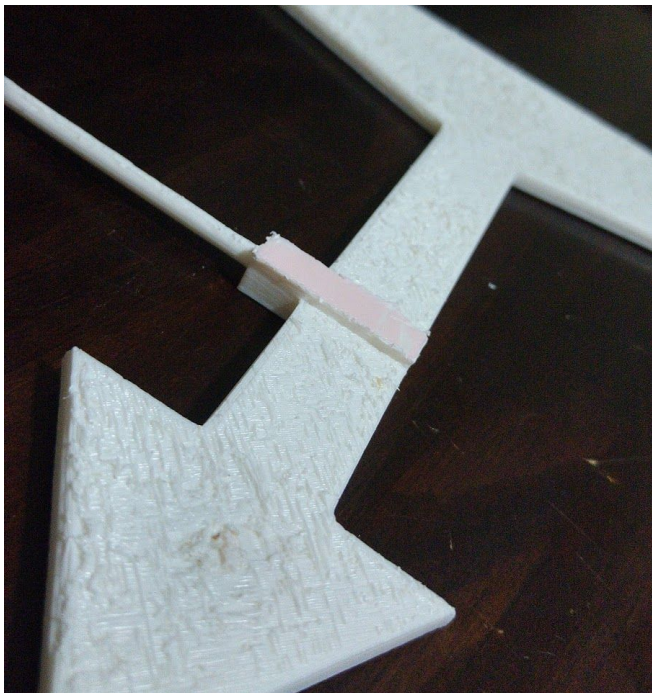

Figure 27. Wind vane printed and cleaned.

As shown in Figure 27 above, the 3D printed wind vane had to be cleaned from the support material that the printer used. This cleaning process involved sandpaper, a multi-tool knife, and clamps to separate the support material from the wind vane. If we had used the uPrint printers, then the support material would dissolve in water instead of using abrasives to separate the object so it would have been easy to remove and looked nicer. However, the regular printers cost $0.05 per gram of ABS plastic whereas the uPrint costs $1.00 per gram. Obviously, it is much more cost-effective to use the regular printers than the uPrint. Since it is not imperative for the wind vane to look perfect, using the normal printer is the optimal solution. Another problem experienced after printing the part is the tolerance of the rotary sensor adapter. Originally, this wasn't a problem since we ran a test print and the part fit perfectly on the rotary sensor. For some unexplainable reason, the print this time was too big and did not fit on the wind vane. For illustrative purposes, we have included a picture of the part of the wind vane we am talking about below in Figure 26.

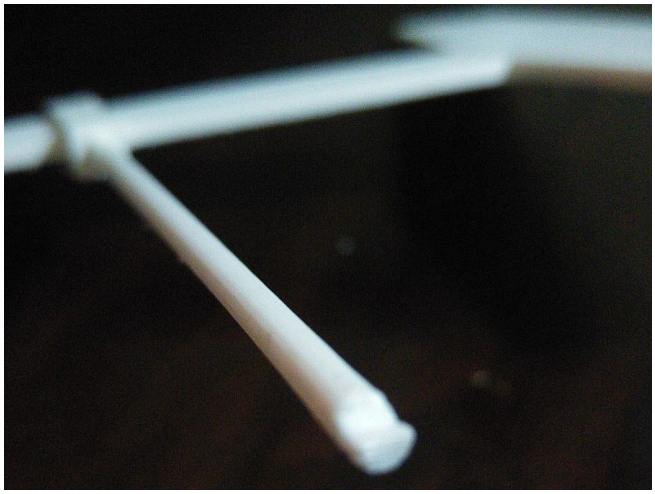

Figure 28. Wind vane rotary sensor adapter.

As seen above, the adapter does not fit on top of the rotary sensor. We used a multi-tool knife to trim the adapter until it was small enough to fit into the sensor. Unfortunately, we trimmed too much of the adapter and now the adapter is slightly smaller than the sensor. The goal was to remove just enough so that it was still a tight fit but now it is too small. In order to fix the problem, we will need to use a glue and/or adhesive putty to make sure the part stays in place. If that does not work, we can always cut off the entire adapter and use the adapter from the test run (the one that has a perfect fit). However, we will need to chop off the end of the wind vane and use tape and glue to join the two plastic pieces together. It is important to reinforce this section because the rod is under a big load due to the end of the wind vane cross area. Both solutions are acceptable to solve this problem and we will explore both possibilities after completing this second report.

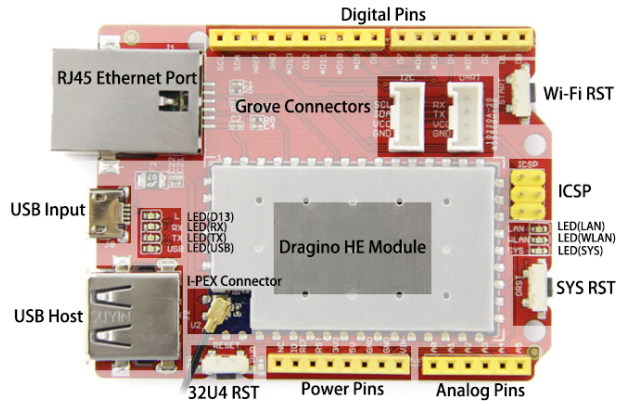

Figure 29. Seeeduino Cloud microcontroller layout.

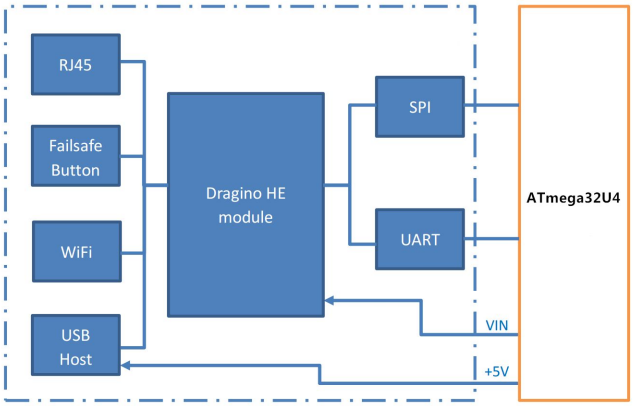

Figure 30. Dragino HE wifi module and ATMEGA32U4 microcontroller block diagram.

According to Figure 29, we have two Grove connectors on the board. Seeedstudio has sensors such as our barometer and UV sensors that can attach at these locations. A Grove sensor is a proprietary sensor offered by Seeedstudio so they have additional connectors for their parts on the board. This is helpful for us since we don't need to waste the analog and digital pins for the barometer and UV sensor since they can connect at these locations with their cable. This gives us more space to connect our LCD interface which uses quite a few of the pins. However, we still have sufficient space to connect the rest of the sensors with the LCD. One interesting property of this specific board is the USB Host feature. If for some reason a user is having internet connectivity problems, or prefers not to use the wifi feature, it is possible to save the data to a USB mass storage device. Moreover, if the user does not have access to a WiFi hotspot, then it is possible to use an RJ45 connection instead. The I-PEX connector is an optional connector and its use will likely vary according to each specific location. This connector is used to attach an external antenna and while not required, may fix poor internet connectivity connections.

As shown in Figure 30, we have a direct connection from the USB host to the microcontroller ATMEGA32U4 unlike the other blocks. This enables us to use the Arduino language to directly write/read USB mass storage devices instead of relying on the linux system on the Dragon HE module. The Wifi settings cannot be controlled directly through the Arduino language since the microcontroller has to use UART/SPI to communicate with these components. Essentially, the user has to connect to the open hotspot available when the system is turned on and open the web host on the device to change the connection settings. A disadvantage of the board layout is that the user cannot enter the wireless connectivity settings from the LCD interface when turning the system on due to the indirect communication between the microcontroller and Wifi module. A simple solution to this is to use a mobile device to open the specific web page and specify the connectivity settings.

B. Display

The display uses the I2C protocol to communicate with the arduino microcontroller. The two devices are connected using eight jumper wires. The pins used are as follows: 3.3Vcc, 5Vcc, SDA, SCL, AREF, GND, RST, and another GND. The communication protocol caused compatability problems between the barometer and the display, which could not be resolved. As a result, the barometer was not used. The software for the display was slimmed down to two submenus (current readings and settings), which allowed for more storage space to be used for other components.

C. Temperature, Pressure, Humidity and UV Sensors

The website for the Grove BMP280 had an arduino library which I uploaded into my program. Also I uploaded a Grove Sensor arduino library for various sensors made by Grove. The Grove BMP280 breakout board measures temperature and pressure and converts them to their digital value. The original units returned for the temperature is in celsius and the units returned are Pascals. For the code I set up functions to call the value for temperature and pressure then convert the units and return the values in fahrenheit and kPa. This device is connected to the Arduino Yun via the I2C grove connector.

The Humidity Sensor also came with a library. So from there I wrote a simple function that when called would return the value for the humidity. This part is connected to the arduino YUN via the DHT interface, which uses three wires: Vcc, Ground and Output (serial data).

The TMP36 didn't come with a library but was simple to write a program for the function that when called would return the temperature. This sensor has three wires Vcc, ground and output, with output usually being in millivolts. To find the temperature in celcius we use the conversion factor of 10 mV/°C then convert the value to fahrenheit.

The programming for the Grove UV sensor did not need a library. It was written as a function that when called

returns the value of the UV Index. The Grove UV breakout board has three connections. The first pin, input voltage is connected to 5 volts coming from the arduino. The second pin is a ground and is connected to the arduino ground. The last pin is connected to the A0 pin of the arduino. For the programming the voltage measured is then converted from a digital value to mV to be displayed and the UV Index values given for certain values of the voltage. The voltage levels for each value of the UV Index were provided by Grove UV Wiki page. Another way to estimate the UV Index value is to divide the Output voltage by 100 millivolts. Both ways to calculate the UV Index have closely mirrored the UV index reported by local weather stations.

The code written for each sensor was put into functions. The reason for this was for when Javier combined the code. This would make piecing it all together much easier, the functions created so that when he wanted a value for a sensor he would just have to implement the function to get it.

Figure 31. DHT11 Humidity Sensor

For the housing of the weather station we decided to use a birdhouse, so it could not only function as a weather station but perhaps as decoration in a backyard garden. We chose a 'build your own' type model so that it would be easy to modify to suit our needs.

Figure 32. Built your own birdhouse.

The first modification to the birdhouse was to cut a hole in the bottom to attach the power adapter as shown below.

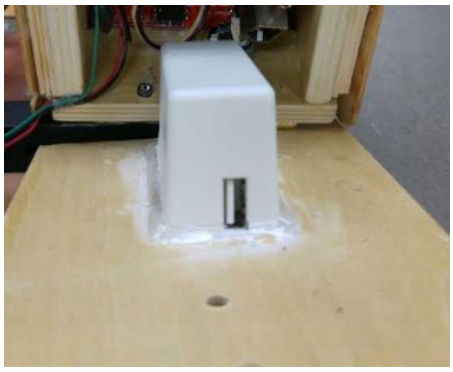

Figure 33. Power Adapter connected to bottom of birdhouse

The adapter was attached using gorilla glue and a silicon seal to keep out moisture. Another modification to the birdhouse was making a hole for the LCD display in the back of the birdhouse. This was also attached using gorilla glue.

Figure 34. The back of the birdhouse LCD display

And in the roof of the birdhouse a stabilizer for the wind vane was attached (a casing for a bic pen). As shown in the next figure as well the roof was decorated with popsicle sticks that were painted white, the color was chosen for the heat factor. Also for decoration and protection the birdhouse was clad with stained popsicle sticks on its walls. To make the insides of the weather station easy to get to, I designed the birdhouse to be able to be opened from the bottom using a small metal hinge. Also I attached the arm for the wind direction sensor to the back of the bird house.

Figure 35. The roof of birdhouse with weather vane and stabilizer attached.

To make the insides of the weather station easy to get to I designed the birdhouse to be able to be opened from the bottom using a small metal hinge. Also I attached the arm for the wind direction sensor to the back of the bird house. And a pole was attached to the bottom of the bird house for it to be put into the ground. The birdhouse worked out very well for our purposes, it was easy to modify and gave our electronic parts protection from the elements. A unique weather station that would look nice in a garden or backyard setting.

D. Wind Sensors

In the final design, the wind speed sensor used the cup anemometer with a hall-effect sensor and magnet. This design also has two parts that have been 3D printed at the Fablab with ABS plastic (Figure 36). The two pieces were test fitted with the metal shaft and wooden skewers. The 3D printed ABS Anemometer base fit well with the metal shaft and skewer. The 3D printed ABS magnet holder indent was too small to hold the magnet. Since the metal shaft had a tight fit into the magnet holder, the solution was to use the flatten end of the metal shaft to shave the access ABS plastic. This worked quite well as it saved time from reprinting and ensured a snug fit onto the shaft.

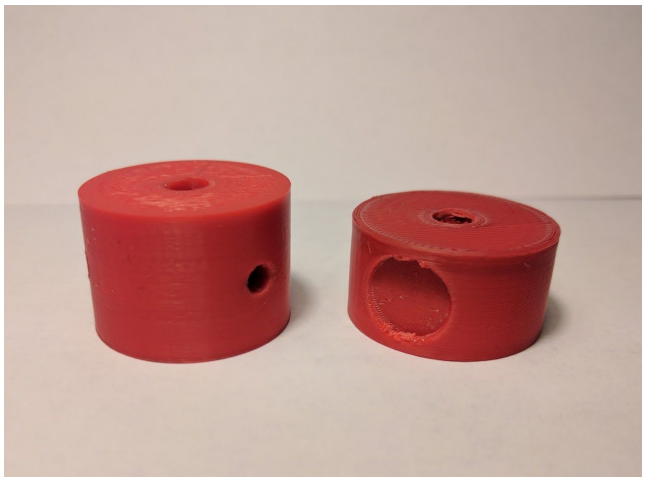

Figure 36. 3D printed Anemometer base and magnet holder

The code began with a example from diy-hacking.com. This example code worked with a latching hall-effect sensor and only measured RPM. The code was modified to work with a non-latching hall-effect sensor and code to convert RPM to MPH was developed. The RPM code worked with an interrupt. Every time the magnet passed the sensor, the interrupt would +1 to a revolution counter. Once five revolutions were counted with the millis() function. The millis() counts how many milliseconds since the program on the arduino has started. To find the change in time millis() was subtracted from timeold, which a previous state count on the millis() function. Refer to the code excerpt for the complete equation.

```
rpm=60*1000UL/(millis()-timeold)*full_revolutions
;
```

The "UL" flag is to make sure that math is done as an unsigned long integer. The rpm was then converted to miles per hour using dimensional analysis. The radius of the anemometer was measured to 4.3 inches. This was used to find the velocity of the air at that radius using this code excerpt.

```
mph = rpm*radius*2*pi*60/63360UL;
```

63360 is the number of inches to a mile. Mph was then return and uploaded to thinkspeak.

The wind direction which uses the resistive position sensor. Testing with the oscilloscope confirmed the specifications on the data sheet. The position sensor was attached to a 3-D printer wind vane. The arduino reads an analog signal and converts that to a direction. The user first sets the wind vane North to calibrate. The wind rotates the wind vane. The arduino takes in that resistance as an analog signal. The analog signal is ranges from 0 to 1023 steps, which then is mapped to 0 to 359 degree compass. The code then returns four directions, North, South, East, West, then uploads it to thinkspeak.

IV. Discussion

After completing the project, there are definitely some things which could have been done better. For one, we could have used more out-of-class project meetings earlier in the semester. This would have helped us to not squish a lot of meetings near the deadline. This is something that we learned from, and we will incorporate later for our senior design project. Another thing that we should have done but ran out of time was to cover the hole in the birdhouse with a clear plastic covering so that rain has a lesser chance of entering the birdhouse while still allowing sunrays through so that the ultraviolet sensor still works. While the birdhouse is glued together and is durable, it is likely possible that water might leak into the birdhouse. The final step is to cover the birdhouse in a waterproofing wood protector so that it can efficiently repel the water. Another change that would make it easier to mass produce the device would be to have the microcontroller and wifi chip on the protoboard so that everything is soldered. The current setup consists of the sensors and the related circuits connected to the protoboard so it would be more cost-effective and reduce the size of the device by putting everything in a protoboard. Better yet, using Eagle software, we can design a pcb board for a more professional product. On another note, I (Javier) personally believe we should get a bigger birdhouse and house the components in the "attic" so that birds can still utilize the house. Sadly, there is no space left in the current design for birds to use the house. On a different note, to bring this product to market we would need to make it easier for a user to handle. The code itself contains the channel information from the thingspeak server so this is inconvenient for the end user. If we simply create a web server that the linux dragon he module can handle then the user simply connects to the open wifi network and goes to a local website on the device to configure channel settings,wifi settings, and uploading rate settings as well. With that information saved, the microcontroller can access storage from the wifi module so that it knows what channel to send the data. Another feature of the device that requires configuration is the wind vane. From the LCD panel, the user can use the set feature to point the wind vane facing northwards and set the north direction. This allows the user to have the birdhouse face any direction instead of having it a requirement to face a certain way. These modifications would make the birdhouse weather station more user friendly and thus better for mass fabrication.

V. Conclusion

As shown above, everyone in the group has been participating to complete the project. Although there have been issues with the design approaches we originally chose, alternatives have been explored to ensure a quick recovery from the setbacks. This junior design project is a great tool in preparation for our senior design project. We now have a greater understanding of what is expected for the senior design project. Working on this project, we have realized that we need to schedule more out-of-class meetings earlier during the semester to maximize productivity and keep us on track. This is one aspect we plan to implement later for senior design. On another note, after writing three IEEE reports for this project, we have a better understanding of what is expected from the reports. This will surely help us write better reports later on in our EE courses and career. Also, we now know the product development cycle and how to implement it to design our project. We started brainstorming ideas that met the criterion requirements, researched possible solutions to implement in our design, developed solutions and implemented them, and tested the solutions to make a working product. The junior design lab helped us "exercise" these skillsets to have a better understanding of the design process. The experience gained from creating the weather station can be applied in future endeavours such as senior design so that we perform well by applying past experience to current situations.

VI. References

[1] (2017) SparkFun website. [Online]. Available: https://learn.sparkfun.com/tutorials/sik-experiment-guide-for-arduino---v32/experiment-7-reading-a-temperature-sensor

[2] (2017) The Seeed Wiki Page. [Online]. Available: http://wiki.seeed.cc/Seeeduino_Cloud/

[3] (2017) Wiki Seeed website. [Online]. Available: http://wiki.seeed.cc/Grove-Barometer_Sensor-BMP280/

[4] (2017) Arduino Playground website. [Online]. Available: http://www.newark.com/ist-innovative-sensor-technology/fs5-0-1l-195/thermal-mass-flow-sensor-gas-0/dp/52R8317

[5] (2017) DIY Hacking website. [Online]. Available: https://diyhacking.com/arduino-hall-effect-sensor-tutorial/

[6] (2017) Arduino Playground website. [Online]. Available: http://playground.arduino.cc/Main/RotaryEncoders

[7] (2017) Grove Barometer Sensor [Online]. Available: http://wiki.seeed.cc/Grove-Barometer_Sensor-BMP280/

[8] (2017) Grove UV Sensor [Online]. Available: http://wiki.seeed.cc/Grove-UV_Sensor/

[9] Bosch Sensortec, "BMP280: Digital Pressure Sensor,"BST-BMP280-DS001-12 datasheet, October 2015[Revision 1.15]

[10] (2017) Grove _ UV Sensor [Online]. Available: http://www.mouser.com/ds/2/744/Seeed_101020043-786558.pdf

[11] (2017) DHT 11 Humidity and Temperature Sensor [Online]. Available: http://www.geeetech.com/wiki/index.php/DHT_11_Humidity_%26_Temperature_Sensor

VII. Appendix