Stevens Institute of Technology

Final Technical Report

E-122-J

Group 3

“I pledge my honor that I have abided by the Stevens Honor System.”

Darren George

Derek Scott

Joseph Insalaco

Professor Andrew Isherwood

May 11, 2018

# Abstract:

This report analyses the Design 2 project, which consisted of building a functioning weather station and field testing the weather station for a week. The team’s design for this project used a WeMos board with a custom shield, an anemometer which was designed and 3-D printed, and various other pieces of hardware that were supplied. In addition, a GUI was created in LabView in order to allow easy access of the information that the weather station gathered. The design performed well over the week that it was tested, constantly uploading results when the battery was charged. The design withstood the elements very well and uploaded accurate temperature and humidity data for the entire week. However, the wind speed did not always register due to several hardware issues. In conclusion, the weather station worked well, but could have been improved with several design changes.

# Table of Contents:

|  |  |  |
| --- | --- | --- |
| Abstract |  | 1 |
| Table of Contents |  | 2 |
| Introduction |  | 3 |
| Discussion |  | 5 |
| Requirements |  | 5 |
| System Design |  | 6 |
| Mechanical Design |  | 8 |
| *Figure One* |  | 8 |
| *Figure Two* |  | 9 |
| *Figure Three* |  | 10 |
| *Figure Four* |  | 11 |
| *Figure Five* |  | 12 |
| *Figure Six* |  | 12 |
| Software Design and Coding |  | 13 |
| *Figure Seven* |  | 14 |
| *Figure Eight* |  | 14 |
| Electrical and Wiring Design |  | 15 |
| *Figure Nine* |  | 16 |
| *Figure Ten* |  | 16 |
| Final Evaluation |  | 17 |
| *Figure Eleven* |  | 18 |
| Conclusion and Recommendations |  | 19 |
| Appendix |  | 21 |
| Code |  | 21 |

# 

# Introduction:

The purpose of this report is to provide analysis on the Design 2 project that was completed over the course of the semester. This report will analyze the different aspects of the design, from the hardware to the GUI, as well as how the designs worked during the weather station’s week in the field. From this project, there are numerous learning objectives that can be taken away. Of these objectives, the most important and most relevant are the following:

* Design, test and operate systems with sensors
* Use software tools to collect, store, and clearly visualize data
* Select and use sensors to measure time varying physical quantity.

To sum up the learning objectives concisely, the team should create a sensor-driven system to measure the temperature, humidity, and wind speed over a given period as well as a user interface to store and interpret the data.

There were several requirements that guided the project. Firstly, the weather station must fit within a 12 inch cube. The anemometer should be able to accurately measure wind speeds from 5 mph to 25 mph. The anemometer will use a slotted wheel and encoder to measure wind speed in terms of pulses per second. The weather station should measure temperatures from 0F to 110F and survive wind gusts of 50 mph. The weather station will need to withstand 7 days of recording data outside. The electronics of the weather station should be housed in a 4” diameter, 6” long PVC pipe, with an included cap.

The weather station uses several steps in order to transmit the data it is recording to the LabView UI. First, the data is recorded and sent to the WeMos. The WeMos, which is connected to Wi-Fi, then uploads the data on a regular interval to a server. Labview then reads and stores the data from this server, which is then displayed to the user.

In the group, each team member had a specific role. Derek was in charge of the hardware. This included design and manufacturing. Darren was in charge of the software. This consisted of the LabView UI and the code for the WeMos. Joe was the project manager. This consisted of making sure the team was moving at an appropriate pace as well as consulting on the hardware and software designs.

# Discussion:

## Requirements:

The requirements for the weather station project were much simpler than the autonomous robot for Design I. Although the weather station looked much simpler it introduced several new elements such as the shield and WeMos, as well as interpreting data using LabView. In addition to recording simple weather data, there were several other design criteria the weather machine needed to include. These design requirements include:

1. Recording wind speed data.
2. Recording Temperature and humidity data.
3. Sending and publishing data to a server via WiFi.
4. Recording data and surviving in the field for seven days.
5. Assembly must be able to fit inside a 12”x12” box.

In addition to these criteria, a PVC pipe was also provided to house our electronic components. To help meet the design criteria, 3-D printing and metal tubes, rods, and brackets were provided by the lab. Although parts could be sourced from anywhere, parts sourced through the lab were sufficient for meeting the design criteria for the project.

## System Design:

In order to produce the most accurate and effective weather station, three different designs for weather stations were produced. These designs main features were two, three, and four arms. However, after quick research it was determined that 3 arms were the most effective at gathering accurate wind speed date. Another design decision made for the anemometer was the use of cups rather than paddles. Cups were chosen over paddles because the shape allows more force to be generated from the hollow side rather than the spherical side.

The final cup design featured an egg-shaped cup attached by metal rods, this design was a combination of conical and hemisphere shaped cups present in many designs. The final anemometer assembly used aluminum rods. The decision to use aluminum rods in the final assembly of the anemometer was to increase the strength of the anemometer assembly and increase the lifetime of the anemometer. The aluminum pipe was inserted into the PVC pipe by a bearing inserted into the cover of the PVC pipe. This pipe was then mounted to the side of the PVC by a metal bracket and the encoder wheel was attached by two shaft collars on the bottom.

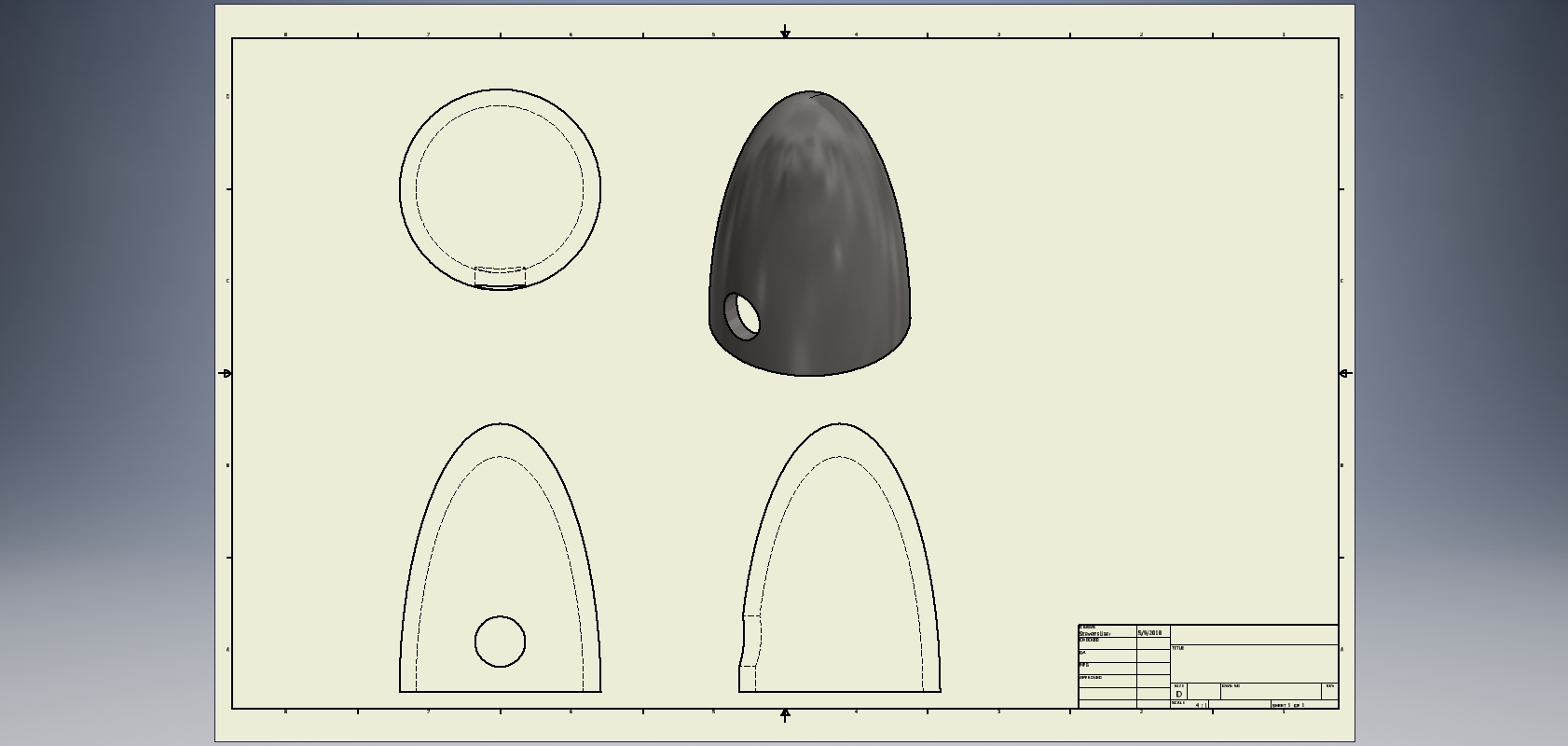
Design of the WeMos and electrical systems did not go through many changes as most of the wiring and code was provided to us. The team encountered some difficulty when the temperature and humidity sensor exploded. When the WeMos was plugged in for testing the temperature and humidity sensor exploded because of this. The WeMos also needed to be mounted in a way that the encoder wheel could be read by the sensor. It was decided that the WeMos could be mounted to the side of the PVC pipe by drilling holes in the side. This mounting system proved efficient and was only ineffective due to design flaws in the bracket. The battery was also mounted using double sided tape inside the PVC assembly. This was because the short battery life required frequent charging of the battery so constant removal was needed. Consideration was given for an easy to install and uninstall mounting system, and double-sided tape met both of these criteria.

## 

## Mechanical Design:

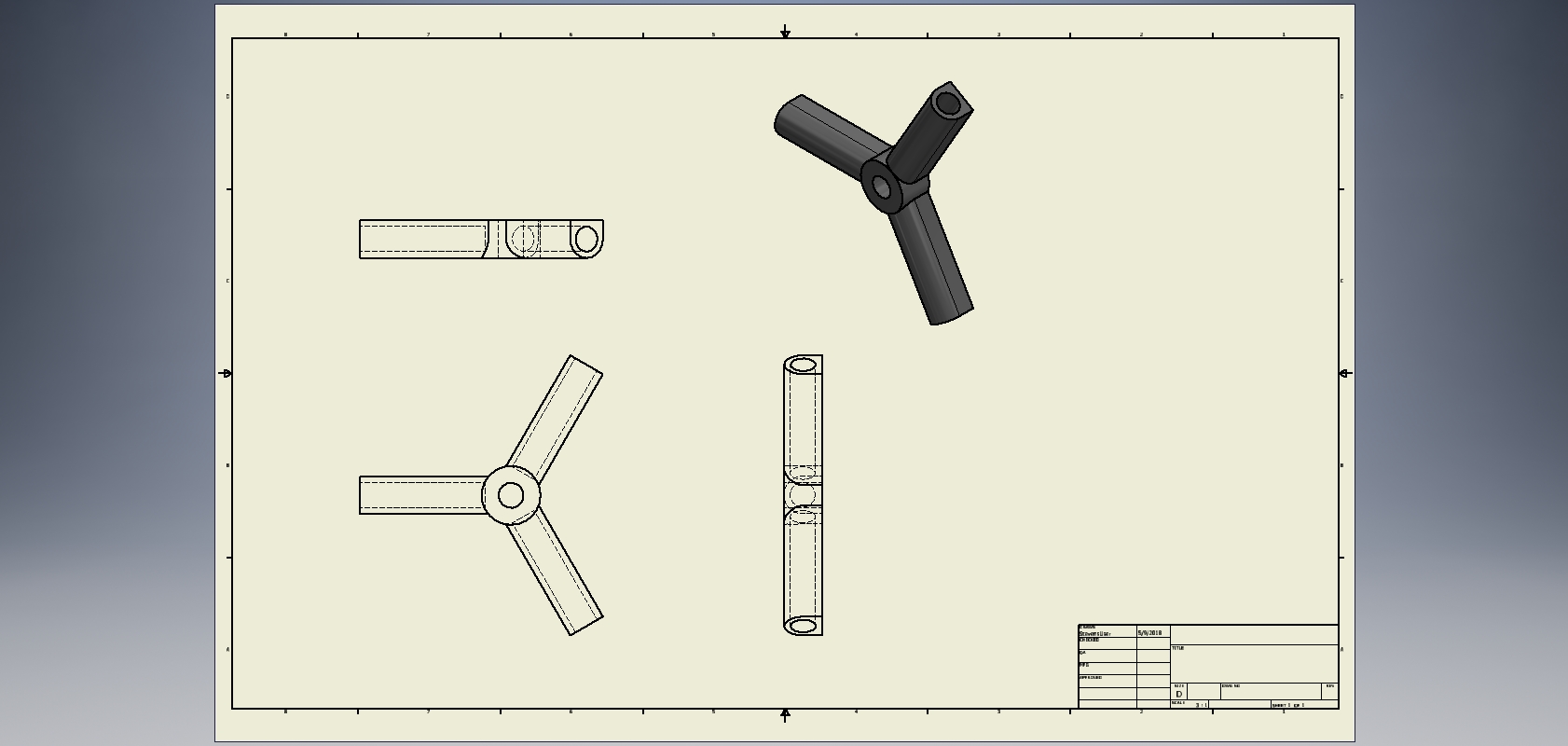
Most of the mechanical design in the weather station centered around the anemometer. Within the anemometer, there were two main parts; the anemometer design itself, and the mechanism that held the anemometer and allowed it to spin.

With the design of the anemometer, there were several main parts. One is the anemometer cups. These were 3-D printed in order to conform exactly to our design. A drawing of the CAD file is pictured below.

*Figure One.* 

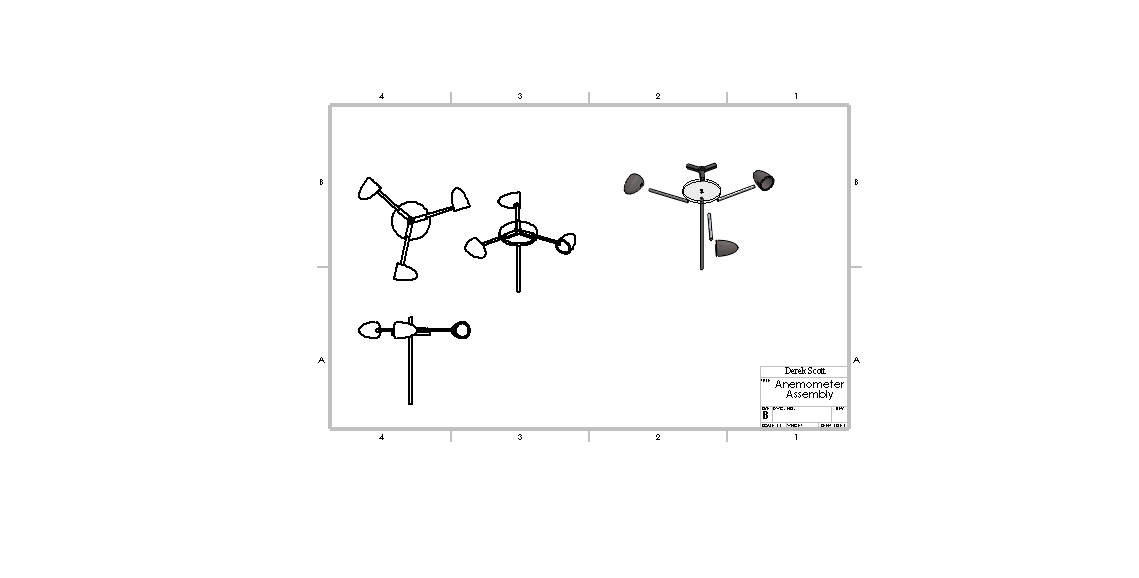
The anemometer cup was designed to have a somewhat spherical, cone shape. Ideally, this would maximize the air resistance in the front and minimize it in the back, creating the greatest possible differential force, propelling the anemometer forward. The anemometer cups would be attached by aluminum rods. The aluminum rods that were used in our design were about 5 inches long, making the anemometer fit well within the 12 inch square. The aluminum rods would then fit into another 3-D printed piece, which is pictured below. This piece would attach the three aluminum rods and anemometer cups securely to the main rod of the anemometer.

*Figure Two.*



This 3-D printed center piece performed very well in the anemometer. In the group’s original design, an acrylic plate was placed under the 3-D printed piece in order to support the weight of the anemometer rods and cups. However, the 3-D printed piece was found to be strong enough on its own to support the full weight of the anemometer cups and rods. Pictured below is the original CAD design of the anemometer assembly. The only difference from the final assembly is the acrylic plate.

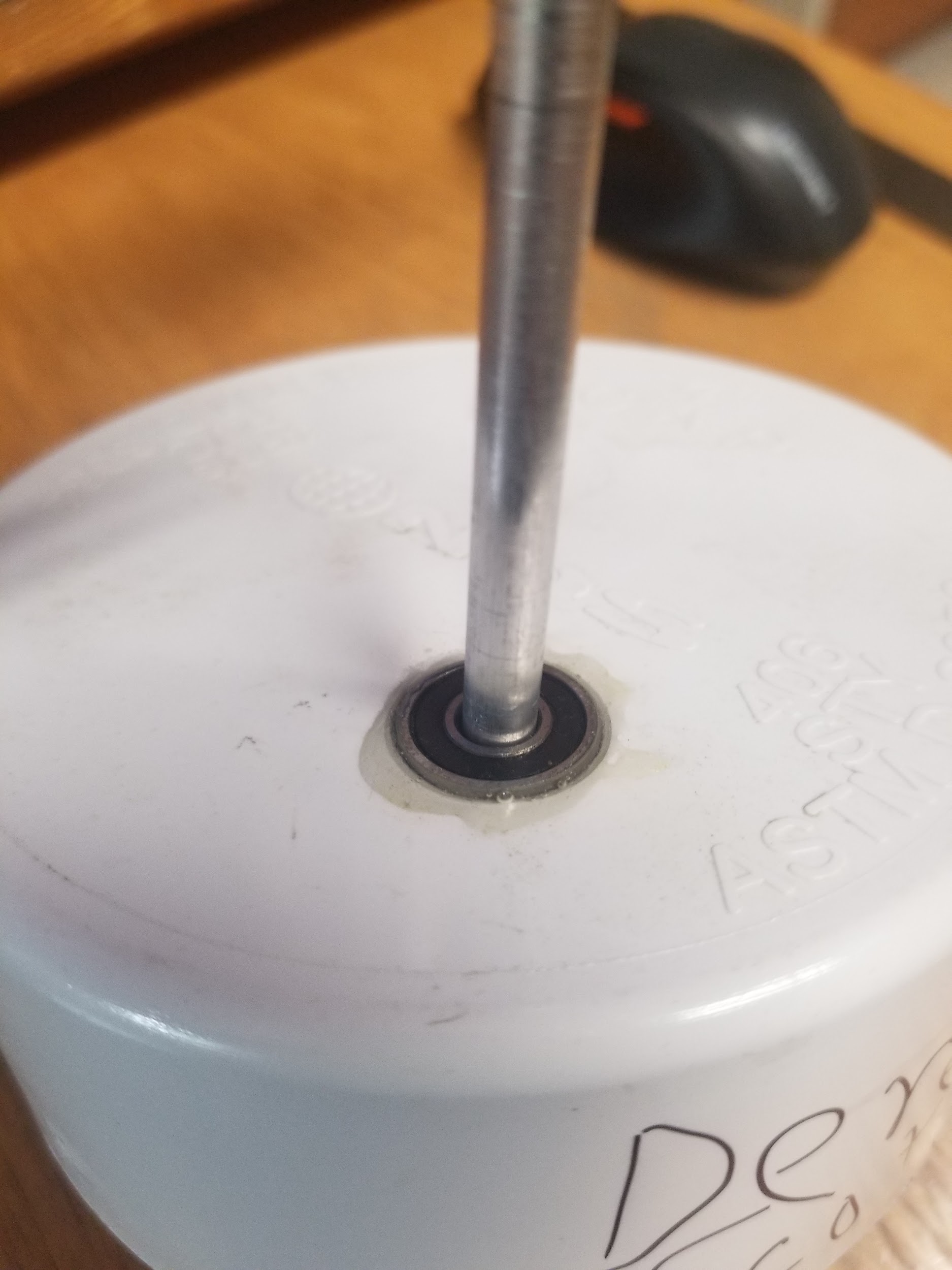
*Figure Three.*



The overall design of the anemometer was created collaboratively by the group. During one of the first classes, each group member drew their concept for the anemometer design. Then, the group compared each design and decided which elements to keep and which to discard in the final design. The final anemometer design was decided to have three spherical, cone shaped anemometer cups that would attach to the center spindle with a 3-D printed piece.

The second part of the mechanical design of the anemometer is the mounting mechanism. The anemometer touches the body of the weather station in two places; the bearing in the lid and the mounting plate inside the anemometer. The bearing in the cap has an internal diameter of ¼ inches and allows the anemometer to stay upright and spin without friction. The bearing is pictured below.

*Figure Four.*



The second place of contact is on the mounting mechanism inside the anemometer, which is pictured below. The mounted mechanism is screwed to the wall of the PVC pipe and is made up of a 90-degree bracket and a straight bracket with a hole drilled in the middle. The shaft of the anemometer runs through the hole in the straight bracket. The shaft is secured with two shaft collars that sandwich two bearings that squeeze the straight bracket. This allows for a tight attachment that still allows the shaft to spin freely. The encoder wheel was held on by an additional shaft collar that held it firmly in place.

*Figure Five. Figure Six.*



## 

## Software Design and Coding:

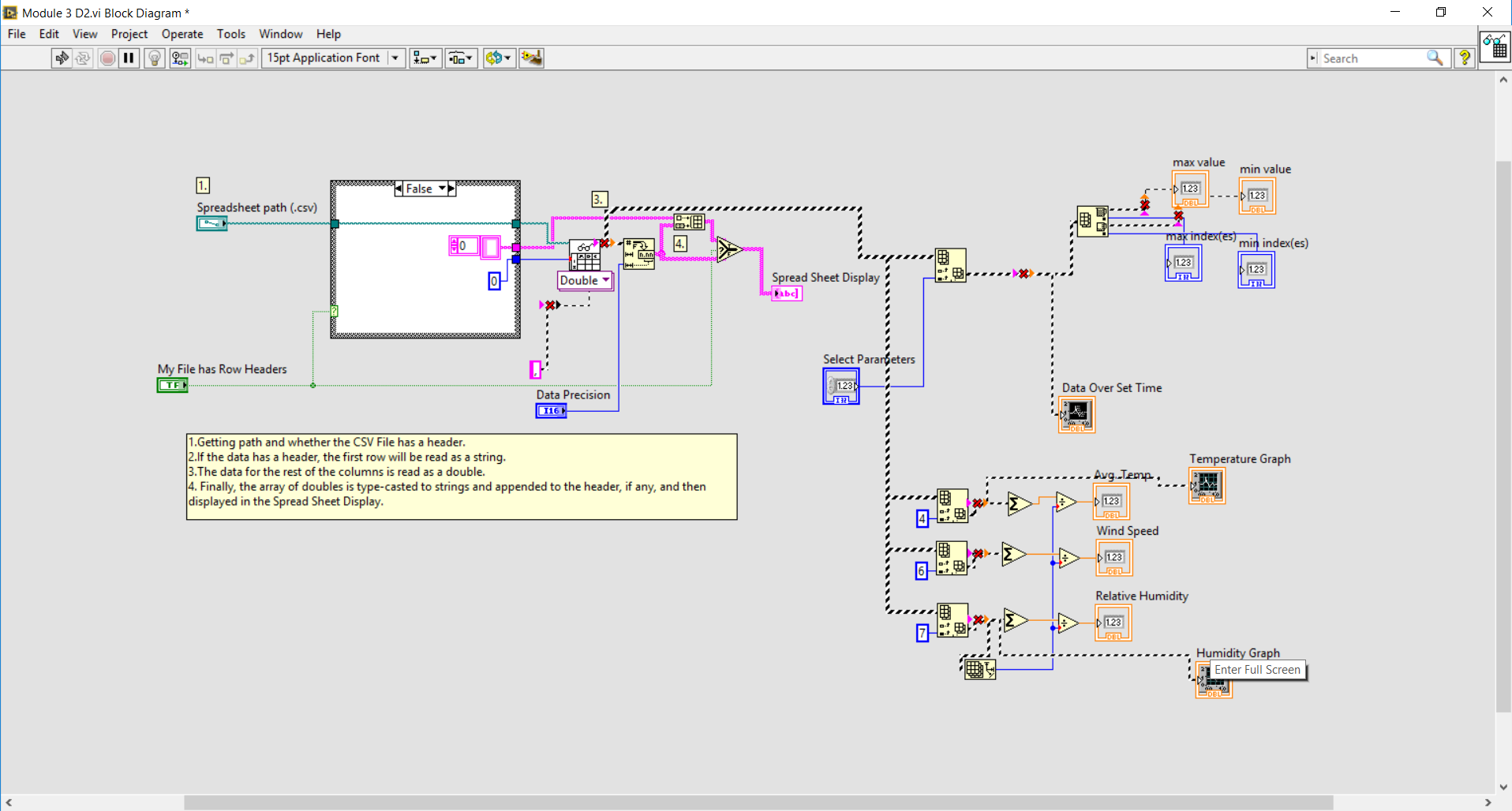
The software design consisted of using the Arduino IDE and LabVIEW. The IDE code was supplied by the instructor except for the sleep setup which we will go into later. LabView was constructed to be a user-friendly interface to assist in analyzing the data.

The WeMos code essential records data from the temperature sensor, humidity sensor (DHT11 ), and pulse counter sensor (OPB616 ) and uploads the data to the MQTT server. The MQTT server holds this data, and data is accessible through the web address: www.dmi.stevens.edu/mqtt/getdata.php?macid=989E

where the data can be collected in a .CSV file. Under the comment MQTT Settings is where the server username and password is to allow the data to access the server. In order to accomplish this, the WeMos must be connected to a WiFi network which it does under the void setup\_widi() function. After the setup function, the countPulses function counts the pulses, and the loop function after runs continuously after the setup is complete. The one component of the code we had to change was the nap time. Under the setup sleep comment, the group modified the int napTotal from 300 to 1. This allowed the WeMos to upload data every 12 seconds instead of every hour, allowing for easier diagnostics. However, it was found that the constant uploading of data drained the battery much more quickly than anticipated.

LabView was made as a user-friendly interface to help visualize the data collected. Labview is a graphical approach to programming which makes analyzing data easier. In the LabView program it was possible to take in a spreadsheet and break the data into different sections. The group made graphs for temperature, humidity, and a data over set time. The first two graphs display the data for temperature and humidity over a determined time, and the data over set time graph allowed the user to change what data we were looking at and change the amount of time the data was collected for. Overall, the user-friendly user interface allowed the group to effectively analyze the data and notice trends regarding the temperature, humidity, and wind speeds.

*Figure Seven, Figure Eight.*



## 

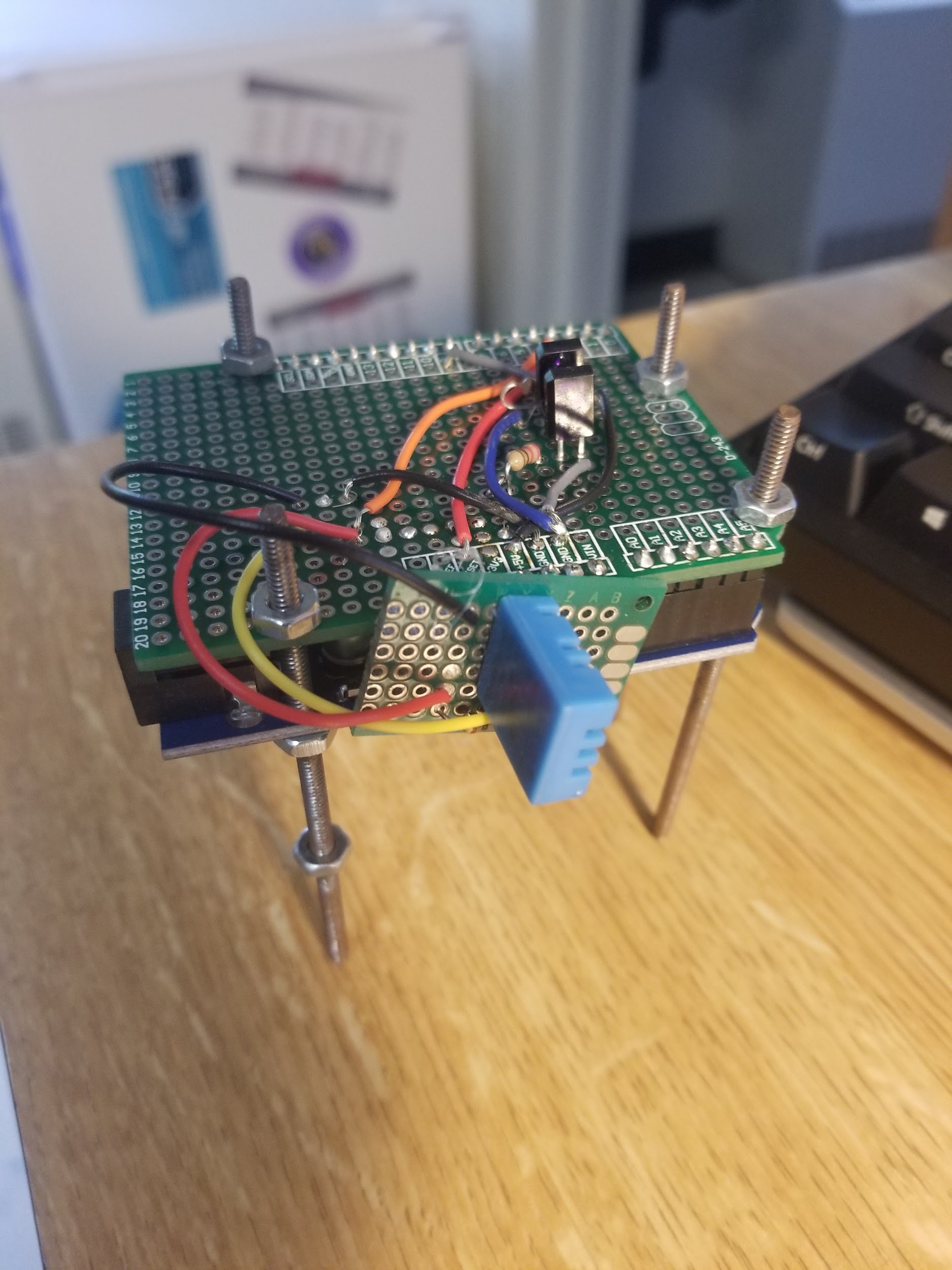
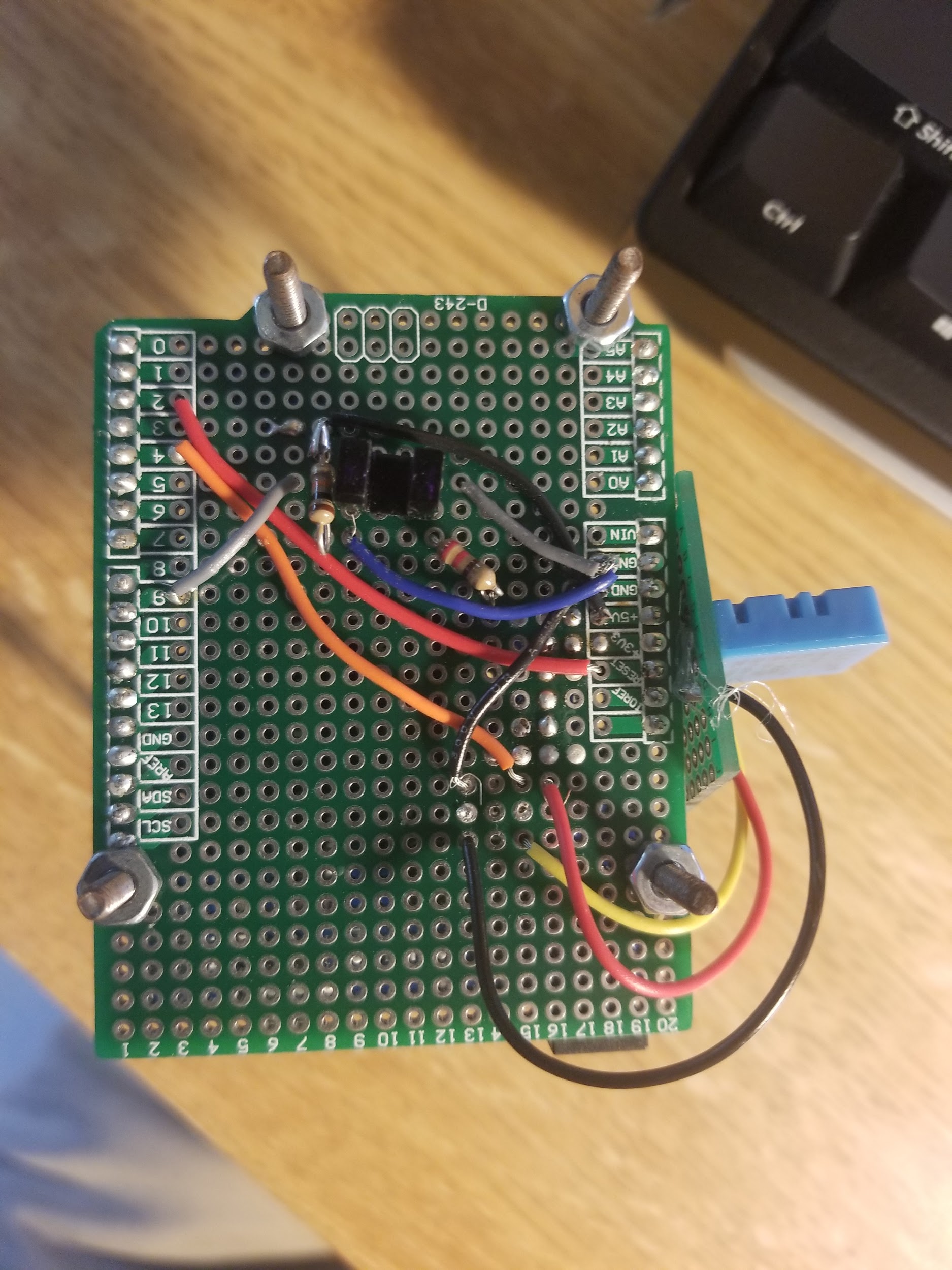
## Electrical and Wiring Design:

The WeMos Board is similar in design to the Arduino. they share many of the same capabilities. However, the WeMos microcontroller supports WiFi which is a critical function for retrieving weather data without storing it locally in the weather station. The WeMos board also has a similar pin layout to the Arduino and uses code for the Arduino.

The shield was designed by first soldering male headers to the protoboard. The male headers in the protoboard correlated to female inputs in the WeMos board. During the soldering, the team found it was difficult to align the male headers to plug into the WeMos and the male headers were manually bent in order to place the shield over the WeMos properly. Then, the DHT11 and photosensor were wired on a breadboard according to specifications outlined in module 2 activity 4 and 5. The shield was tested for shorts and plugged into the WeMos board to fully test. However, when the shield was plugged into the WeMos and powered on, the temperature and humidity sensor exploded. This was due to the connections to the sensor soldered backwards. The process to fix this error and rewire the sensor to the current design set the team back an hour, however the team was successfully able to rewire and move the humidity sensor to a different board. This allowed the photosensor and the temperature and humidity sensor to be separated and helped resolve interference issues with the encoder wheel touching the DHT11 sensor.

In the field there was little consideration to save battery power, according to estimates provided to us, the battery was expected to last a week in the field. Our team experienced a fraction of that time, with an average battery life of 24 hours. This was in part due to failing to adjust the time between taking data when the sensor was initially calibrated. However, the expected battery life for the device was around 2-3 days. Despite the battery lasting less than expected, the team was able to replace the battery and gather data in the field. Because of the way the WeMos was mounted, it was difficult to insert the USB and change the coding. This was the major reason the team did not attempt to fix battery life issues during fielding. Even with these issues, the team was able to collect a reasonable amount of weather data, and more care in the future will be taken to preserve battery life.

*Figure Nine. Figure Ten.*



## 

## 

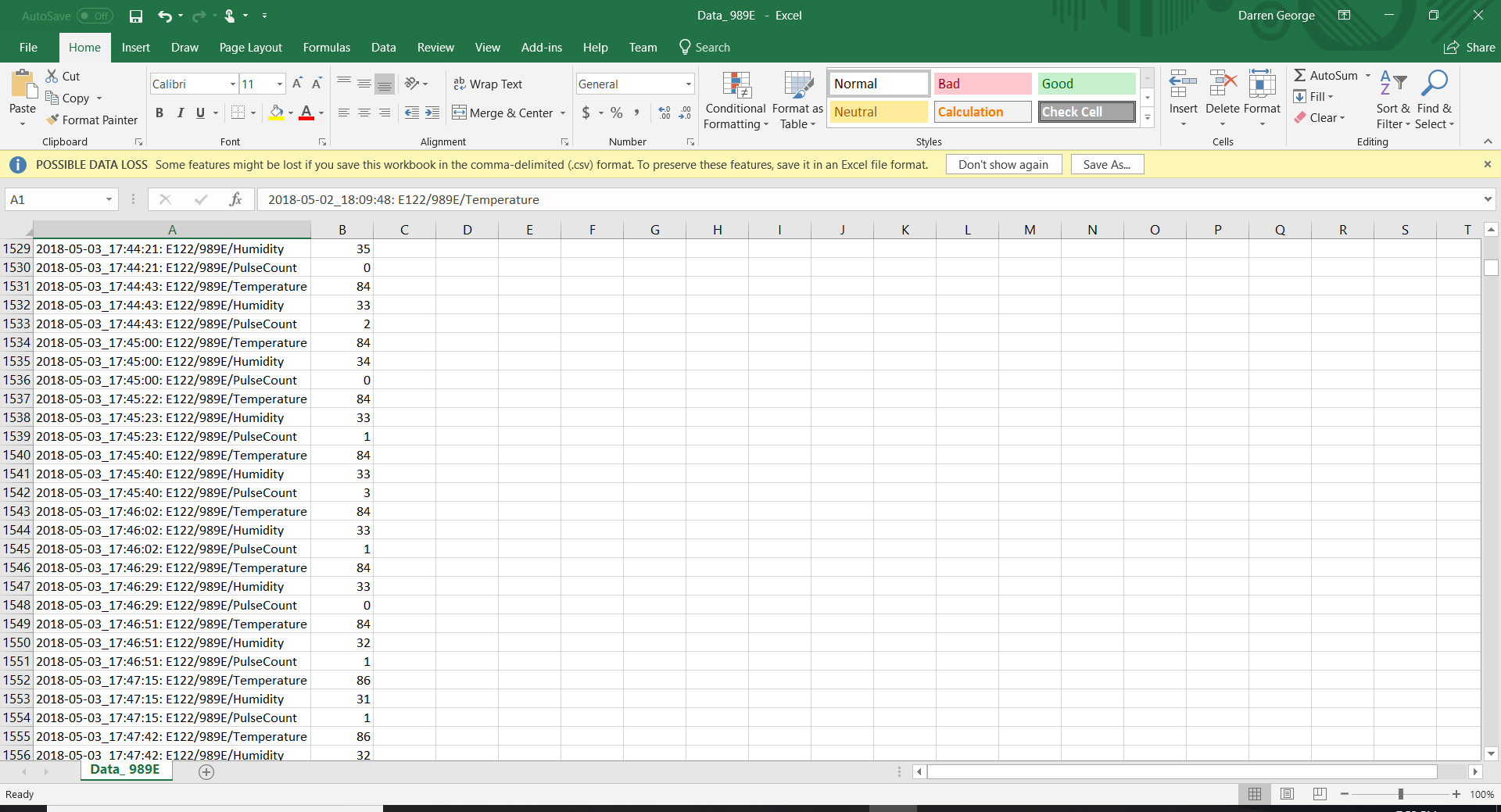
## Final Evaluation:

# The overall performance of the anemometer was successful. The group deployed on Thursday the 3rd and collected data for approximately 24 hours. This was because the battery died. The group was able to collect data for temperature, humidity, and wind speed which means the electrical design worked properly. During the time the anemometer was deployed, there was heavy rain and wind for several hours and the anemometer was able to withstand the harsh conditions. The group attributes this to the sturdiness of the design which proved to be very beneficial, however, it did pose a negative. Since aluminum rods were used to connect the cups to the central 3D printed piece, it made it slightly heavier than preferred. This caused the wind speed to not always pick up data when the wind was weak. Overall the performance of the design was successful, and although wind speed could have been picked up better with lighter materials, the sturdiness allowed it to endure the harsh weather conditions.

The data was stored in the MQTT server, and the group used the WeMos board’s specific MAC Address. The data was downloaded to an Excel file, so it was easiest to analyze on Excel itself. Soon after the anemometer was initially deployed, the wind speed was not recorded, so a group member checked out the anemometer and realized that the encoder wheel was slightly out of place for the DHT11 sensor. Using the Excel file was resourceful in allowing the group to quickly see the data and fix the issues. Although the group did use Excel to analyze the data, when the spreadsheet was uploaded into LabView, the group was easily able to view the graphs of the data. This was particularly useful in seeing how the anemometer withstood the harsh, uncaring weather that mother nature bestowed on us.

The group is proud of the design and the data that was collected during the fielding week. However, it would have been preferable to record a whole week of data, which could have been possible with a more robust battery. By collecting more data, a more precise conclusion could have been drawn.

*Figure Eleven.*



Of the data that was collected, there were numerous successes. Qualitatively, the data that was recorded matched with the weather that was felt for the day. For example, after the heavy rains on Friday night, the weather station recorded a drop-in temperature and a sharp spike in humidity that held until the following morning.

The tools that were used to analyze the data proved to be very useful. The Excel .CSV file was useful for a quick analysis of the data that was collected. The LabView was very useful for an in-depth analysis of the data, including graphing and viewing specific blocks of time. In all, Excel and LabView combined to give the group all the tools necessary to analyze the data.

# Conclusion and Recommendations:

The anemometer collected data; however, there were several small flaws in the design of the weather station that prevented more accurate data from being collected. While the weather and humidity sensor functioned and published data properly, the anemometer failed to collect all wind speed data. The malfunction of the anemometer can be directly attributed to the straight bracket mounting the anemometer to the inside of the pipe. Because of the design of the bracket, the aluminum rod shifted out of place and the encoder wheel did not pass through the sensor. Although the anemometer spun, it failed to record wind speed and therefore provided incomplete data. Improvements should be made to the anemometer assembly to make the bracket more stable or replacing it with a different mounting system entirely.

Another problem encountered was poor battery life. Although this problem was partially attributed to the over-collection of data points, the battery life in general was very poor. With regular polling of data, the battery lasted on average around 2-3 days. This was much shorter than the expected 7 days and in the future investing into a better battery than the one provided by the lab should be done for proper fielding of the weather station assembly.

In the future, many of these problems could be avoided by investing in a better battery and utilizing more lab hours. Some issues that arose during the building period were excess cancellation of lab hours. Although the weather station was successfully deployed, less time was available to 3-D print parts and fix errors that arose before deploying the weather station. In the future, extra lab hours should be taken more heavily into consideration and designs should be more planned out to avoid unforeseen problems such as the faulty bracket in our case.

During this project, the team learned several important lessons. First, the team learned the importance of correct wiring after witnessing the explosion of the temperature/humidity sensor. Another lesson learned was learning to properly manage time when faced with unexpected snow days. Several snow days in a row and cancellation of class put the lab a few weeks behind other sections. However, proper time management allowed for the successful deployment of all anemometers for the section on time. Finally, the last lesson learned is adapting your design. Although the anemometer failed to accurately measure wind speeds, changing designs will be considered for all future design projects. Although it may be challenging, the team should always be open to design changes even when the project is far along. Despite requiring extra work, changing the design may be the difference between a highly effective and ineffective project.

# Appendix:

## Code:

/\*

E122 - Wemos With MQTT and DHT11 - Version 0.1

Updated: February 20th, 2018. at 4:07 PM.

Integrated from components picked off the net by Prof.KP.

Check the configuration section for what you can tinker with.

Modify code on your own risk.... But, ... do take risks....

\*\*\*\*\*\*\*\*/

/\* Installation of drivers and other set up needed - Check CANVAS...

To install the ESP8266 board, (using Arduino 1.6.4+):

- Add the following 3rd party board manager under "File -> Preferences -> Additional Boards Manager URLs":

http://arduino.esp8266.com/stable/package\_esp8266com\_index.json

- Open the "Tools -> Board -> Board Manager" and click install for the ESP8266"

- Select your ESP8266 in "Tools -> Board"

\*/

#include <ESP8266WiFi.h>

#include "PubSubClient.h"

#include "info.h"

#include "DHT.h"

// CONFIGURATION SETTINGS ....BEGIN

extern "C" {

#include "user\_interface.h" // this is for the RTC memory read/write functions

}

//Setup Sleep

const unsigned long ONE\_SECOND = 1000 \* 1000;

const unsigned long ONE\_HOUR = 60 \* 60 \* ONE\_SECOND;

int quickNap=12000000;

int napTotal=1;

boolean firstTime = false;

unsigned long thisSleepTime;

typedef struct {

byte markerFlag;

byte counter;

unsigned long sleepTime;

} rtcStore \_\_attribute\_\_((aligned(4))); // not sure if the aligned stuff is necessary, some posts suggest it is for RTC memory?

rtcStore rtcMem;

void sleep1Hour() {

system\_rtc\_mem\_read(65, &rtcMem, sizeof(rtcMem));

printRtcMem();

// 126 in [0] is a marker to detect first use

if (rtcMem.markerFlag != 126) {

rtcMem.markerFlag = 126;

rtcMem.counter = 0;

rtcMem.sleepTime = quickNap;

firstTime = true;

} else {

rtcMem.counter += 1;

firstTime = false;

}

if (rtcMem.counter > napTotal) {

rtcMem.counter = 0;

}

thisSleepTime = rtcMem.sleepTime;

if (rtcMem.counter == 0) {

Serial.println("Rise and shine!");

return; // a day is up, go do some work

}

if (rtcMem.counter == napTotal) {

backToSleep(true); // next wake up has WiFi on

} else {

backToSleep();

}

}

void backToSleep() {

backToSleep(false);

}

void backToSleep(boolean wifiOn) {

printRtcMem();

system\_rtc\_mem\_write(65, &rtcMem, sizeof(rtcMem));

Serial.print("\*\*\* Up time: ");

Serial.print(millis());

if (wifiOn) {

Serial.println(", waking up... ");

ESP.deepSleep(50);

} else {

Serial.print(", deep sleeping for ");

Serial.print(quickNap);

Serial.print(" microseconds with WiFi on ");

Serial.print(wifiOn);

Serial.println("...");

ESP.deepSleep(quickNap, WAKE\_RF\_DEFAULT);

}

}

void printRtcMem() {

Serial.print("rtc marker: ");

Serial.print(rtcMem.markerFlag);

Serial.print(", counter: ");

Serial.print(rtcMem.counter);

Serial.print(", sleepTime: ");

Serial.print(rtcMem.sleepTime);

Serial.print(", thisSleepTime: ");

Serial.print(thisSleepTime);

Serial.print(", firstTime: ");

Serial.print(firstTime);

Serial.println();

}

//Wifi Settings

//const char\* ssid = "DLabsPrivate1";

//const char\* password = "L3tsM@keSometh1n";

const char\* ssid = "Stevens-Media";

const char\* password = "Stevens1870";

// ANY WEMOS WITH A STICKER ON THE BACK IS REGISTERED TO THE NETWORK...

// THOSE WITHOUT THE STICKERS DO NOT WORK...

//MQTT Settings

const char\* mqtt\_server = "155.246.18.226";

const char\* MQusername = "jojo";

const char\* MQpassword = "hereboy";

// This magic word is added to the topic for control messages E122/<MAC4>/Control/MAGIC\_WORD Param, Value

// Params Accepted PUBLISH\_DELAY VALUE in ms

// SLEEP\_DELAY VALUE in ms

//

const char\* MAGIC\_WORD ="XYZZYPQQRT";

#define PUBLISH\_DELAY 10000

#define SLEEP\_DELAY 2000

#define DEEP\_SLEEP\_SECONDS 10

//const char\* mqtt\_server = "broker.hivemq.com"; //This is a public server - no username/pwd

// Sleep delay is how long Wemos goes into sleep mode before re-checking --

// May help conserve power in the field -- you may have to check.

// Want to put it in Deepsleep while in field -else will burn through batteries. - power conserve mode...

// convert to microseconds

//ESP.deepSleep(sleepSeconds \* 1000000);

//Set up the DHT11

#define DHTPIN 4 //This is GPIO4 which is WeMos D2. Probably just hide this from students

#define DHTTYPE DHT11

DHT dht(DHTPIN,DHTTYPE);

//MQTT - 5000 ms is 5 sec.

//Setup pulsecounter

volatile unsigned long counter;

const unsigned long interval = 1000; // ms

#define signalPin 13

//GPIO13 in WeMos D1 R2 is D7

void eventISR (){

counter++;

}

//Interrupt call back function. This increments the global counter variable.

//

//Note that since this is a real Wemos board -- it runs forever as opposed to

// the fakemos -- http://www.dmi.stevens.edu/fakemos/

// CONFIGURATION SETTINGS ....END

WiFiClient espClient;

info board\_info;

PubSubClient client(espClient);

long lastMsg = 0;

char msg1[20],msg2[20],msg3[20];

char MQTopic1[50],MQTopic2[50],MQTopic3[50];

char ControlTopic[50];

char BoardMac4[5];

int value = 0;

float temp, hum,freq;

int tm = 0;

int dt,ta,ha;

unsigned long thiscount;

void setup\_wifi() {

delay(10);

// We start by connecting to a WiFi network

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

espClient.setTimeout(10);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

if(millis()>15000)backToSleep();

}

randomSeed(micros());

Serial.println("");

Serial.println("WiFi connected");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

// Board Last 4.

BoardMac4[0]=board\_info.mac()[12];

BoardMac4[1]=board\_info.mac()[13];

BoardMac4[2]=board\_info.mac()[15];

BoardMac4[3]=board\_info.mac()[16];

BoardMac4[4]='\0';

//

// Need the null termination for the string stuff to work, declare 1 extra character...

//

Serial.println(" MAC 4 of the board");

Serial.println(BoardMac4);

sprintf(MQTopic1,"E122/%4s/Temperature",BoardMac4);

sprintf(MQTopic2,"E122/%4s/Humidity",BoardMac4);

sprintf(MQTopic3,"E122/%4s/PulseCount",BoardMac4);

sprintf(ControlTopic,"E122/%4s/control/%s",BoardMac4,MAGIC\_WORD);

Serial.println("Setting the topics");

Serial.println(MQTopic1);

Serial.println(MQTopic2);

Serial.println("This Wemos Control channel is");

Serial.println(ControlTopic);

}

void callback(char\* topic, byte\* payload, unsigned int length) {

Serial.print("Message arrived [");

Serial.print(topic);

Serial.print("] ");

for (int i = 0; i < length; i++) {

Serial.print((char)payload[i]);

}

Serial.println();

// Switch on the LED if an 1 was received as first character

if ((char)payload[0] == '1') {

digitalWrite(BUILTIN\_LED, LOW); // Turn the LED on (Note that LOW is the voltage level

// but actually the LED is on; this is because

// it is acive low on the ESP-01)

} else {

digitalWrite(BUILTIN\_LED, HIGH); // Turn the LED off by making the voltage HIGH

}

}

void reconnect() {

// Loop until we're reconnected

while (!client.connected()) {

Serial.print("Attempting MQTT connection...");

// Create a random client ID

String clientId = "ESP8266Client-";

clientId += String(random(0xffff), HEX);

// Attempt to connect

if (client.connect(clientId.c\_str(),MQusername,MQpassword)) {

// if (client.connect(clientId.c\_str())) {

Serial.println("connected");

// Once connected, publish an announcement...

//client.publish(MQtopic1, "00000");

// ... and resubscribe ---- Dont subscribe KP

// Wemos subscribes to its control channel E122/<mac4>/control

client.subscribe(ControlTopic);

// #KP - No announcements --- No Subscribes.

}

else {

Serial.print("failed, rc=");

Serial.print(client.state());

Serial.println(" try again in 5 seconds");

// Wait 5 seconds before retrying

delay(5000);

}

}

}

unsigned long countPulses (int interval)

// interval defines

{

// reset set counter to zero

counter = 0;

// Attach the interrupt - count - detach

attachInterrupt (digitalPinToInterrupt(signalPin), eventISR, FALLING);

delay(interval);

detachInterrupt(1);

// Return the counter to the caller

return counter;

}

void setup() {

Serial.begin(115200);

sleep1Hour();

pinMode (signalPin, INPUT);

pinMode(BUILTIN\_LED, OUTPUT); // Initialize the BUILTIN\_LED pin as an output

delay(1000);

Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Serial.println("\* E 122 Design II: Field Sustainable Systems \*");

Serial.println("\* \*");

Serial.println("\* Wemos Firmware with MQTT and DHT11 Support \*");

Serial.println("\* \*");

Serial.println("\* Version 0.1 Feb 20 2018 (Stevens) \*");

Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Serial.print("Board Mac Address:");

Serial.println(board\_info.mac());

setup\_wifi();

dht.begin();

client.setServer(mqtt\_server, 1883);

client.setCallback(callback);

}

void loop() {

if (!client.connected()) {

reconnect();

}

client.loop();

thiscount = countPulses(1000);

temp = dht.readTemperature(true);

hum = dht.readHumidity();

snprintf (msg1, 20, "%d", (int) temp);

snprintf (msg2, 20, "%d", (int) hum);

snprintf (msg3, 20, "%d", (int) thiscount);

Serial.print("Published :" );

Serial.print(MQTopic1);

Serial.print(" with value: " );

Serial.println(msg1);

client.publish(MQTopic1, msg1);

Serial.print("Published :" );

Serial.print(MQTopic2);

Serial.print(" with value: " );

Serial.println(msg2);

client.publish(MQTopic2, msg2);

client.publish(MQTopic3,msg3);

Serial.print("Published :");

Serial.print(MQTopic3);

Serial.print(" with value: ");

Serial.println(msg3);

delay(1000);

backToSleep();

}