ELEC 391 Engineering Report:
Wind Tracking Power Generator

Submitted by:

Name	Student Number
Berk Calis	31602155
Bruno Rodriguez	19024173
Cyrus Cheung	77080844
Felipe Ballesteros	10703156

Submitted for:
ELEC 391
On Apr 14th, 2019
Professors:
Ignacio Galliano Zurbriggen &
Cristian Grecu

Abstract

This term, for our ELEC 391, Design Studio course, we set out to design a wind turbine that will generate power from a box fan provided to us. The requirement is that for our wind turbine is that is must meet dimensional requirement of 50x50x60cm and our boost converter must produce 10V - 15V output. Our goal for this project was to create an efficient turbine made with recycled materials. We settled on using legos for our chassis, since one of our members had previously owned them. We designed our turbine to be a 3-phase PMSG. Our generator will be equipped with a MPPT algorithm that controls duty cycle of our boost converter. With our design, we managed to provide 1.4 W of power with a 100Ω load. Our turbine is able to sense the wind and rotate our generator to track our wind with a 180° arc. To complete our system, we created a website, that has four main features: real-time monitoring, historical visualization, analytic services, and an alert system. With our real time monitoring system, we can see the current output of our generator, the data is stored in the cloud and past data can be retrieved and displayed on our website. We can also use our analytic services to process the data we are outputting. If our turbine is not operating correctly, we have an alert system that sends SMS alerts if the voltage, current, or power is either too high or too low. To ensure we minimize time debugging, we took step by step approach while integrating, so we can easily rule out conflicting modules. In total, we spent $379 of our budget, but we calculated with full efficiency, that our design could be replicated with $163. With our original timeline, roughly kept on schedule, we did however integrated our project later than expected. Finally, we discovered that we did not meet our goal to create an efficient turbine, since commercial turbines produce $1.75/W and our turbine produces $116/W. We did however design our design a very modular website that can be paired with any generator with an microcontroller capable of I2C communication.

1. Table of Contents

List of Figures

List of Tables

List of Abbreviations

MPPT	- Maximum Power Point Tracking
P&O	- Perturb and Observe
ADC	- Analog to Digital Converter
PMSG	- Permanent Magnet Synchronous Generator
I2C	- Inter-Integrated Circuit
PCB	- Prototype Circuit Board
O.P	- Operating Point
B.C	- Boost Converter
IC	- Integrated Circuit
MISO	- Master-in Slave-Out
IO	- In out
VM	- Virtual Machine
GCP	- Google Cloud Platform
SMS	- Short Message Service
HTTP	- HyperText Transfer Protocol
IDE	- Integrated Development Environment

Introduction

This year for ELEC 391, we designed a real small scale wind tracking power generator that can output up to 1.4W. Its design is composed of a custom designed PMSG worked on by Bruno, a custom diode bridge rectifier and boost converter on a PCB by Berk, fully designed Simulink control loops, operations and sensing ICs by Cyrus and a MISO arduino controller implemented with an MPPT algorithm and motor control by Felipe.

Our main goal for our project is to develop a small scale wind turbine that can be monitored in real time with software. With the real time monitoring software, we can ensure that our turbine is working properly, as well as display the history of the power extracted with the generator. We created this turbine to be created as economically and environmentally efficient as possible by clever optimizations on each respective part. The project allowed us to gain a proficiency on many open source software applications (SolidWorks, Altium, Simulink, Arduino IDE) and use them to create what we believe is a very efficient wind tracking power generator.

Project Description

Our wind turbine consists of a Permanent Magnet Synchronous Generator with 12 stator slots and 10 magnetic poles**.** The generator sits on a lego chassis which connects to a rectifier and a boost converter circuit controlled by an Arduino to produce 10V to 15V DC output and determines the point of max power. The turbine is equipped to measure generator current, voltage, boost converter output voltage and output power. With a load of 100Ω, we can output 1.4W.

For our ammeter, we used a differential amplifier followed by a voltage divider, and for our voltmeter, we used a voltage divider and a voltage follower. For wind tracking, we designed a PCB to fit a surface mounted rotary potentiometer, one on a free spinning blade to rotate with the wind, and the other mounted on our stepper .motor to provide proportional closed loop feedback.

Our wind tracking fin sits on top of the generator and we mounted our generator on a lego chassis connected which are with lego gears and a lego lazy susan with a PCB potentiometer to allow for feedback. The stepper turns the lego gears based off the wind tracker using proportional control with a total of 180 degrees..

On top of all the descriptions mentioned above, we decided to create a website to include a real time monitoring system to track the performance and health of the turbine. The features of the websites include: real-time monitoring, power history visualization, data analyzer, and a health alert system.

Goals

For this project, we set out to develop a working prototype of a PMSG wind turbine that can be monitored with real time with software from anywhere in the world. With the real time monitoring software, we can ensure that our turbine is working properly, and as well as display historical data such as monthly or yearly averages. We figured this is important when trying to determine the feasibility of planning a wind energy farm. In order to do that, we want to design our website to be as modular as possible which means the online module could be attached to any microcontroller that can interface with our arduino via I2C and we can display the data for that specific generator live in the cloud.

With our prototype, we can determine the feasibility of a renewable source of energy that is portable and can be placed in areas with no traditional power grid. Our main design goal is that we want this turbine to be created as economically and environmentally efficient as possible. This means using recycled parts, or parts we have in excess and trying to minimize our budget. We understand that this means the power electronics will not perfectly optimized and precise, but at the same time we do not require precision as what's important in a wind farm is the monthly power average.

As a secondary goal, we hope to achieve expertise with certain software design tools that will help us in industry later in our lives (ie. Use of SolidWorks, Arduino IDE, Altium PCB, Simulink, etc). Moreover, we hope to gain a level of familiarity with certain tools and machines whilst practicing their respective safety standards at all times.

Requirements

According to the specifications given, the following are the minimum required parameters for our generator:

1) It has to be a PMSG design
2) MPPT control
3) Wind Tracking
4) DC-DC Boost Converter from 10 - 15V
5) 50 x 50 x 60cm (W X L X H)

For our economical requirement, we were given a budget of $1000, with our grade dependent on our actual spending. In addition to our minimum requirement, we need a monitoring system that can show our output voltage, current, and power.

Design

For this section, we will split our design process into 5 parts, one for each of our respective roles in our group, as well as integration design.

Generator

The role of the generator design entails the following requirements.

1. PMSG design
2. 6V peak output voltage.
3. Sufficient output current to turn on MOSFET in boost converter circuit

For the design of the generator, we took 3 steps to achieve the requirements: research, iteration and test, and reiteration.

Research

Upon initial research of a basic PMSG design, we found that the main constraint is to have an even number of magnetic poles in order to be considered permanent magnet synchronous. The design of most wind turbines use around 6, 8, 10 or 12 magnetic poles depending on the size of the generator. Since all of these numbers are arbitrary and dependent on how much flux you can extrapolate from each magnet to one of the induced stator coils, there is no real effect on what we choose. In addition wind turbine technology tends to be fairly large but due to the constraints of a 50x50x60 cm dimension boundary for the project, the generator design needs to be fairly small but efficient.

Before we started the design process, we analyzed two sets of equations which determined a baseline for the generator's parameters described in Figure 1:

$$R' = \frac{1}{\mu_o} \frac{\hat{\ell}}{\hat{A}}$$

R' = an approximation to reluctance

$\hat{\ell}$ = an average flux-path length

$\hat{A}$ = an average cross-sectional area to the path

magnetic field strength = flux density

= flux per unit area

$$B = \frac{\phi}{A}$$

$$\boxed{\phi = BA}$$

Figure1: Equations for Reluctance and Magnetic Flux

To generate the most peak voltage at the output we had to analyze the magnetic circuit of the flux being induced for different reluctances. The main factors to deal with the minimization of reluctance was the length of the path and the area of the cross section of the stator tooth. Increasing cross sectional stator area would ultimately lead to higher induced voltage for bigger applications (ie. an actual size wind turbine), however decreasing the size of the path that the flux goes through has a similar effect and would be a more suitable model for cost efficiency for smaller applications such as this project. A graph of reluctance minimization depending on area and length is provided below.

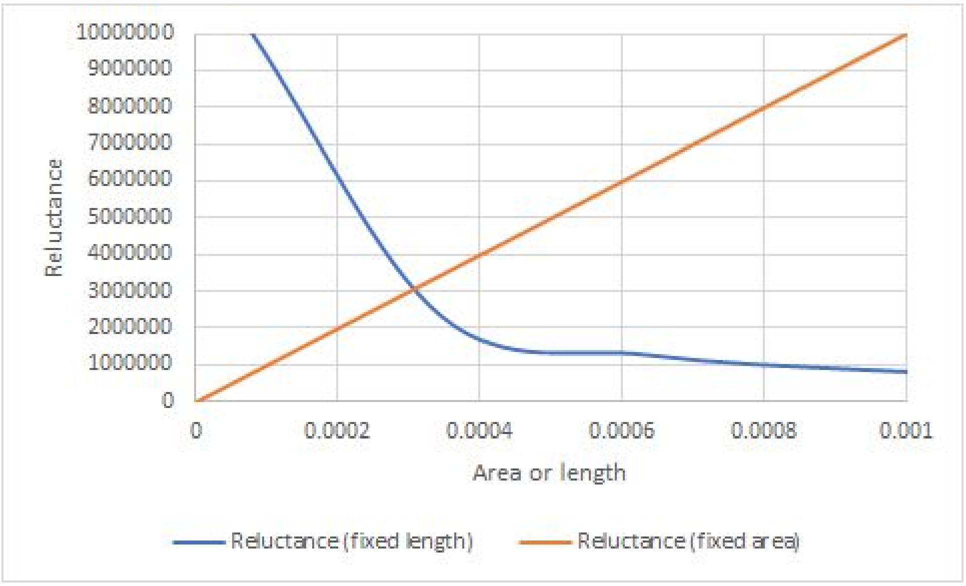


Figure 2: Reluctance (A*t/Wb) over Area or length (cm or cm^2)

In order to achieve a perfect reluctance minimization between an increase in area or decrease in length, the length of the path would have to be less than 0.0003m or 0.3 mm which is too small for a generator. In order to compensate for this, multiple stator teeth would be added for increased surface area and flux in addition to creating a relatively small and compact design in terms of path length to reduce the reluctance.

In terms of voltage and current generation, the amount of turns and gauge of wire plays a big factor. Higher gauge wire meant more turns could be added per stator tooth which leads to more voltage generation however it means that there would be less current due to the resistivity of the thinner wire and vice versa.

Iteration and test

Our first iteration of the PMSG design used 4 magnetic poles on the rotor and 6 stator teeth with around 150 turns per phase and 28 AWG wire. The generator is connected as a wye for simplicity of design. The stator casing was relatively small with a 5cm diameter, 4.5cm height and a cross-sectional area of 7.875 cm^2. This design, although functional, did not meet any real current or voltage requirements as it was 3D printed. Figure 3 shows the simulations of the voltage of the first design in the oscilloscope.

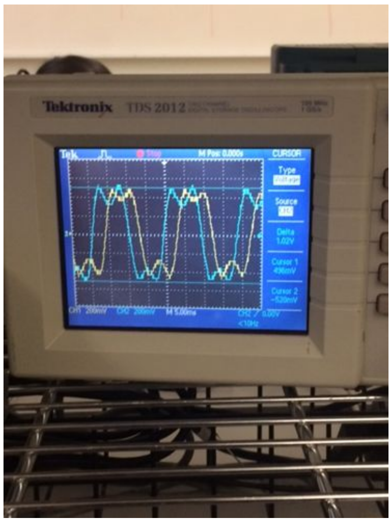


Figure 3: 2-phase Voltage for the first iteration of the PMSG

The harmonics appearing on the first simulation are a result of a mismatch in terms of reluctance which are due to but not limited to: mismatch in stator tooth area, mismatch in wire inductance, etc.

After testing 25 additional turns per phase (which was the most we could fit in the stator teeth for the current iteration), voltage increased by 0.2V at the output but resistance of each phase also increased which minimized the current output. After testing that design, 2 more designs were made in an attempt to increase the number of turns and magnetic flux.

Re-iteration and Finalization

After testing the initial design and knowing certain constraints about increasing the area of the stator cross-section and length of the stator, we ultimately decided on a slightly larger design with 12 stator teeth to increase the number of turns that could be added per phase. In addition to stator changes, the rotor was changed from a 4 pole to a 10 pole design which increases the effectiveness of each magnetic pole to each stator winding. Since there were more stator teeth, the effective area for magnetic flux increased and more turns with thicker wire were able to be added.

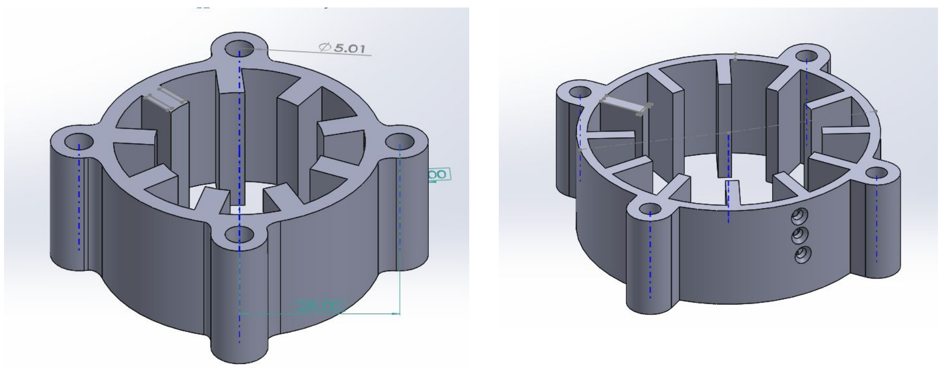


Figure 4: 9 tooth stator (50 mm diameter) and 12 tooth stator design (80 mm diameter)

For this new design, 300 turns were used per phase at 26 AWG wire to increase the induced current. Again the generator is a wye connected formation for simplicity. This version of the design is able to generate up to 3V peak with just a 3D printed design. Figure 4 shows the 12-slot stator/10 pole rotor’s line-line voltage with the fan.

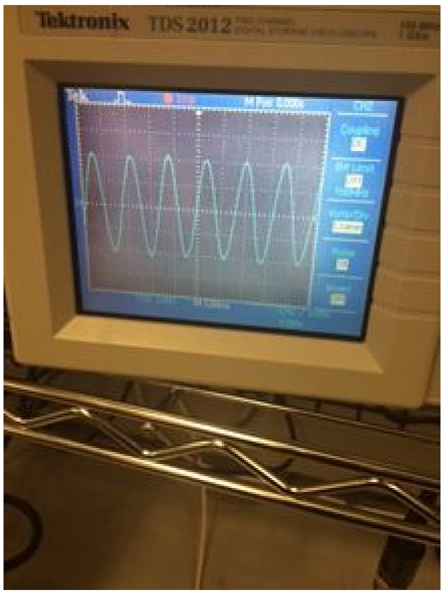


Figure 5: 2nd PMSG iteration output voltage

Since the permeability of the 3D printed design was just the permeability of free space which is $4\pi*10^{-7}$, the voltage and current were still relatively low outputs. However an increase of 50% was expected after water jetting an iron core for the stator which would increase the permeability which will reduce reluctance.

After waterjet cutting the stator and rewinding the 300 turns/phase, the output voltage of the PMSG with the blades and fan (**saving talking about fan design for integration area of report)** was around 7.02V peak with an output short circuit current of around 0.5 A. This finalized the development of the PMSG as it meets all the requirements for the project. Figure 5 shows the final version the PMSG.

Power Electronics

The role of power electronics expand on the following:

1. Converting AC Voltage from the generator to DC by implementing a diode bridge rectifier.
2. Boosting the converted DC voltage up to 10-15V level.
3. Optimizing power generating so that the chosen circuit components generate the maximum power output when integrated.

For our power electronics design process, we had 3 stages to complete: research and calculation, testing, optimizing, and finalizing.

Research & Calculation

With the research and calculation stage of our design, there were 3 main areas of focus: rectified voltage, current in the inductor and component choices. Rectified voltage is important because we wanted to maximize the AC/DC converted voltage output. To maximize DC voltage, we had to use diodes that had lowest possible forward voltage values while still being able to handle nominal current and power ratings. For this reason, we decided to use Schottky Diodes in our diode bridge and boost converter designs. The current in the inductor was also essential for our design because it is equal to the input current of the boost converter, thus directly influencing the integration of power electronics and generator. Lastly, component choices were integral for our design specifications because of their direct effect on the output voltage, output current, therefore changing the output power. Taking the design specifications into account, we came up with our component choices with the following calculations:

$$V in = 6V \quad V out = 12V \quad Duty\ Cycle = \left(1 - \frac{V in}{V out}\right) = 50\%$$

$$\Delta Il = 10\% \quad \Delta vo = 3.75\% \quad fsw = 10\ kHz \quad Rload = 80\ \Omega$$

$$I_L = \frac{Vout^2}{Ro \times Vin} = 0.30\ Amp$$

$$L = \frac{Vin \times D}{\Delta I_L \times f_{sw}} = 1mH$$

$$\Delta V_o = 0.075 * V_o = 0.9V$$

$$C = \frac{Vin \times D}{\Delta V_o \times (1 - D) \times R_o \times f_{sw}} = 100uF$$

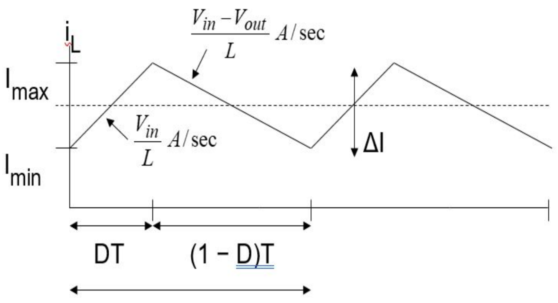


Figure 6. Average current in the inductor.

Testing

After deciding on the components for the design, we started simulating the design that we came up with, so that we could see if it would work if we were to implement it on a breadboard.

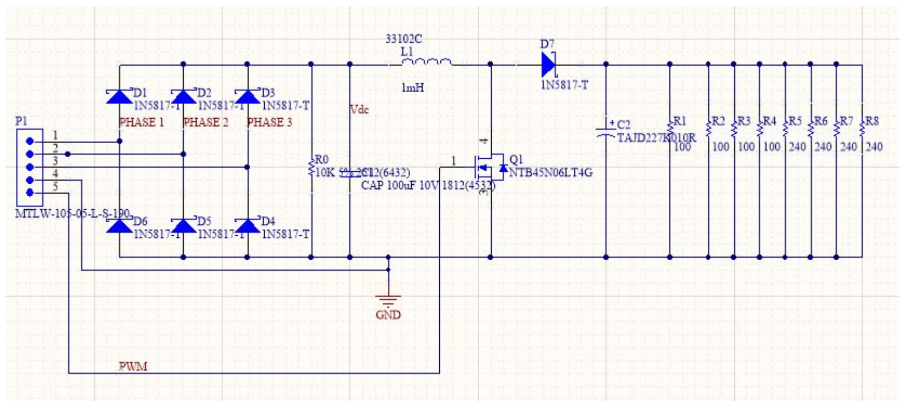


Figure 7. First simulation schematic

After seeing our simulation is working, we decided to try it on a breadboard to begin with. We divided our implementation into two parts: Diode bridge and boost converter. To avoid unnecessary reiterations, we emphasized on a step-by-step principle that allowed us to be on point when we were dealing with bugs or mistakes. Thus, we initially started the implementation of the diode bridge with a single phase full wave rectifier using 4 diodes. Once we made sure the rectification is how we want it to be, we wired up the circuit into a

3-phase diode half-bridge. For the 3-phase source, we used the test generator that came with the project kit. Just as our simulation showed, the diode bridge was working as it should be. We then move on to the harder part, the boost converter. For this part, the initials tests were conveyed by the use of the DC Voltage Supply as the Vin, and the Function Generator acting as the PWM at 10kHz and 5V which was connected at the gate of the fast switching power mosfet of our circuit. During the testing of boost converter, we realized that the calculated capacitor value of 100uF was not enough to filter the ripple, therefore we increased the capacitance to 220uF for the boost converter filtering even it it meant bigger voltage discharge and possibly higher current spikes. Although we made the boost converter work, the circuit needed 0.3 A to operate, which was a lot of current to expect from the generator. To make sure that it would not cause a problem for us during integration we increased the output load to 220 ohms so that the current requirement for the integration would drastically decrease.

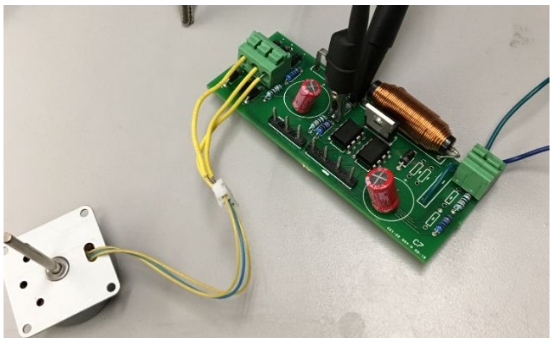
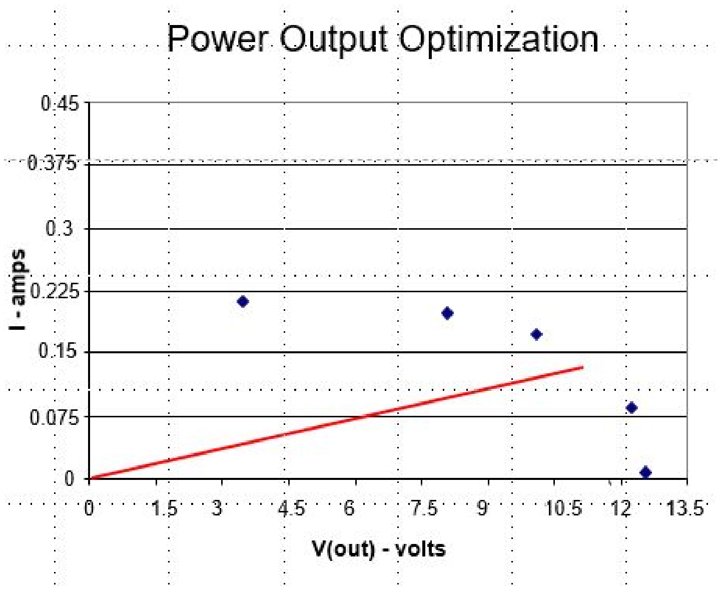


Figure 8. Demo-2 PCB Design post-testing.

Optimizing

After integrating both parts, we realized that the power output is not at a level that we desired it to be. Test generator was only operating at 0.04 Amps, and to provide that we had to use a load bigger than 220 Ohms. Before changing anything, we wanted to see how much current our generator could provide so that we could optimize the load for the best output power. The generator that we build was able to provide us more than 0.2 Amps, therefore we began our optimization. After trying many load resistances and logging them, we found the optimal value for the load resistor which gave us the maximum power output at 50% duty cycle.



$$\frac{11.5\text{V}}{0.13\text{ A}} = 89\,\Omega$$

Figure 9. Voltage vs Current Graph for Optimization.

Using 89 ohms was supposed to give 1.49 Watts power output for our design:however, we decided to go with a resistor we had an easier access for: 100 Ohms. With a 100 Ohms resistor, our output power was still very close to the maximum and it would also operate under a lower current, which would both make our design safer, and our voltage output higher. The output of our final design was 1.40 Watts.

Finalizing

With the optimization done, we went on to finalizing our design's power electronics part on a PCB. We did not want to have multiple PCBs due to the space they occupy in our structure and their high cost of manufacture. For this reason, we integrated the power electronics and the sensing together into a single compact PCB which saved us money and space. Here is the final design:

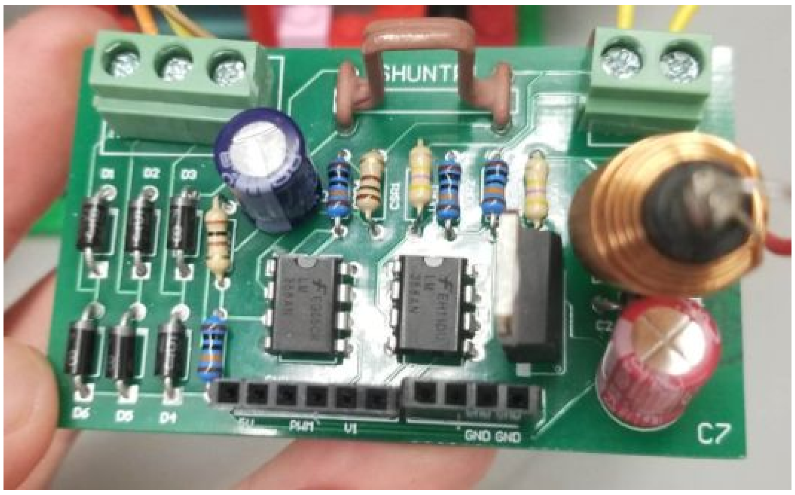


Figure 10. Final PCB Design

Sensing and Modeling

The role of sensing and modeling entails the following:

4. Voltage, Current sensing

5. Wind angle position, generator angle position
6. MPPT Models and sensor models

For our sensors' design process, we took four steps from inception to completion: research, modeling, prototyping, and finalizing.

Research

With the research stage of our design, we found that there were 2 main implementation of the current sensing: the hall effect sensor, and the shunt resistor. The hall effect sensor takes into effect the magnetic field produced with a current in a wire to produce an analog signal, while the shunt resistor produces a voltage drop across which is what is measured. We decided to go into with the shunt resistor due to its popularity and ease of implementation on a prototyping board. Initially we used a 0.01 mΩ, 2 terminal resistor, but after speaking with the TA, we discovered about 4 terminal resistor, which gives much more accurate readings. However a 4 terminal resistor is very hard to use while prototyping because they don't fit in traditional breadboards or perfboards. We decided to prototype the 2 terminal resistor but implement the four terminal resistor on a PCB. In addition we want to decide minimal power loss, so we wanted a small shunt resistor, but due the small voltage drop in a shunt resistor we need to amplify the signal. We decided on using a differential amplifier with a large gain to amplify the small signal. We used a voltage follower after the differential amplifier to isolate the circuit.

For our voltage sensors, there are not that many clever ways of achieving this, so we went with the simple voltage divider. We also used a voltage follower to isolate the circuit. A very important safety design we came up with was to supply the op-amps with 5V. With a 5V supply rail, if our generator output a much higher voltage such that the divider exceeded 5V, it would saturate, and the microcontroller would be safe. The equations for our voltage divider and differential amplifier is as follows:

$$V out = V in \frac{R2}{R1+R2} \text{ \& } V out = (V2 - V1)\frac{R2}{R1}$$

For our wind sensors and generator angle sensor, there were two main ways we could implement it with, the rotary encoder, and a rotary potentiometer. Initially we used a rotary encoder that pulses a logic 1 when the encoder rotates by 15 degrees. But we discussed that it would be difficult to implement this with feedback control, so we decided to use two identical potentiometers with their initial angle oriented in the exact same manner, that way we can compare their analog values to determine whether or not they are aligned. We decided to go with SMD rotary encoders as they are cheap, low friction, and can sit flat nicely on our legos.

Modeling

Following the research stage, we modeled our sensors and MPPT using CIRCUITMAKER2000 (SPICE) and simscape. To ensure that our sensors would work we modeled them in circuit maker as shown below in figure 11.

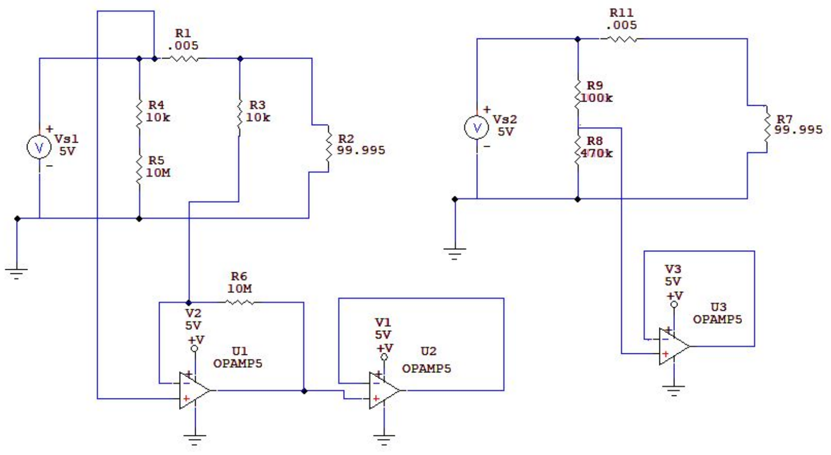


Figure 11. Ammeter and Voltmeter in CIRCUITMAKER2000.

We simulated this circuit with the DC O.P and found that values were exactly what we expected. We could then push this into prototyping.
For the boost converter and MPPT, we modeled our system using energy flow modeling (physical modeling) as shown below. We tried to model our system as close as possible to our real system, that is we added the voltage divider resistors and the shunt resistor.

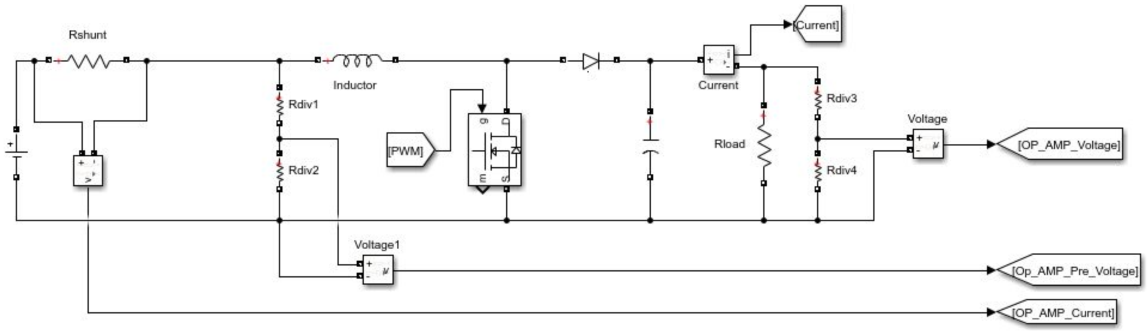


Figure 12. Simscape Model of our Boost Converter

To further simulate our real system we modeled the microcontroller. Every step that the microcontroller takes to convert our ammeter signals and voltmeter signals is taken to ensure that we can easily go from our model to our real system. For example, from the voltage divider, our microcontroller ADC takes in value that range from 0 to 1024 (10-bit). We modeled that and to convert from the voltage at our boost converter, we simply divide by 1024, and multiply by our voltage divider value which is 3 in this case.

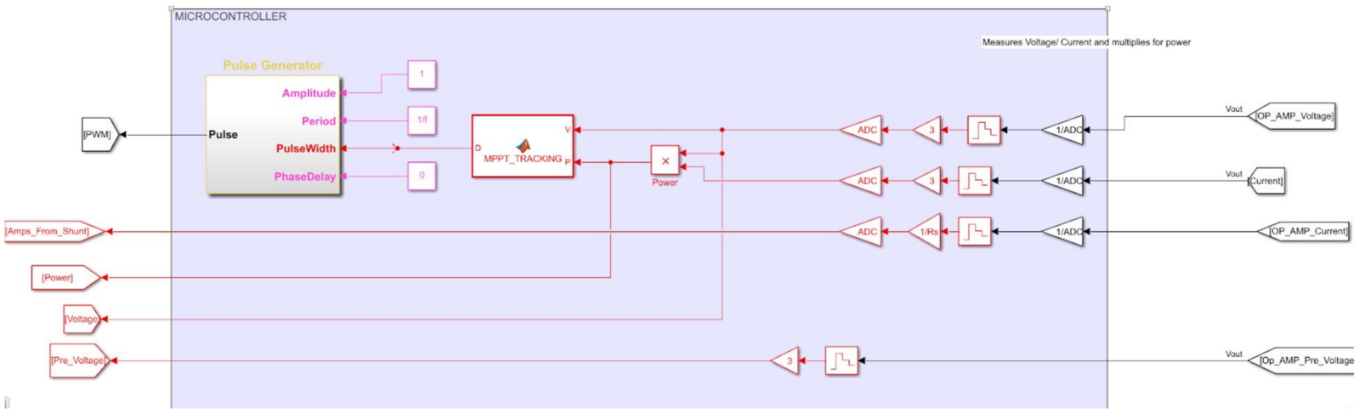
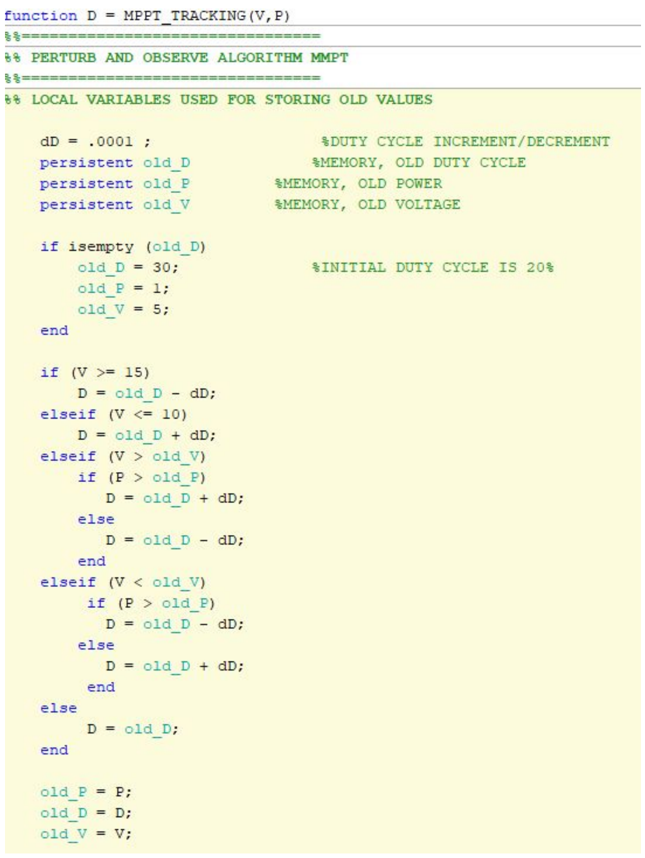


Figure 13. Model of our Microcontroller in Simscape

For the MPPT, we have chosen to use the perturb and observe algorithm. Upon researching MPPT algorithms, the P&O utilizes sensors we already have, thus is the cheapest to implement. Other algorithm require additional sensors, and will be more expensive to implement which oppose our design goal. The main drawback to using P&O is that the it the slowest algorithm. However this is not important to us as we feel that a difference of a few seconds is irrelevant since a wind turbine's most important parameter is it's monthly output power. With that reason we believe that it's not worth it to implement additional sensors to speed up our MPPT algorithm. For our MPPT modeling, we used a MATLAB function block. This block takes in the voltage from our boost converter and the power of our generator, and outputs a PWM. We coded this MATLAB block using the the MATLAB scripting language. This language is very similar to C code and can be easily implemented with minimal changes. Our code is shown below in figure 14. The advantage of using this type of modeling is that it is purely mechanical when converting from our simulated models to our real life system. It is very easy for anyone interpret and implement our sensors and our MPPT algorithm based off our model. One would only need to change the duty cycle implement, sample rate, and starting duty cycle. Our MPPT is shown working below in figure 16. Since we used a voltage source, we can see that the voltage is limited to 15V as specified in requirement. The limitation here is that since we are using an ideal battery source model in lieu of our generator model, the source can provide unlimited current which puts our power to unlimited if no bounds are specified. In reality, our generator will only output a finite amount of current, which will limit our MPPT to between 10V to 15V. In order to complete this model we require a exact working model of our generator, which we could not complete given the time frame.



```
function D = MPPT_TRACKING(V,P)
%%================================
%% PERTURB AND OBSERVE ALGORITHM MMPT
%%================================
%% LOCAL VARIABLES USED FOR STORING OLD VALUES

    dD = .0001 ;                    %DUTY CYCLE INCREMENT/DECREMENT
    persistent old_D               %MEMORY, OLD DUTY CYCLE
    persistent old_P           %MEMORY, OLD POWER
    persistent old_V           %MEMORY, OLD VOLTAGE

    if isempty (old_D)
        old_D = 30;                %INITIAL DUTY CYCLE IS 20%
        old_P = 1;
        old_V = 5;
    end

    if (V >= 15)
        D = old_D - dD;
    elseif (V <= 10)
        D = old_D + dD;
    elseif (V > old_V)
        if (P > old_P)
           D = old_D + dD;
        else
           D = old_D - dD;
        end
    elseif (V < old_V)
         if (P > old_P)
           D = old_D - dD;
        else
           D = old_D + dD;
         end
    else
         D = old_D;
    end

    old_P = P;
    old_D = D;
    old_V = V;
```

Figure 14. MPPT P&O Algorithm

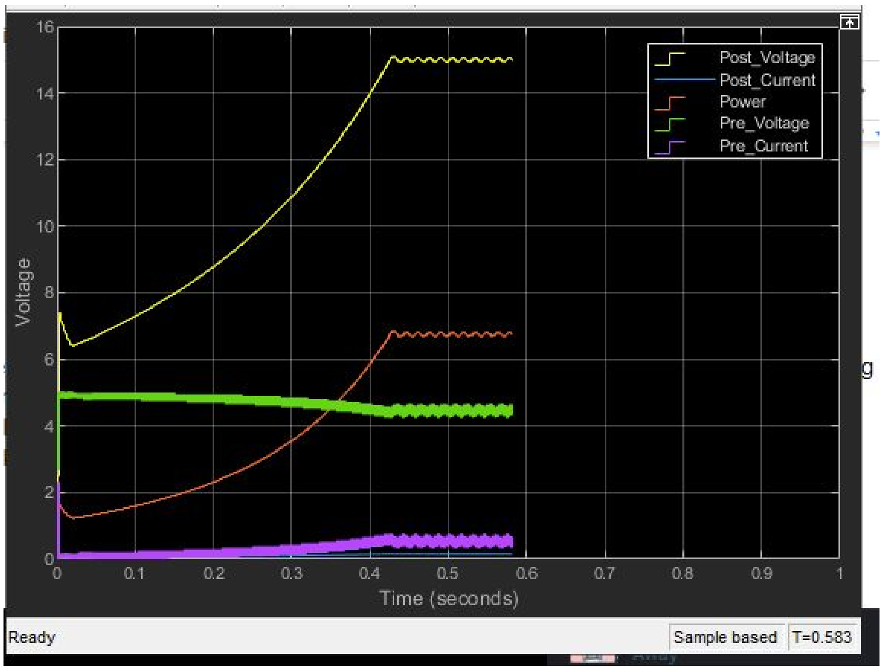


Figure 15. MPPT tracking maximum voltage given bounds of 10V - 15V

We also modeled our rotary encoder before we decided to use the potentiometer, it’s shown below:

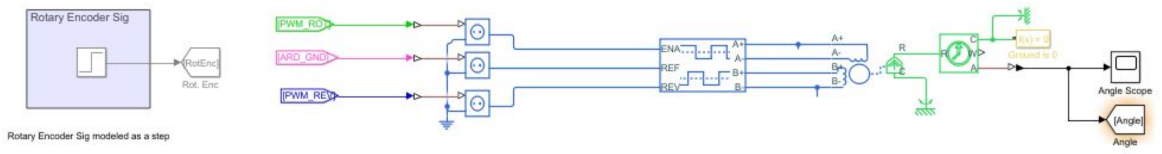


Figure 16. Rotary Encoder model, logic 1 defined as a step.

This model was created before our idea of using two potentiometers. Our model takes a unit step from our encoder pulse, and rotates the generator by 0.3 rads (15 degrees) as shown on Figure 18:

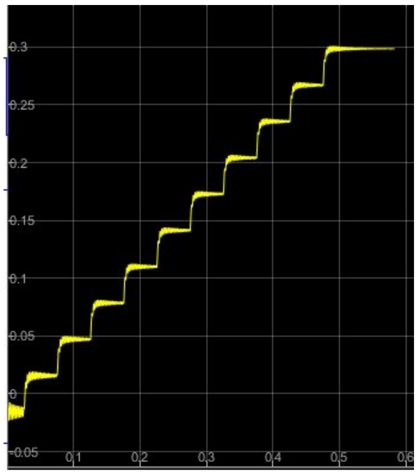


Figure 17. Waveform of Rotation of 15 degrees on our stepper motor.

Prototyping

With our modeling done, we can move on to prototyping. First we needed to select our op-amps. We wanted our microcontroller to power our op-amps as a safety measure just incase the voltage ever got too high. So we decided to use a single rail supply op-amp. Since we purchased elec kits from previous years, we had plenty of spare op-amps that were not used. In spirit of recycling we decided to use the LM358AN. First prototype was successful as shown in demo 1. As a reminder this is what our demo 1 circuit was, it's shown on the below in figure 19. In this prototype the 2 terminal resistor was used as mentioned above, it is a 0.01 Ω resistor coupled with a differential amplifier with a gain of 100. We used a voltage divider of 1/3rd (R_1=100kΩ and R_2=200kΩ) for our voltage divider. This circuit worked perfectly for demo 1. Since demo 1 is a proof of concept, we didn't exactly plan out what our nominal current and voltage would be so the prototype ammeter was rated for low currents, and the voltmeter went up to 15V as specified in the requirements.

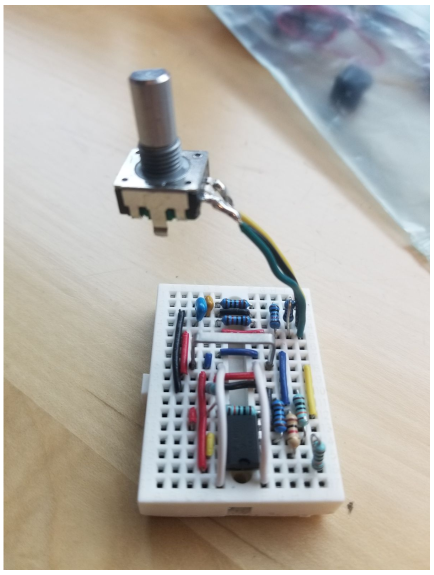


Figure 18. Demo 1 Circuit

Finalizing

With our prototyping finished, we went on to finalizing our design on a PCB. We drafting up a PCB design using Altium Designer and our design is shown in figure 19 below.

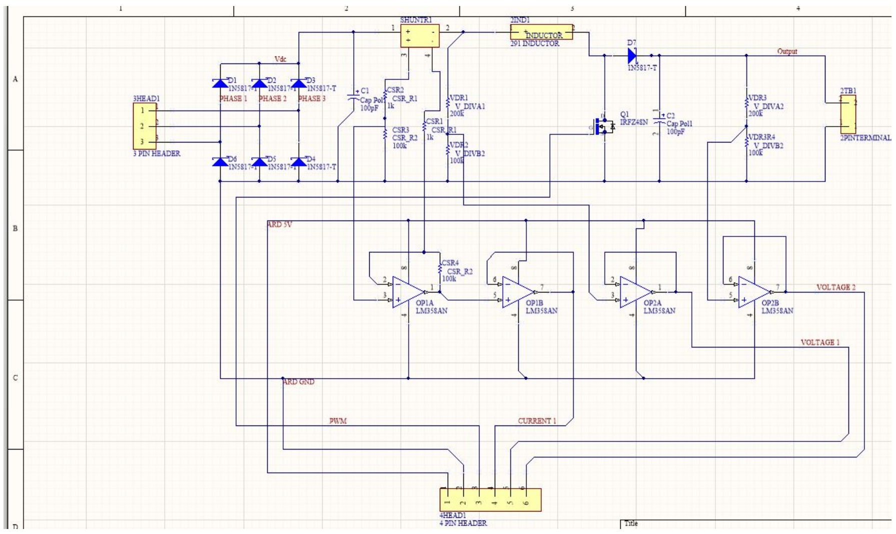


Figure 20. PCB schematic in Altium Designer 19

And the final PCB from that schematic is shown under the integration section at figure 23 page 24. In this PCB, we replaced our 2 terminal shunt resistor with a 0.005Ω 4 terminal shunt resistor. At this stage of the design we had a clear idea of what the output current of our generator was, so that we could design around it. We decided to set a nominal current of 1A. At 1A, we get a voltage drop, 0.005V, and we want to amplify it such that it's sufficient to read. We decided to use a gain of 1000 (R_2 = 100kΩ , R_1 = 100Ω), such that at nominal current we get a reading of 5V for our microcontroller. Realistically, our turbine only generated 1A but we wanted create our PCB so that it was modular and compatible with other generators. This means over-specifying our sensors such that they can be used on any generator that can supply up to 1A. For our final design, we also recalculated our voltage divider values. We found that when we approached 15V with the old setup, voltage output of our voltage follower were not accurate. We did some research and found that when the op amp is close to saturation and output is close to the supply voltage (0V or 5V), we get inaccurate readings. To remedy this we dropped our voltage even further with a ratio of R_1/R_2 of 4.7 (R_1 = 100k, R_2 = 470k). This dropped our voltage from 1/3 to 1/5.7. This gave us more accurate readings when we approached 15V. With this setup we can plot the specifications of our sensors in table 1 below. We assume the generator will operate at a nominal values of 5V at 1A, and that the post boost is 15V.

	Max before damage	Max before saturation	Min	Resolution	Power loss
Pre Boost Voltage	50V	28.5	0	+/- 30mV	43uW
Post Boost Voltage	50V	28.5	0	+/- 30mV	390uW
Current	23A	1A	0	+/- 5uA	5mW

Table 1. Specifications of our Sensors

And thus, our sensors meet the required specifications listed in the requirements section. The validation of our current and voltage sensors will be discussed below.

For our wind sensors, as discussed above we decided to go with a rotary potentiometer instead of a rotary encoder. We purchased a low friction SMD rotary encoder (3382G-1-104G). We picked this sensor because it provided us with a linear relationship, and it was small and portable such that it could be mounted anywhere.

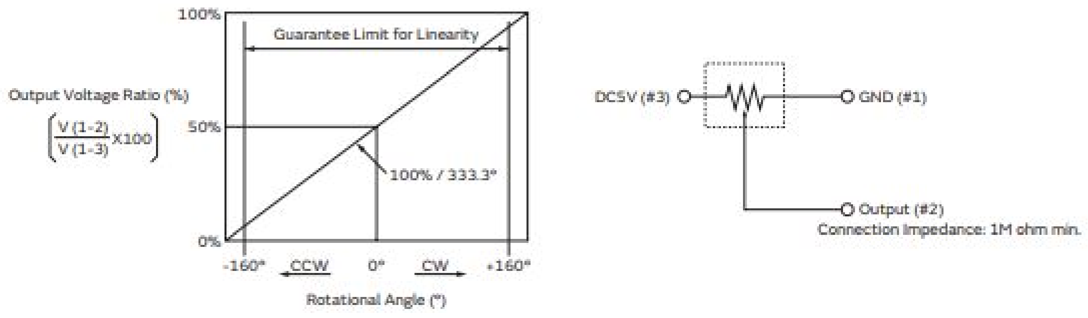


Figure 20. 3382G-1-104G schematic and output voltages.
https://www.murata.com/~/media/webrenewal/support/library/catalog/products/sensor/rotaryposition/r51e.ashx

The only drawback was that we had to design a PCB for it. Luckily for us, with the tutorial directed to us by Professor Nacho and Professor Grecku, we had the knowledge to design the PCB which makes this trivial. The SMD component required a simple schematic which is shown in the data sheet which we've pulled and displayed in figure 21. Since the schematic is very simple, it's very easy to implement the PCB for the SMD potentiometer.

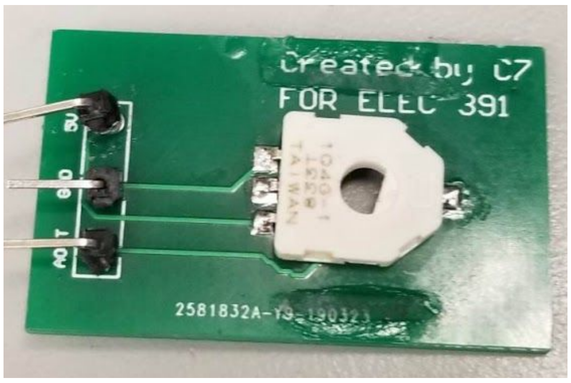


Figure 21. SMD Rotary potentiometer

Controls

The controls systems was built using one single Arduino Uno and it was flashed using the official arduino IDE. The following are tasks assigned to the control:

- **Generate PWM**
  We had to provide a PWM with variable duty cycle to the boost converter in order to be able to function properly. The PWM was generated by using one of the 3 timer registers of the Arduino Uno, to be more specific, it was implemented using Timer_1 configured to run at Fast PWM mode.

  Timer_1 was configured by writing to the internal registers of the arduino by referring to them using globals variable provided by the arduino framework bindings/mappings. The configurations for each individual register used to control Timer_1 where taken from the official Atmega328P data sheet, as well as from various open source references and frameworks.

- **Read Analog data from sensors**
  The sensing of our turbine's parameters was done using analog sensors which are fed into the arduino's internal ADC. In order to read and process this data, a open source State Machine design pattern framework called Automaton was used. Automaton framework lets you create state machines that react to stream of events and (inputs) and transitions between states performing actions on each state. Automato_Analog machine is a timer based machine that reads values from an analog pin from the arduino and the triggers callback to use the data in the main process. The callback frequency can be updated and the data of the analog pin can be averaged automatically while the machine idles waiting for the callback to send the average as one of the callback parameters if desired.
  A total of 5 automaton machines were used, one for the pre boost voltage sensor, preboost voltage sensor, current sensor, wind direction sensor, and wind turbine position sensor. Each sensor was configured differently and data was averaged and process differently in each callback.

- **MPP Tracking**
  MPPT tracking was developed by using a P&O algorithm in which the current power and voltage is compared to the previously read power and voltage, the duty cycle is then changed. 1.5 seconds are allowed for the system to react to the duty cycle change before the algorithm is called again and the process is repeated every 1.5 seconds. The MPPT is triggered by the same timer that calls the wind position function, when ever the timer counter hits 1.5 seconds the MPPT is called gain.

Integration

Submodule Assembly

For our integration concept, we tried our best to modularize individual parts and we parallelized our individual parts to fast-track our project. This allowed us to quickly have every submodule of our project ready for integration. Our integration is centered around having a PCB with ports than can easily be melded together. We created a centralized PCB that could be connected to most of our submodules. All of the ports that connect to the PCB are either terminal blocks or arduino header pins, which allows us to easily connect and modify our submodules if needed. This made integration easier as we could connect and disconnect the the generator, load, and the arduino from this PCB easily. Our PCB is shown in figure 23.

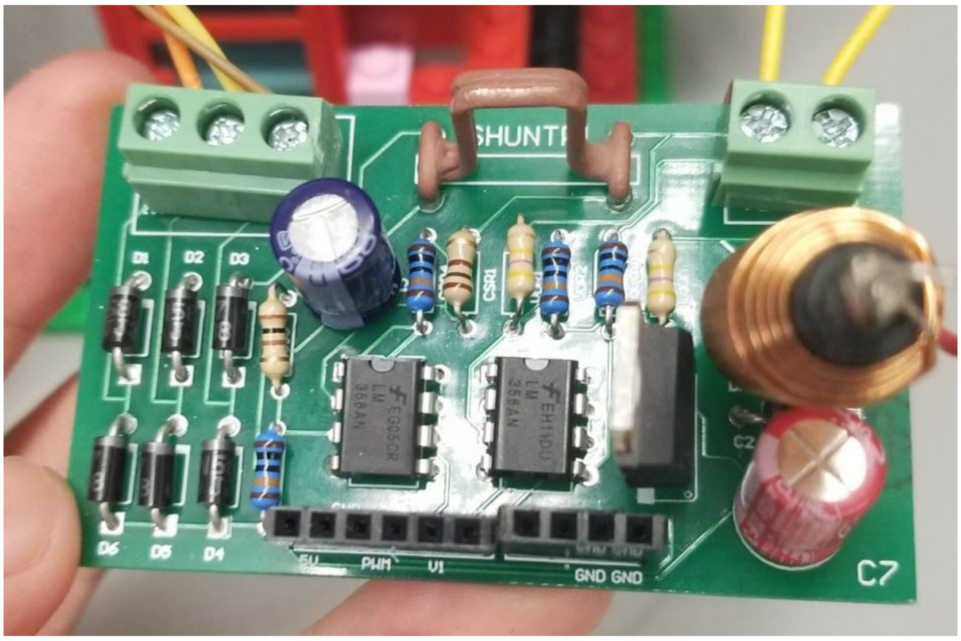


Figure 22. Final Demo PCB

We can split our integration into 3 steps:
1. Verifying individual components.
2. Integrating sensors with power electronics.
3. Integrating sensors, power electronics, and generator.
4. Integrating sensors and controls.

First of all, to ensure smooth integration, we tested sub modules to verify that it was completely functional before merging our modules sequentially. This allows us to debug easily as we would know which modules are conflicting with each other. This was our task for demo 2.
Secondly, we want to integrate the sensors with the power electronics. As we can see from figure 23, right from the start, the sensors, and the power electronics have already been integrated. We first supplied 5V DC before the shunt resistor, and supplied a pwm of 10kHz with 50% duty cycle. With the oscilloscope, we made sure each measurement made sense before moving forward. For example, we expect 10V post boost converter, then our analog voltage is calculated below. The 5.7 is the ratio of voltage divider as calculated in the sensor design part of the report.

$$V analog = 5V / 5.7 = 0.877V$$

And our post converter voltage as:

$$V analog = 10V / 5.7 = 1.754V$$

Once we verify that everything is working, we move on to the next step.

Thirdly, the generator is easily integrated to sensors and power electronics. In our PCB, we have a terminal block so that we can easily in plug our three phase generator and our load. Instead of the 5V DC supply, we attach our generator to the rectifier. With the exact same process as step 2, we test our generator, boost converter, and sensors with the oscilloscope. We expect it to work flawlessly, since the it worked with our 5V DC supply, with the only difference being the generator used as a supply. We clamped the generator down using the vice grips available in the labs. Once we verified that the generator, boost converter, and the sensors are working, we can interface with controls.

Finally, we integrate everything together. To interface our PCB with our micro-controller, we send our analog signal to a arduino compatible header pin. We can use dupont cables to attach these pins to the pins of the arduino for a clean look. The arduino has a 10-bit ADC, which means we have to convert the ADC value into voltage, current. For our voltage sensors, to convert our value we use the following conversion:

$$V_{actual} = ADC_{reading} * R_{divider} * V_{Max} / ADC\ value$$

$$V_{generator} = ADC_{preboost} * 5.7 * 5 / 1023$$

$$V_{boost\ converter} = ADC_{post\ boost} * 5.7 * 5 / 1023$$

Similarly, for our current sensors we the following conversion:

$$I = ADC_{reading} * V_{max} / (ADC\ value * R_{shunt} * G_{OP\ AMP})$$

$$I_{gen} = ADC_{current} * 5 / (1023 * 0.005 * 1000)$$

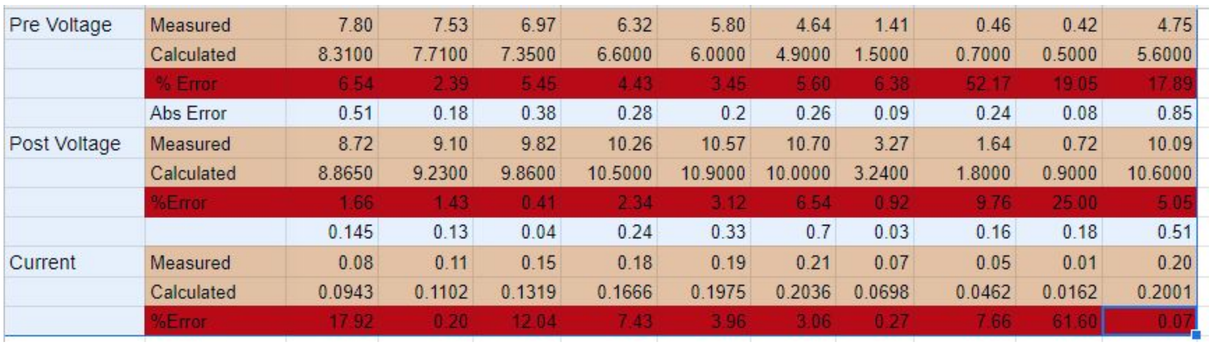
The accuracy and verification of our data will be discussed in the data validation section below.

For the potentiometers, we bought a potentiometer with a linear relationship between angle and voltage. We place our two potentiometers with their references identically aligned with 0° facing the front of the generator. We then know the wind direction is changing if we compare the two ADC values. The wind spins the potentiometer which changes the ADC value, which signals our stepper to rotate our generator until the ADC matches. Since we're using references, no conversions are needed.

With all the parts fully integrated, we can move on to data validation.

Validation

In order to ensure that our sensors are accurate, we had to validate our data to ensure that we are correctly reading output voltages. We created a table to compare our sensor readings with that of a known accurate sensor. In our case we compared our sensors with the oscilloscope. This data is constructed in to table below:



Pre Voltage	Measured	7.80	7.53	6.97	6.32	5.80	4.64	1.41	0.46	0.42	4.75
	Calculated	8.3100	7.7100	7.3500	6.6000	6.0000	4.9000	1.5000	0.7000	0.5000	5.6000
	% Error	6.54	2.39	5.45	4.43	3.45	5.60	6.38	52.17	19.05	17.89
	Abs Error	0.51	0.18	0.38	0.28	0.2	0.26	0.09	0.24	0.08	0.85
Post Voltage	Measured	8.72	9.10	9.82	10.26	10.57	10.70	3.27	1.64	0.72	10.09
	Calculated	8.8650	9.2300	9.8600	10.5000	10.9000	10.0000	3.2400	1.8000	0.9000	10.6000
	%Error	1.66	1.43	0.41	2.34	3.12	6.54	0.92	9.76	25.00	5.05
		0.145	0.13	0.04	0.24	0.33	0.7	0.03	0.16	0.18	0.51
Current	Measured	0.08	0.11	0.15	0.18	0.19	0.21	0.07	0.05	0.01	0.20
	Calculated	0.0943	0.1102	0.1319	0.1666	0.1975	0.2036	0.0698	0.0462	0.0162	0.2001
	%Error	17.92	0.20	12.04	7.43	3.96	3.06	0.27	7.66	61.60	0.07

Table 2. Data Validation of sensors

As stated in the sensors design section, when we reach the upper and lower bound of our sensors (0 and 5V in all cases), we can see that there are huge inaccuracies. Another reason of inaccuracies could be due to the incorrect reading of the oscilloscope. Since MPPT is changing output voltage and input current constantly, we might not be reading the correct values as it's changing constantly. In our case when we took these readings, we had our MPPT change duty cycle every 0.1s. This can be hard to read as we cannot instantly read a voltage value at a single time instant from the oscilloscope and compare to the voltage of our sensors. To remedy this error, we recalibrated our sensors to be accurate at MPPT O.P, to ensure that the voltage readings are completely accurate at our O.P. The result is that at about 70% duty cycle, we get a voltage error of about .3% and an absolute error of 0.03V (12.1V vs 11.8V)

For our rotary potentiometers, we verified they were working by attaching them to the analog pin of our arduino. We see that they exactly match the linear relationship specified by the datasheet.

Structure Assembly

In our design, we chose to use legos to build the structure for 2 main reasons. The first one is the low cost: We had spare parts of legos that were supposed to be recycled if they had not been used for this project. We as a group took our chances and decided to build the entire structure out of legos. The second reason was the flexibility of restructuring. If we were to use any other material or 3D printed parts, we would have to buy or make new parts if we wanted to change something in the design. Meanwhile with legos, we could make any shape we wanted in no time, saving us time and money. Moreover, we used lego glue that we already had to stick the pieces together to increase durability for the vibrations and shaking created by the fast speed turbine.

Our structure is made of various levels that hold a different part of our design on each.At the base, we have the cave-like structure that hosts the 2 Arduinos. On one of the sides, we have the load for the power electronics under a roof. On the levels above the ground, we have the power electronics PCB and the stepper motor respectively. PCB had to be placed not too far from the Arduinos and the load due to their direct connection on terminals. The stepper motor has a semi-long block placed vertically that has another gear on the top and is directly in contact with the gear, which the entire turbine was built. This allows the stepper motor to be able to rotate a very big structure (the turbine) without the need of a big torque. As mentioned earlier, the turbine (generator and the blades attached to it) are placed on the gear on the top. Finally, the wind sensing parts and the PCB that contains the ADC are placed behind the turbine.

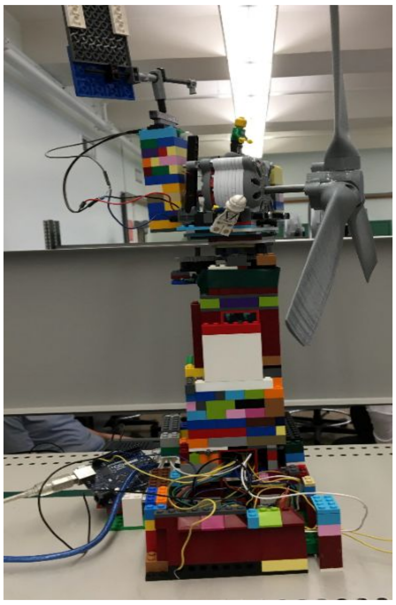


Figure 23. Final Structure

Testing

With our new chassis we had to verify the integrity of our structure and its ability to withstand our turbine while its operating. Since we created our chassis out of lego, we took necessary precautions by glueing our legos together to try to minimize the vibrational energy. We were pleasantly surprised at how minimal it wobbled under operation as demonstrated in the final demo. We did notice that our structure slid on the metal desk, so we remedied this problem by adding some rubbery electrical tape to increase friction.

We noticed that after integrating everything, that our integration introduced some noise into our system. Our previously clean ADC readings were being polluted by something in our system. To ensure we get steady values, we added some capacitors across our stepper motor, and our sensors. We also performed some software filtering by averaging values. As discussed later, since we settled on a duty cycle increment period of 1.5 seconds, and we're sampling our ADC every 1ms, we performed a running average of 500 values to smoothen our readings.

We can then test our MPPT algorithm. Implementing our MPPT algorithm seamless. Recall from our modeling that we actually simulated our MPPT algorithm with matlab blocks, which made transplanting our MPPT algorithm to C code very easy. To validate our MPPT, we attached a 100Ω as our load for our generator. Then to find the duty cycle corresponding with maximum power, we manually changed the duty cycle to determine where our MPPT duty cycle was. We found that it was at 70% was where the we got the highest power. After knowing the MPPT duty cycle, we initialized our MPPT algorithm. To verify that the algorithm works, our algorithm would keep the duty cycle to 70%. Initially we changed our duty cycle every 100 milliseconds and increment it by .1%. We noticed that this was not optimal as the inductors and our capacitors had no time to settle. This led us to increase our duty cycle increment, and the period at which we incremented our duty cycle. We finally settled on a duty cycle increment of 1%, with a 1.5 second period.

When testing our wind tracking close controlled loop, we noticed the wind fin does not correctly align with with the wind direction of the fan. We narrowed the problem down to the the single sided wall that is on our benches. It seemed that wind was being reflected off the walls and pushing our sensors towards one direction. Luckily for us this was an easy fix as it was a constant offset from the centre. We corrected this within the code.

Once we tested that everything was indeed working, we could upload our data to the cloud and onto our website.This website and it's construction and feature will be discussed the in the following section.

Extra Website Service

C7 Monitoring System

We decided to include a real time monitoring system to track the performance and health of the turbine. We believe that this is a vital feature that every turbine should have, thus the system was built using a modular design pattern; which means, that our monitoring system should be able connect to any other turbine control system in a seemingless fashion.

The website is till live and can be viewed at: https://monitor.febgdev.com. How ever, because having cloud services running costs money, the website is static, meaning that data is currently preloaded and the web app will NOT connect to the cloud services. Please contact any of the team members if it is required to deploy the services again for demonstration purposes.

In order to be able to explain this section in detail it is important to first define the following terms since they are going to be referenced in the next sections.

- **Microservice Architecture:** Architectural style in which a collection of small independant services (server <-> database pair) is used instead of a single big monolith service.

- **Microservice:** Independently runed, deployed and maintained server <-> data base pair. One of the services of a microservice architecture collection
- **Docker Container:** Portable lightweight isolated OS environment (similar to a VM)
- **EC2 Instance:** Amazon Web Services linux server (linux machine running on cloud)
- **noSQL database:** Database which stores Object data types instead of SQL type columns and rows in a table

We've also constraint our website to the following requirements:

- Secure Connection (SSL)
- Certified Domain
- Built using microservices
- Run on docker containers (Lightweight VM's)
- Hosted in the "Cloud"
- Accessible from anywhere in the world
- Use noSQL databases
- Handle at least 100,000 Req/sec
- Auto Scalable
- Stream live data (Less than 500 millisec latency)

In order to be able to accomplish all of our requirements we decided to create a web based service "**C7 Cloud Monitoring"** to which anybody could access by using a web browser. The information being monitored on the web application is polled from any microcontroller attached to the turbine (provided wifi/ethernet capabilities). This information will be then be processed and stored safely in a cloud database. **C7 Cloud Monitoring** resides on top of a combined stack of cloud services powered by Google Cloud Platform and Amazon Web Services. The specific technologies and architecture used for this feature will be explained below in detail. Below, a flowchart of the cloud services network created for this project.

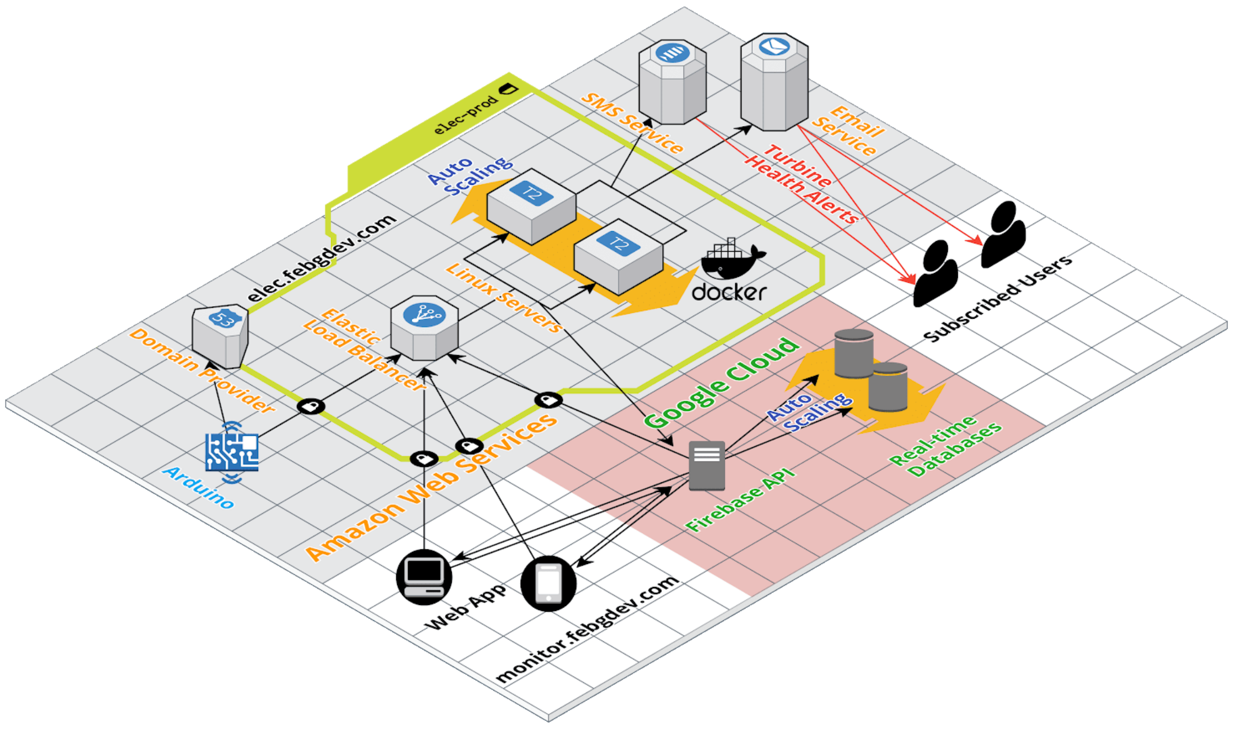


Figure 24.Architecture of the cloud

Data Recording

As mentioned in previous sections, the main functionality of the turbine's control is to record information about wind, duty cycle, pre/post boost voltage and current in a periodical basis to be used for orientation and MPPT. In order to be able to upload data to cloud in a seemingless way, the control's main logic is simply extended by taking the already recorded data and transfer it an Arduino WIFI (slave) over I2C protocol every second. The slave Arduino WIFI establishes an HTTP connection with our dedicated data collection server which will process and store the data.

By using this approach we are able to meet an important aspect of the "Add-on" requirement. Our arduino slave could be attached to any other microcontroller supporting I2C serial communication through two pins (**SDA, SLK**). This would enable them to upload data to the cloud by simply "printing" to the connection.

Note that this is not necessary if the microcontroller supports wifi (Eg. Raspberry pie) or if it uses an external wifi chip (Eg. ESP8266). In this case data can be sent directly to our cloud server over HTTP request by one simple call.

Data Collection/Processing

Our first microservice is Node.js server built on top of the well known Express framework. This service exposes and HTTP REST-API to which the arduino and other services can communicate. In order to host this server on the cloud, we decided to use and configure an AWS EC2 instance which is nothing more than a custom made linux machine running on our own cloud network.

This server is used as a gateway between the arduino and the database, it takes the data being passed by the arduino and then it securely encrypts and redirects the data to a real-time database. It is also used as the gateway through which any engineer can instantly access to information about the turbine as well as logs from the microcontrollers without the use of the web application (Debugging, maintenance, etc).

Lastly, since this is the service that process and stores data this is the service responsible for handling at least 100,000 Req/sec mentioned in the requirements. This requirement was chosen with the idea of ideally having not **one** but **many** turbines in a wind farm. So for example if we have a wind farm with many turbines all sending data multiple times per second we need be able to handle and store all of those requests. This was accomplished by configuring and deploying an **Elastic Load Balancer** on AWS. The load balancer is configured to receive all the requests and then route each request to different identical copies of the same Node.js server that was mentioned above. So for instance, suppose we have 10 turbines sending data at exactly at the same time, without a load balancer a single server had to handle all 10 request at the same time. Now if a load balancer is attached along with one more identical copy of the server, the traffic is distributed evenly between the two servers, so they both handle 5. In order to avoid wasting resources when not being used (Eg. having two servers when we only have 1 turbine) custom scripts were written to configure auto scalability for the load balancer. This means that the load balancer will analyze the traffic that it is receiving, and decide based in out configuration how many server are needed to server that given traffic. If the actual server count is above the calculation, then the load balancer kills a server, if the calculation is below then it simply deploys a new copy.

Normally instantiating a brand new linux machine with a custom Node.js server take quite some time. For this reason our Node.js server runs inside a docker container as specified by our requirements. Docker container can be booted from zero-to-online in less than a minute. This helps with the efficiency of our load balancing by killing and booting machines faster.

Web Application

We created 4 core services for our website: **real-time monitoring**, **historical visualization**, **analytics**, and an **alert service**.

Real-time monitoring

Our real-time monitoring feature streams live data from the wind turbine. This service was built using Firebase and it is hosted in Google Cloud Platform. The firebase server was written in Node.js and was able to connect via WebSockets to the same real-time database used by our data collection service. This service was chosen to be served in GCP because the real-time database is also hosted in GCP, thus there is less latency in the connection between them.

Historical Visualization

This feature is hosted in the same firebase server as the real-time monitoring system in GCP. The differences is that is its an static RESTful-API that GETS a single query from database. For this service, the user is presented with a calendar in the web app. The user con pick a range of dates with time with a precision of up to 1 second (Eg. April 1st, 2019 - 10:49:25 am). The web application sends this parameters to the GET end point and the server returns all the available turbine data for that range of dates. The data is then presented to the user in the from of line charts.

Analytics Service

This feature is hosted in the same RESTful-API than the **Historical Visualization** service and it accessible through a different HTTP end-point. Like for the **Historical Visualization** endpoint, a range of dates is passed to the server as parameter of an HTTP GET request. The server then uses this parameters to fetch all the data available in between those dates. In contrast to the **Historical Visualization**, this time data is not just simply returned to the web app, inteads the server process the data and calculates the following data:

- Average voltage, current, power, duty cycle throughout the entire range
- Average voltage, current, poser, duty cycle per DAY throughout the entire range
- Average voltage, current, poser, duty cycle per HOUR throughout the entire range

The data is then sent back to the web application, and it is presented in the form of pie graphs and bar graphs.

Note 1: Any other desired type of analytics algorithm can be easily added to the service in the from of Javascript scripts.

Note 2: Average voltage, current, poser, duty cycle per HOUR throughout the entire range end point is implemented but it is not displayed on the web app due to timing shortage.

Alert Service

Our website also features an alert system, where any visitor can place healthy values as a parameters, as well email and/or password and our server will send an email or SMS message warning the users if our voltages, and current exceed the healthy values.

Every time new data is saved to the database, a new process triggered on a different thread of the OS hosting the server that saved the data. During this after save trigger, the data is compared to all the threshold values provided by different users. If the values of the data that triggered the trigger are below/above the threshold values provided by the user, then the owner of the alert is notified.

Notifications are delivered by email and SMS through the use of Amazon web services infrastructure. To follow the microservices design pattern the email and sms services are completely independent from one another. Both services are hosted in AWS and in order to send an alert, a simple API is provided in which the user only provides the recipients email of phone number as well as the message that is being sent as parameter of a GET HTTP request.

Cost Analysis

We were given a budget of $1000 dollars, with our grade being dependent on how much we spent. For our turbine, we spent a total of $379 from the budget. This included the following:

- Water-jetting stator and rotor ($137).
- PCBing our designs ($74).
- Buying our electrical components ($167).

However, we were not entirely efficient in our budget and only actually need $229 out of the $379. In regards to the water-jetting, we only used our stator which means that the cost of the rotor waterjet is not justified in our final cost. Our final design required a $72 waterjet. For our components, we spent a lot of money prototyping and doing tests. Although there were lots of components that we haven't used in our final design, we feel that it was needed for us to forward our ideas the project. A total of $83 of components were crucial out of the $167. In total we only needed to use $229 out of the $379 used, which gives us a budget efficiency of 60%. A full list of exactly what is spent, and what is used is detailed in the appendix.

A lot of the money in our budget was used as prototyping and because of that reason, we can replicate this project for even less. We discovered that we could order PCBs online for $5 + shipping. This would reduce our cost heavily as we've expended $74 of our budget on PCBs. With a working final design, we can display the bill of materials of our final design as shown in table 3. The green highlight on the table signifies parts that we've already owned before the project. Because we want to be as extensive as possible, we've included components that we've originally already owned.

Part #	Description	Quantity x Price	Total
M8275-ND	Inductor for Boost Converter	1 x $3.26	$3.26
Terminal Blocks	Load, Generator input / output	$0.49 + $0.36	$0.85
Assorted Resistors	100k,470k,1M, 100 resistors	N/A	N/A
Assorted Capacitors	100uF, 200uF	Approx $1.	$1
LM358	Sensors	2 x $0.59	$1.18
22R476C	Schottky Diodes	7 x $0.39	$2.73
STP60NF06	N- channel for B.C	1 x $1.46	$1.46
MRS-12985L00FE140	5mOHM shunt	1 x $4.96	$4.96
MIKROE-1530	Stepper 5V for turbine wind tracking	1 x $12.15	$12.15
3382G-1-104G	SMD potentiometer for windtracking + feedback	2 x $3.96	$8.00
Arduino Header Pin	.1 in spaced pins	N/A	N/
Magnetic Wire	N/A	N/A	N/a
Arduino Rev 2	Our mcu	$45	$45
PCB	For boost converter + SMD potentiometer	$5 x 2	$10
Waterjet	Stator	$72	$72
Lego Chassis		N/A	N/A
Total	**N/A**	**N/A**	**$162.59**

Table3: Final design bill of materials

As shown we can replicate our final design for **$162.59**. If we discounted all the parts we've already owned, then we would have needed $100. That means with our current knowledge, and previously owned parts, we could have replicated this project for $100.

Timeline

In lieu of the final demonstration and how different our timeline actually was in comparison to our preliminary timeline, we condensed our final Gantt Chart into one single stage. The preliminary Gantt charts have also been included for comparison

Gantt Chart for the project

.

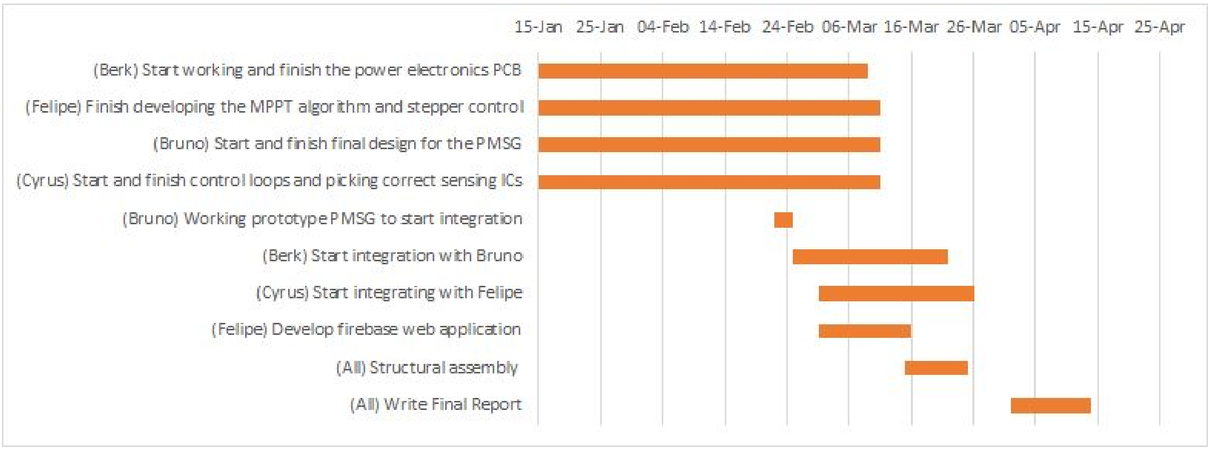


Figure 25: Final Gantt Chart of the overall project

<u>Preliminary Gantt Charts from our project proposal</u>

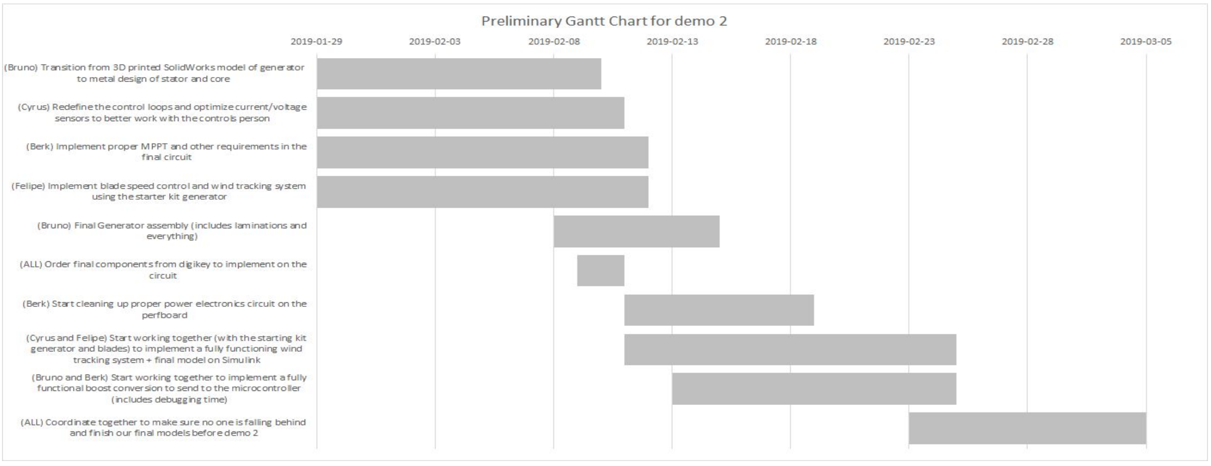


Figure 26: Gantt chart for demo 2 of the proposal

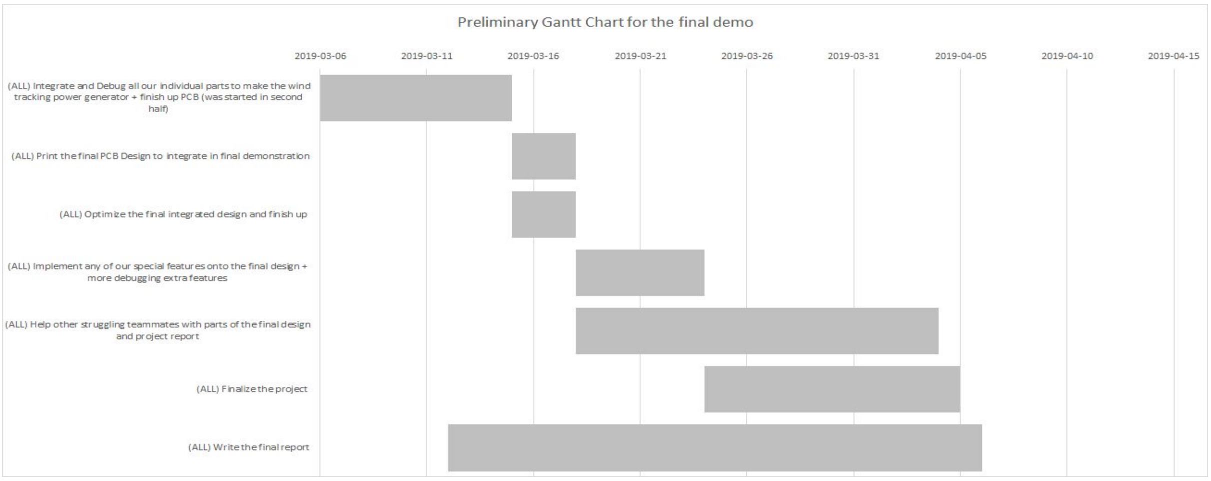


Figure 27**:** Gantt Chart for the final demo in our proposal

Our timeline was more or less met, and our integration process began around 20 days after what we have initially predicted. This is mainly due to a lack of knowledge of when each part could do certain steps (ie. learn how to submit a Waterjet request, PCB submissions). So most of the time was spent waiting until the next steps which will allow us then to integrate.

Final Remarks

Originally we set out to find out if placing our wind turbines in remote areas were cost efficient. We determined it would cost $162.9 dollars to implement our current design. However, with our current design, our turbine would produce $116/Watt, which makes it uneconomical as well as inefficient. We see that commercial turbines produce about $1.75/Watt, if we were able to scale our project up, maybe we can minimize cost per power. As of now, our design is not feasible. Despite our wind turbine design not being feasible, we did design a monitoring system that can be feasible and implemented on a large scale wind farm. Our website is modular and only requires I2C communication to work. We believe that our website covers the most of the basic functions required to monitor a power system. With that said, we did accomplish our secondary goal, which to learn. We learned a lot during this course, and at the end of this project, we believe that it’s actually more important than our primary goal; with the most important topic learned being I2C communication, HTTP, and PCB design. All in all, we were ecstatic with our final implementation of our wind turbine. Finally, we’d like to formally thank Professor Nacho, Professor Grecu for providing us with this opportunity, and we’d like to thank all the TA’s for all their support during the course.

References

Sen, Paresh Choudra. *Principles of Electric Machines and Power Electronics*. Wiley, 1989.

“analogRead()”, *Arduino*, 19 January 2019,
https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

“Arduino and MCP3008.”, *Arduino Learning*, 10 Mar 2019
http://arduinolearning.com/code/arduino-and-mcp3008.php

“How wind turbines work”, *USDOE*, 6 January 2019,
‚https://www.energy.gov/eere/wind/how-do-wind-turbines-work’

“MPPT Algorithm”, *MathWorks*, Feb 16th, 2019,
https://www.mathworks.com/discovery/mppt-algorithm.html

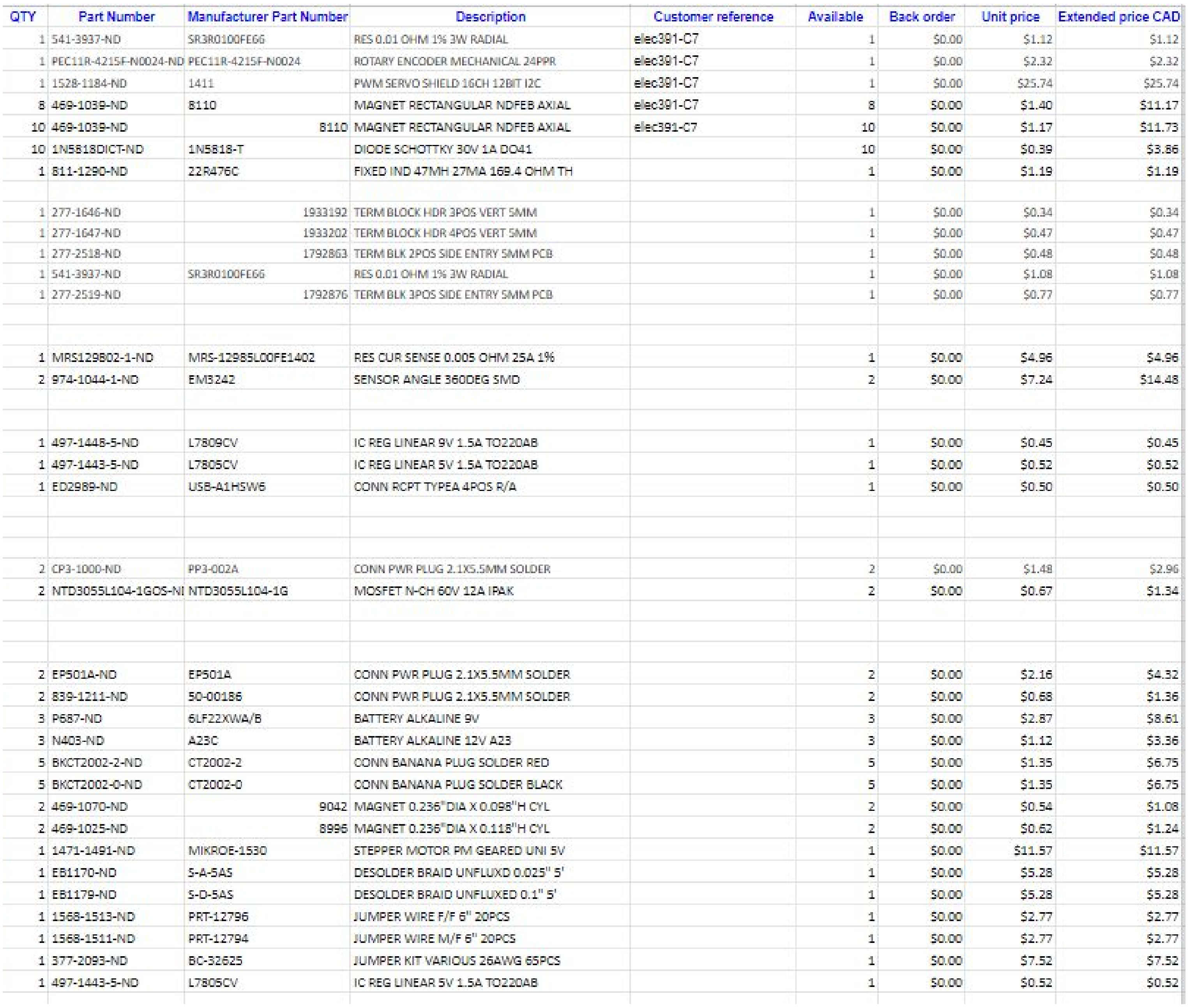
“The Differential Amplifier”, *Electronics Tutorial*, 5 February 2019,
https://www.electronics-tutorials.ws/opamp/opamp_5.html

“Unity gain amplifier or voltage follower in a voltage divider”, *Analogic Tips*, 5 February 2019,
https://www.analogictips.com/voltage-follower-unity-gain-amplifier-voltage-divider-faq/

“Wifi Library”, *Arduino*, 14 February 2019, https://www.arduino.cc/en/Reference/WiFi

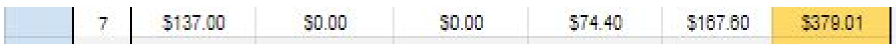
Appendix A : Budget Usage

Components List:



QTY	Part Number	Manufacturer Part Number	Description	Customer reference	Available	Back order	Unit price	Extended price CAD
1	541-3937-ND	SR3R0100FE66	RES 0.01 OHM 1% 3W RADIAL	elec391-C7	1	$0.00	$1.12	$1.12
1	PEC11R-4215F-N0024-ND	PEC11R-4215F-N0024	ROTARY ENCODER MECHANICAL 24PPR	elec391-C7	1	$0.00	$2.32	$2.32
1	1528-1184-ND	1411	PWM SERVO SHIELD 16CH 12BIT I2C	elec391-C7	1	$0.00	$25.74	$25.74
8	469-1039-ND	8110	MAGNET RECTANGULAR NDFEB AXIAL	elec391-C7	8	$0.00	$1.40	$11.17
10	469-1039-ND	8110	MAGNET RECTANGULAR NDFEB AXIAL	elec391-C7	10	$0.00	$1.17	$11.73
10	1N5818DICT-ND	1N5818-T	DIODE SCHOTTKY 30V 1A DO41		10	$0.00	$0.39	$3.86
1	811-1290-ND	22R476C	FIXED IND 47MH 27MA 169.4 OHM TH		1	$0.00	$1.19	$1.19
1	277-1646-ND	1933192	TERM BLOCK HDR 3POS VERT 5MM		1	$0.00	$0.34	$0.34
1	277-1647-ND	1933202	TERM BLOCK HDR 4POS VERT 5MM		1	$0.00	$0.47	$0.47
1	277-2518-ND	1792863	TERM BLK 2POS SIDE ENTRY 5MM PCB		1	$0.00	$0.48	$0.48
1	541-3937-ND	SR3R0100FE66	RES 0.01 OHM 1% 3W RADIAL		1	$0.00	$1.08	$1.08
1	277-2519-ND	1792876	TERM BLK 3POS SIDE ENTRY 5MM PCB		1	$0.00	$0.77	$0.77
1	MRS129802-1-ND	MRS-12985L00FE1402	RES CUR SENSE 0.005 OHM 25A 1%		1	$0.00	$4.96	$4.96
2	974-1044-1-ND	EM3242	SENSOR ANGLE 360DEG SMD		2	$0.00	$7.24	$14.48
1	497-1448-5-ND	L7809CV	IC REG LINEAR 9V 1.5A TO220AB		1	$0.00	$0.45	$0.45
1	497-1443-5-ND	L7805CV	IC REG LINEAR 5V 1.5A TO220AB		1	$0.00	$0.52	$0.52
1	ED2989-ND	USB-A1HSW6	CONN RCPT TYPEA 4POS R/A		1	$0.00	$0.50	$0.50
2	CP3-1000-ND	PP3-002A	CONN PWR PLUG 2.1X5.5MM SOLDER		2	$0.00	$1.48	$2.96
2	NTD3055L104-1GOS-NI	NTD3055L104-1G	MOSFET N-CH 60V 12A IPAK		2	$0.00	$0.67	$1.34
2	EP501A-ND	EP501A	CONN PWR PLUG 2.1X5.5MM SOLDER		2	$0.00	$2.16	$4.32
2	839-1211-ND	50-00186	CONN PWR PLUG 2.1X5.5MM SOLDER		2	$0.00	$0.68	$1.36
3	P687-ND	6LF22XWA/B	BATTERY ALKALINE 9V		3	$0.00	$2.87	$8.61
3	N403-ND	A23C	BATTERY ALKALINE 12V A23		3	$0.00	$1.12	$3.36
5	BKCT2002-2-ND	CT2002-2	CONN BANANA PLUG SOLDER RED		5	$0.00	$1.35	$6.75
5	BKCT2002-0-ND	CT2002-0	CONN BANANA PLUG SOLDER BLACK		5	$0.00	$1.35	$6.75
2	469-1070-ND	9042	MAGNET 0.236"DIA X 0.098"H CYL		2	$0.00	$0.54	$1.08
2	469-1025-ND	8996	MAGNET 0.236"DIA X 0.118"H CYL		2	$0.00	$0.62	$1.24
1	1471-1491-ND	MIKROE-1530	STEPPER MOTOR PM GEARED UNI 5V		1	$0.00	$11.57	$11.57
1	EB1170-ND	S-A-5AS	DESOLDER BRAID UNFLUXD 0.025" 5'		1	$0.00	$5.28	$5.28
1	EB1179-ND	S-D-5AS	DESOLDER BRAID UNFLUXED 0.1" 5'		1	$0.00	$5.28	$5.28
1	1568-1513-ND	PRT-12796	JUMPER WIRE F/F 6" 20PCS		1	$0.00	$2.77	$2.77
1	1568-1511-ND	PRT-12794	JUMPER WIRE M/F 6" 20PCS		1	$0.00	$2.77	$2.77
1	377-2093-ND	BC-32625	JUMPER KIT VARIOUS 26AWG 65PCS		1	$0.00	$7.52	$7.52
1	497-1443-5-ND	L7805CV	IC REG LINEAR 5V 1.5A TO220AB		1	$0.00	$0.52	$0.52

Waterjet usage : $137
Component : $167
PCB : $74.40



	7	$137.00	$0.00	$0.00	$74.40	$167.60	$379.01