Maker Portfolio Details

By Sean Donelan

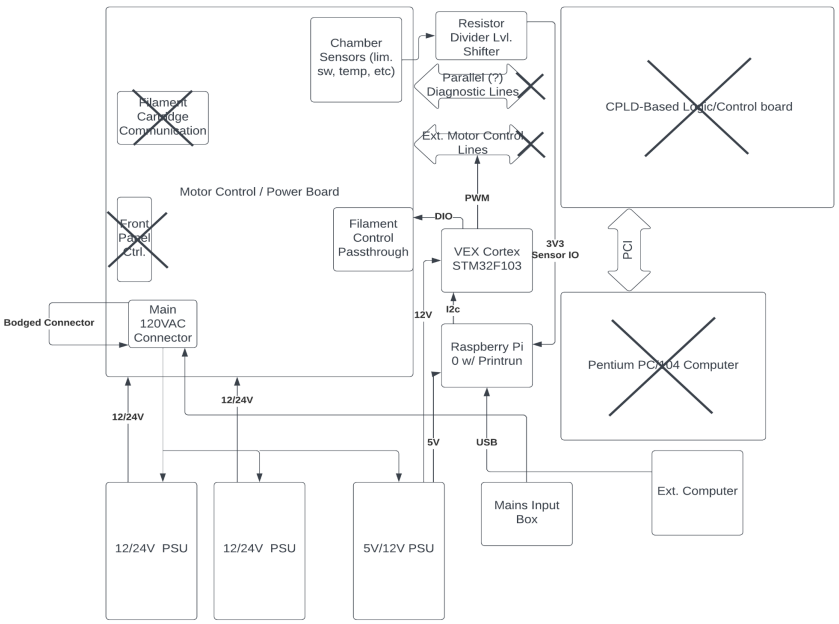
Much of my personal time is spent in a self-made workshop at school, which I helped set up in an unoccupied area, and my workbench at home. Over the course of roughly a year, I have created the following four projects from different pieces of electronic waste that I have found and managed to understand.

1. Stratasys 3D Printer Recovery & Repair

Behind my school district's office, there is a scrap pile where I found the following:
-- A Stratasys Dimension Elite ABS FDM 3D-printer
-- Soiled metal cart with Cleanstation Soluble Support Removal System DT3
-- 8 half-empty bottles of Stratasys WaterWorks P400 solute
-- 1 broken Stratasys ABSPlus cartridge, filled with about 4 inches of support material
-- 1 broken Stratasys ABSPlus cartridge, filled with the half full of model material
All of these objects seemed to follow a common theme -- together they could be used to print high quality, durable 3D prints out of ABS plastic. So, I decided to attempt to get the printer and its supporting equipment up and running.

Repairing the Printer

I had trouble starting up the Dimension Elite printer, despite my attempts to bypass the broken thermal regulation switch and provide power to the printer's logic and motor control boards. It turned out that the hard drive was damaged and unable to be booted from, causing the Pentium-based PC/104 form factor industrial computer responsible for controlling the printer to fail. In an attempt to resolve the issue, I removed the faulty hard drive and used Autopsy, a forensics software based on TheSleuthKit, to locate an installation of embedded Linux with an outdated Kernel (v2). However, I was disappointed with the results of the filesystem of this Linux installation, as it only contained compiled executable binaries that would require extensive reverse-engineering efforts to use. As an alternative approach, I chose to reverse-engineer the motor controller's hardware instead. Below is a simplified drawing of the hardware configuration I settled on:



Technical Details

Crossed-out components represent unused sections of the old system. The onboard Raspberry Pi Zero runs a stripped-down modified version of Printrun to control the printer.

The Pi Zero acts as a USB gadget, allowing it to mount storage for prints and communicate using TCP.

The printer can now produce quality objects in both modeling (ABS plastic) and support material, which we could then chemically treat with the CleanStation to make nice plastic products.

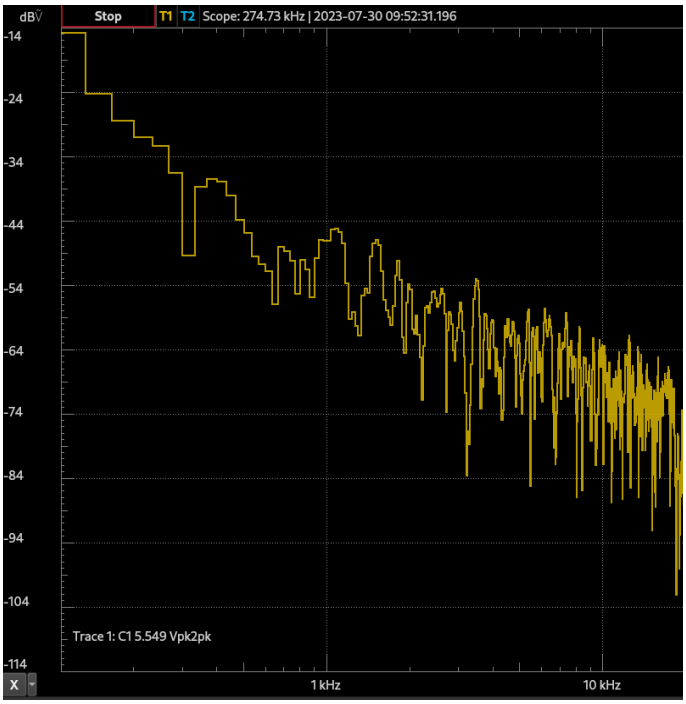
2. Colored Noise Maker - Therapeutic Sleep Aide

I have several friends who struggle with sleeping issues. Interestingly, they all believe that white noise or similar variants are comforting and help them sleep better. To assist my friends in finding suitable noise preferences for restful sleep, I have created a system to measure and utilize such preferences easily. After a quick survey of my friends, I discovered that they could categorize the types of noise they prefer based on the different “colors” of noise. Each “color” represents a unique characteristic power spectrum in a signal. I had to create every variation of this particular noise color to produce efficient sound from a device. Afterward, I combined them following their input intensities.

Computationally Generating the Noise

After conducting research, I decided to repurpose existing programming to generate white, blue, Brownian (red), violet, and pink noise:
--Blue, Brownian, and Violet Noise: I relied on the work of Stephane Plaszczynski’s rather ingenious method of producing noise with a specific frequency spectrum, characterized by filtering a Gaussian white noise stream into streams with arbitrary 1/f^alpha characteristics.
--Pink Noise: I used a Python implementation of the Voss-McCartney algorithm, which is more efficient than the algorithm used for the other streams.

I was especially happy with how my Brownian noise generator worked, as it had an output power spectrum distribution that looked like this:



Hardware Implementation and Testing

The five streams of noise were rather computationally difficult to produce, which was likely a product of my own inefficient programming. I could only manage an output bitrate of 1100Kbps on my originally planned embedded computer before the program would attempt to buffer samples or overrun the host machine’s computing resources.

I originally planned to run the five noise stream generators on an ODROID C0 I found in a dustbin at a convention, but I unfortunately had to switch to a considerably larger host. I eventually settled on a stripped-down Acer C720 Chromebook taking input from potentiometers connected to one half of an Arduino (the other half was tragically lost in a sawing accident).

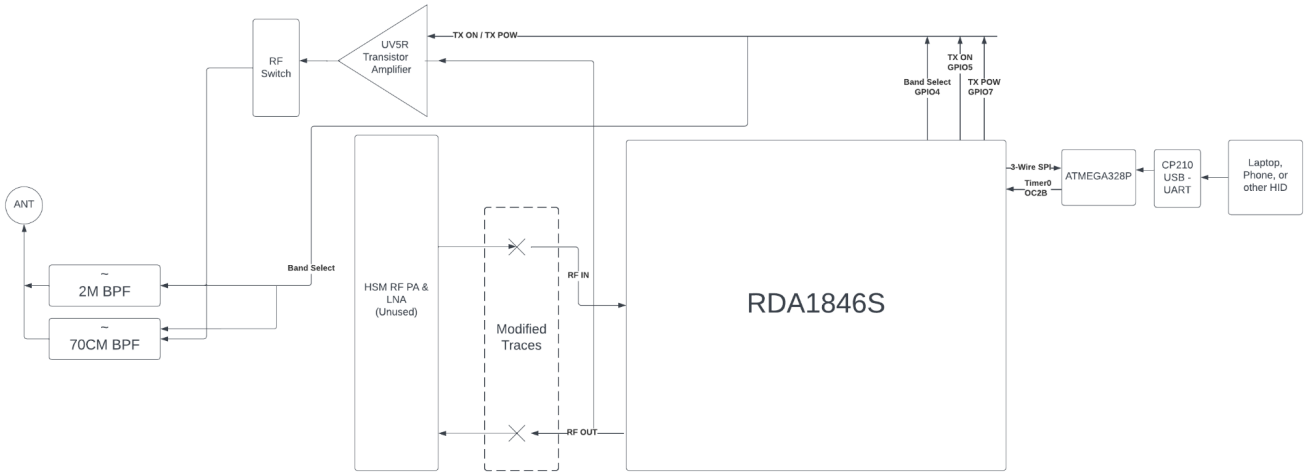
After my friends tried using my somewhat obtuse device, they reported improvement in their sleep compared to a placebo or plain white noise.

3. Packet Radio Project - Getting a Signal Anywhere

I go to Northgate High School, which is notable for its unique architecture. The school is situated on the outskirts of town and is primarily constructed with concrete, resulting in extremely poor cell signal reception. To address the issue, I endeavored to create a monopole antenna that would ideally possess superior characteristics compared to the antenna on my phone. Regrettably, it yielded disappointing results, prompting me to search for a more effective resolution.

Designing the Radio

I decided to overcome this problem using RF power and my amateur radio license. After some thought, I found that AFSK modulation combined with an FM signal would allow for adequate data transfer speeds as well as power within my grasp. A block diagram of the radio I constructed is below:



The modulation of the RF signals is based on the RDA1846 FM transceiver on a chip, which uses a rather clever amalgamation of an I/Q demodulator and ADC (with integrated DSP system) for receiving while sharing the same VFO as the transmitting side. From the outset of this project I believed that I could simply modify an off-the-shelf Baofeng UV5R to allow for my microcontroller to send in signals to the RDA1846's three-wire SPI interface. However, the newer revisions of the UV5R use a custom version of the RDA1846 for the UV5R, which have nonstandard register mappings making programming impossible. So, I used an external module that has a standardized RDA1846 onboard.

Setbacks and Piecing together the Radio

After carefully understanding the slight differences in RF characteristics of the two chips, mainly being output power, I used pieces of 50 ohm coaxial cable cut from a broken laptop Wi-Fi antenna to send the input and output RF signals of the RDA1846 to the RF front end of the Baofeng UV5R. After modifying the gate collector voltages on a few of the transistors of the UV5R's amplifier, I managed to get a workable 21.3V RMS to the output feed line after filtering, equating to about 8W of average radiated power.

Transmitting and Receiving Data

After removing the Arduino bootloader from my ATMEGA328P (Arduino UNO) board, I modified some fuses controlling TIMER0, which I would later use to modulate AFSK signals. I could now achieve a baud rate of a whopping 1450 baud (~1.4Kbps), slightly improving over the originally planned 1200 baud. I added little in the way of link-layer (or OSI Level 2) code due to time constraints, but my radio could page my parents about my whereabouts.

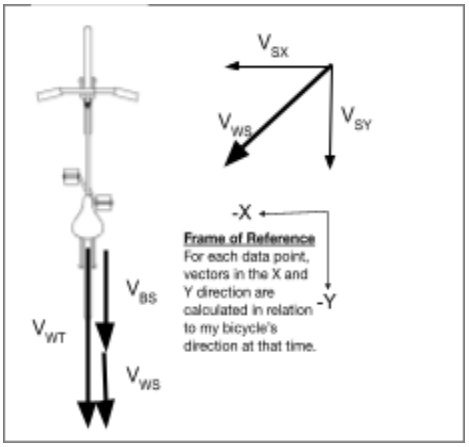
The base station receiver was little more than a Raspberry Pi running a heavily modified version of RpiTx, software that uses the Raspberry Pi's onboard timers and GPIO to modulate an RF signal, a borrowed RF amplifier, homemade RF filters, and a borrowed RTL-SDR dongle using GNU Radio for receiving. The radio performs admirably and has helped keep me in contact with my parents and the outside world even during power and internet outages.

4. Bicycle Anemometer Project - Beating Headwinds

I really like biking, and every day I ride my bicycle to work, school, errands, or friends. However, there are many literal resistive forces at work that make biking less enjoyable for me, the chief of which is wind. To alleviate the associated miserable resistance headwinds pose to biking fast, I could make myself more aerodynamic. Or, instead of wearing skin-tight clothing and stripping my bike down of components, I could avoid the wind altogether.

Collecting the Data

To bike without being affected by strong winds, I need to determine how much wind blows in the opposite direction of my travel on different routes. I surmised that there would be a correlation between the wind resistance I encountered on my path and the data recorded by nearby weather stations. As weather stations typically provide hourly updates, and my commute lasts roughly 30 minutes, I plan to use the hourly weather readings to forecast the wind resistance I may experience on the trail. To establish a correlation, I built a package of sensors for my bicycle. The most important of these sensors is an anemometer (wind meter) created from a discarded PC fan. After rewiring the PC fan to work as a DC generator (rectifying the output of the fan's coils), I could use a ADS1015 ADC to find the speed of wind moving through the fan. Then, a GPS radio, accelerometer, compass, and SD card connected to an ESP32 microcontroller was used to collect the data needed. I then rode my bicycle on a few routes to and from work.

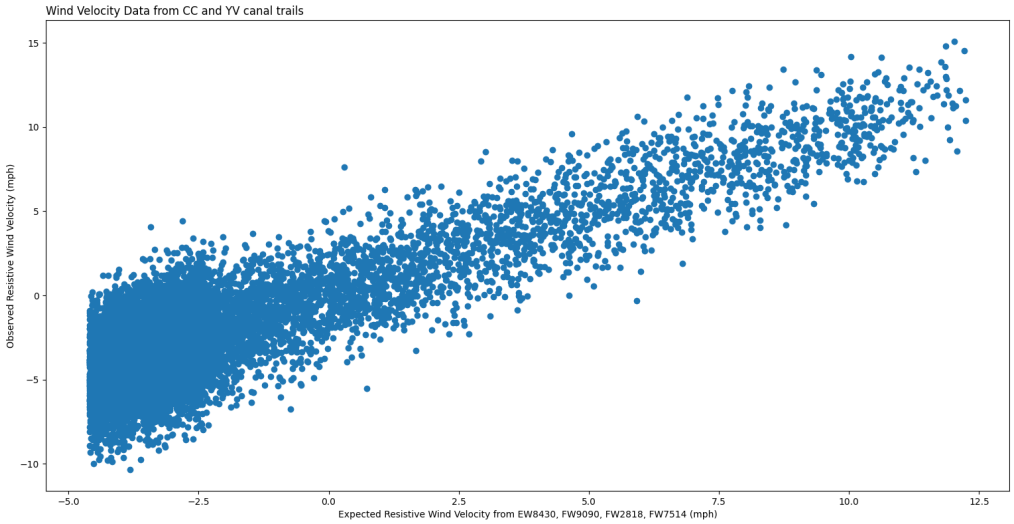


Normalizing the Data for Statistical Inference

To find a correlation, I needed to find a normalized variable representing the headwind experienced at any particular point on a route and the expected headwind at that same point. A diagram of the various vectors I used to represent wind velocities along my route is shown. To do this, I subtracted the velocities of my bike measured at every point (V_{BS}) from its corresponding measured wind velocity (V_{WT}) to find the headwind experienced at that time in relation to my bike's position and direction. Then, I found the direction of the wind expected by the weather station in relation to the direction of my bike, and used that to find the X and Y components of the expected wind velocity (V_{SY}). I could repeat the process of finding expected values for different weather stations along my route to find the stations that could most accurately predict the wind I felt.

Predicting Wind on the Trail

For every data point, I then calculated the expected resistive wind velocity using the geographically nearest APRS weather station with wind direction and velocity sensing capabilities. I then placed the data on a scatterplot and ran a computer regression analysis. The results were rather conclusive, as shown by the P-values in the regression analysis. My data as well as an example prediction is shown below. This then allowed me to predict which trail had the lower average wind speed (in relation to the frame of reference from above) using the readings of weather stations near me.



(Above) The rather good-looking expected vs observed wind velocity graph, showing a strong, positive, linear correlation.

S = 1.8 R^2 = 0.855 R^2(adj) = 0.855

Predictor	Coef	SE Coef	T	P
Constant	0.027	0.02	1.331	0.183
Expected Resistive Wind Velocity (mph)	0.998	0.003	290.838	<0.001

Analysis of Variance

Source	df	Sum of Squares	Mean Square	F-value	P-value
Regression	1	274154.631	274154.631	84586.757	<0.001
Error	14398	46665.442	3.241		
Total	14399	320820.073			

< The associated regression analysis shows that my predictions can account for 85% of variation in the observed wind. This r-sq value greatly exceeds my original expectations for this project.

< An example map showing that the average predicted wind speeds suggest that I should take a route mainly along the Contra Costa Canal Trail, as it has an overall lower mean expected resistive wind velocity along it than the YV Canal Trail Route.

These predictions are calculated using the above mentioned prediction normalization technique combined with a t-test for difference of means across ~900 points along the routes. The significance value for the test behind the prediction on the map is 0.05.

Thank you! I appreciate you for taking the time to read my tech notes. Please feel free to reach out to me at seanhyund@gmail.com if you have any questions or suggestions.