

AC Field Detector

Max Ladabaum x Neosensory

Materials

Adafruit Feather nRF52 Bluefruit LE nRF52832

https://www.adafruit.com/product/3406?gclid=Cj0KCQjw7sz6BRDYARIsAPHzrNILZgBPcT6tB-UBBfGiD5HbH5x_o9r0-GSERAXActK4HJMUYe0gzrgaApxpEALw_wcB

Feather Compatible 3.7V Battery

https://www.amazon.com/gp/product/B00L0W61VO/ref=ppx_yo_dt_b_asin_title_o06_s00?ie=UTF8&psc=1

10 Megaohm Resistor

https://www.amazon.com/gp/product/B07P8QG28N/ref=ppx_yo_dt_b_asin_title_o00_s01?ie=UTF8&psc=1

Copper Wire (I used 20 AWG, but anything from 15-30 AWG should work)

https://www.amazon.com/gp/product/B003UDC5UM/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1

Optional:

Jumper Wires and Breadboard

https://www.amazon.com/gp/product/B07PYW94WN/ref=ppx_yo_dt_b_asin_title_o03_s00?ie=UTF8&psc=1

Arduino Toggle Switch

https://www.amazon.com/gp/product/B07PYW94WN/ref=ppx_yo_dt_b_asin_title_o03_s00?ie=UTF8&psc=1

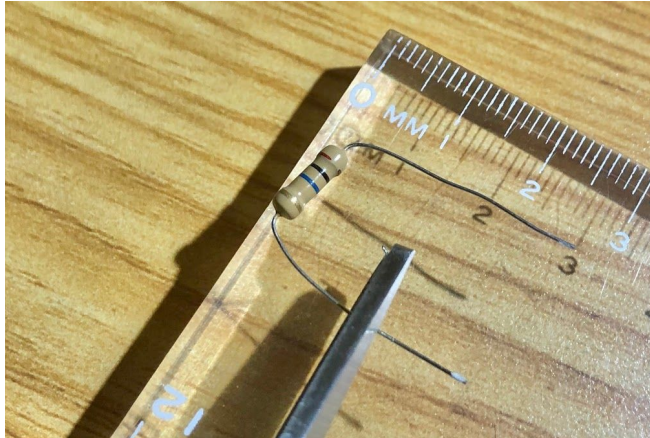
A case to put the electronics inside of. A 3D printable case that I designed is included in the "media and files" section.

An arm strap

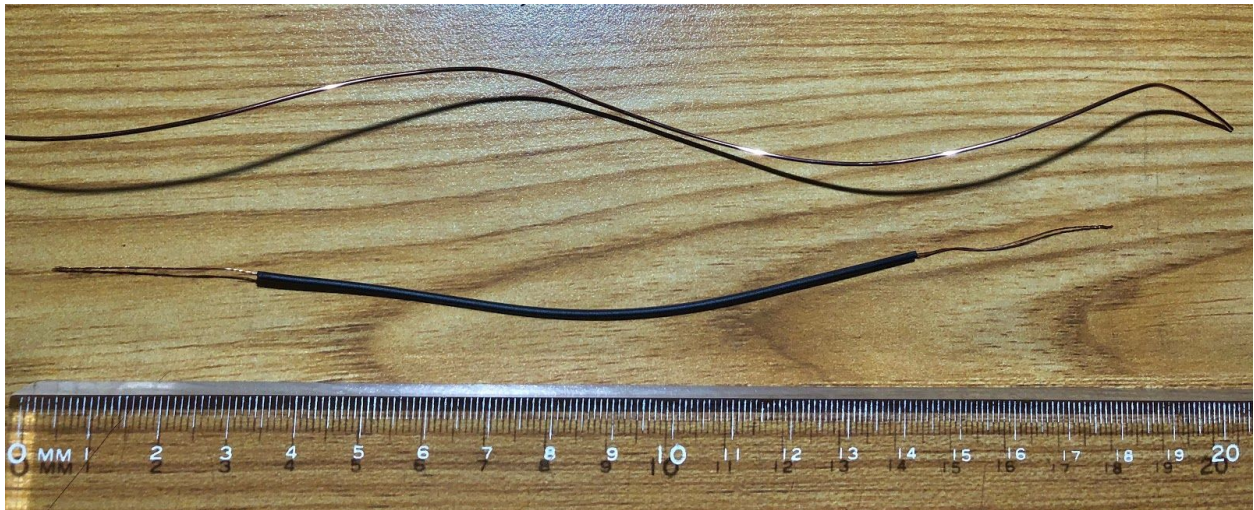
https://www.amazon.com/gp/product/B072C3SCW6/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1

Hardware Instructions

- 1) Take a 10 Megaohm resistor. Cut the ends of the wires such that there is about 1cm of wire on each side of the resistor element.



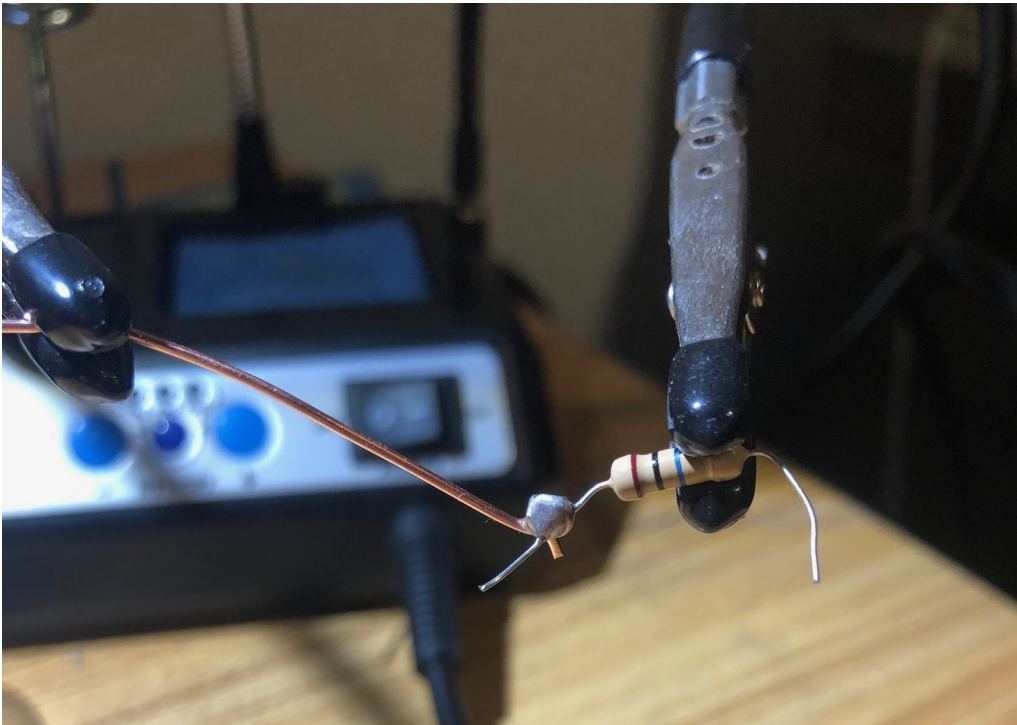
- 2) Cut a ~20cm piece of copper wire. I used 20AWG bare copper wire, but you can just strip a jumper wire if you do not have raw copper wire available. This wire will serve as an antenna.



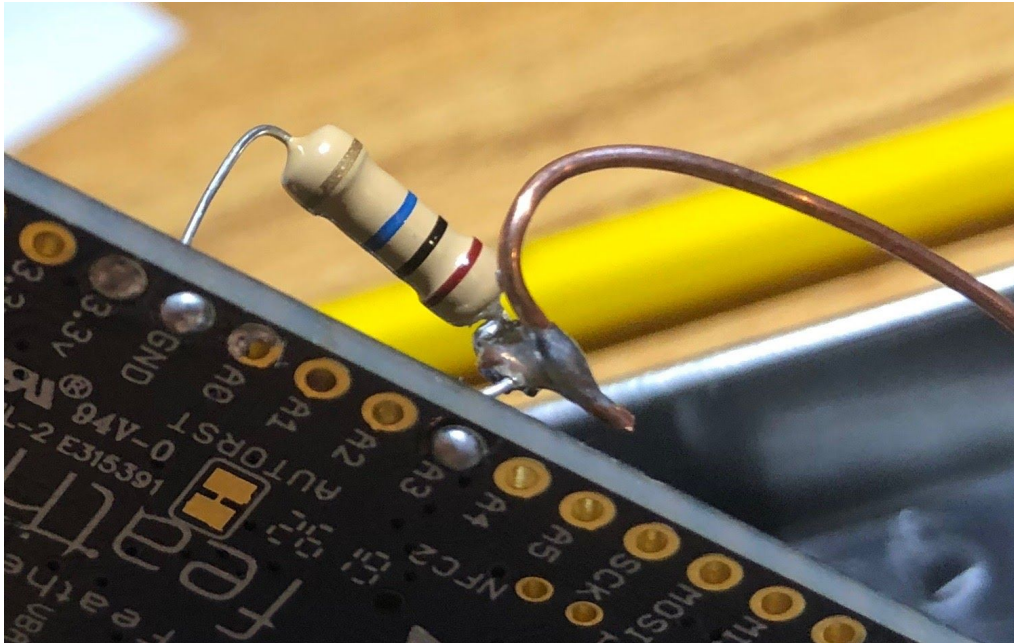
- 3) Make a tight loop at the end of the copper wire (or jumper wire). Thread one end of the resistor through the loop.



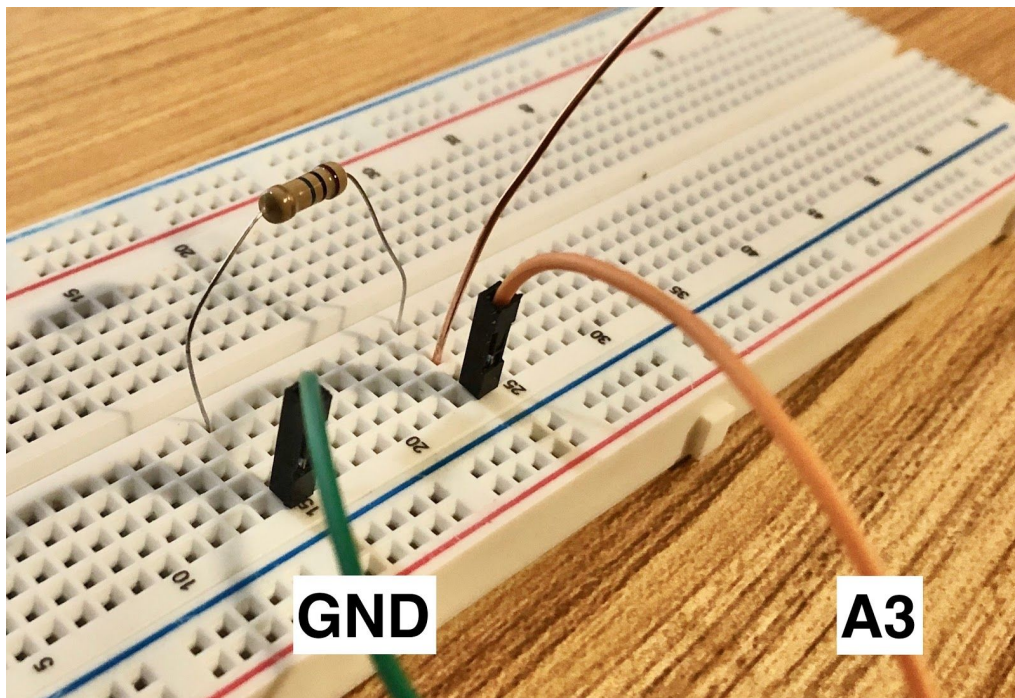
- 4) Solder the copper wire and the resistor together. Cut off the loose end of the copper wire. NOTE: If you are using a breadboard instead of soldering, go to step 6.



- 5) Solder the resistor wires to A3 and GND. **VERY IMPORTANT:** Make sure the side with the copper wire is soldered to A3.

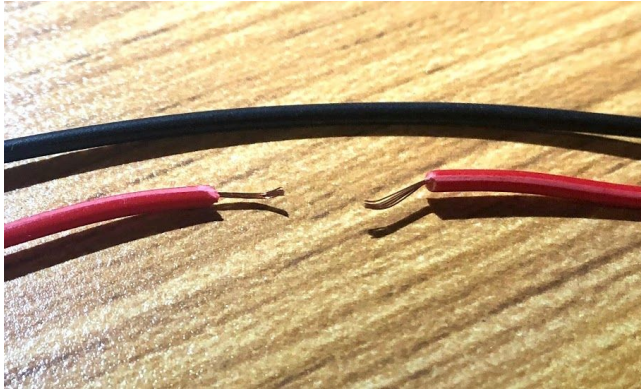


- 6) For those using a breadboard, the circuit should look like this:

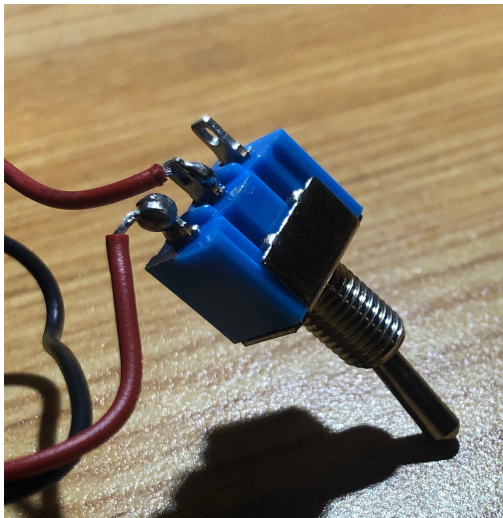


The following steps are optional

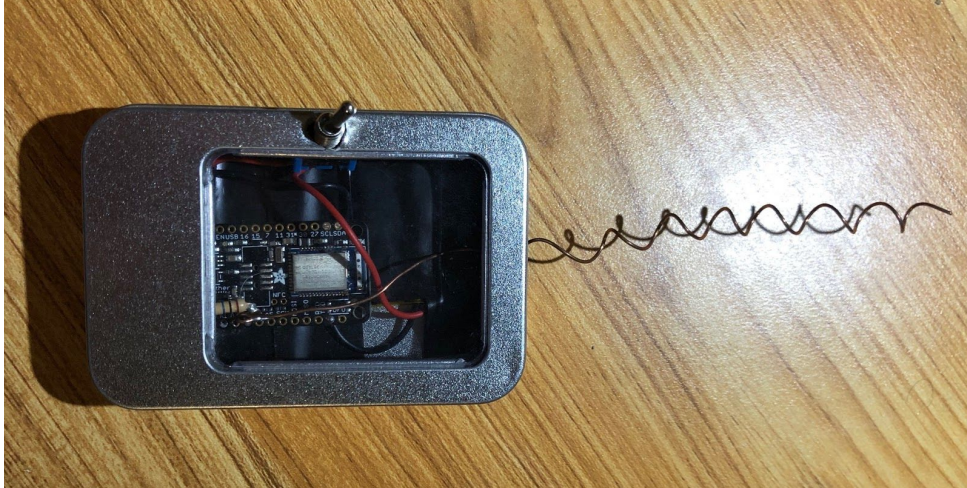
- 7) Cut the red wire from your 3.7V battery at its median. Strip both ends of the wire.



- 8) Solder one end of the wire to the middle prong of your toggle switch. Solder the other end of the wire to one of the outer prongs. This switch will allow you to turn the feather on and off without unplugging the battery.



- 9) Place the feather and the battery, along with the attached copper wire, resistor, and switch, inside of a container. I have included a 3D-printable case in the "files" section. If you don't have a 3D printer, any case will work- I sometimes like to use a playing card box. Use some insulating tape to make sure the antenna doesn't touch the feather or the battery.



- 10) You can attach the case to an armband for a no-hands experience!



Note: The case that I designed for 3D printing is running late. It was supposed to arrive from Shapeways on September 6th, but is now expected to arrive on September 12th. I will update this document with photos once the new case arrives.

Software Instructions

Calibration

The design of this device is highly sensitive to even the slightest changes in resistance. For this reason, it is important to calibrate the device before running the final code.

The code, linked below in step 2, collects data about the electromagnetic field surrounding the detector by reading voltages created by induced current in the antenna. We use `analogRead()` to assign integer values between 0 and 1023, corresponding to voltages between 0V and 3.3V on pin A3. This relationship implies a difference of 0.0032 Volts between each integer value.

Due to the extreme sensitivity of assigning integer values from `analogRead()`, every circuit will be different. Using the code from step 2, you will need to find the resting value of your circuit when there is no electromagnetic field present. You will also need to find the maximum value when the sensor is very close to a strong electromagnetic field. You can find both the resting value and maximum value by looking at the serial monitor. More details are included in the steps below and in the comments of the code in step 2.

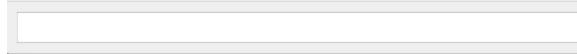
Additionally, please note that since we are building an electromagnetic field detector, we must be mindful of any sources of noise during calibration. This includes **UNPLUGGING YOUR LAPTOP FROM POWER** when calibrating.

WARNING: DO NOT ACCIDENTALLY STICK THE COPPER WIRE INTO A WALL OUTLET

- 1) Make sure you have set up the feather with the Arduino IDE. You can learn more [here](#).
- 2) Upload [this code](#) to your feather using the Arduino IDE.

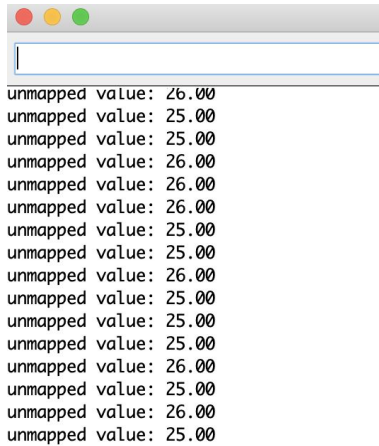
- 3) Once you have the code uploaded onto your feather, keep the feather plugged into your computer and open the serial monitor. You should see a fast stream of integer values. Each integer value represents the average of 164 `analogRead()` values. `AnalogRead()` takes 1.02 milliseconds to perform, so each integer value shows an average of 16.7 milliseconds worth of `analogRead()` values. A delay is included in the code such that a new value should be measured and displayed in the serial monitor every 164 milliseconds (~ 6 Hz). We measure a single 60Hz AC oscillation (16.7 milliseconds), then rest for about 10 60Hz oscillations (164 milliseconds), and repeat. By measuring this way, we create a beat frequency of values (more on this later).

Your serial monitor should look something like this:
(likely you will have different values)



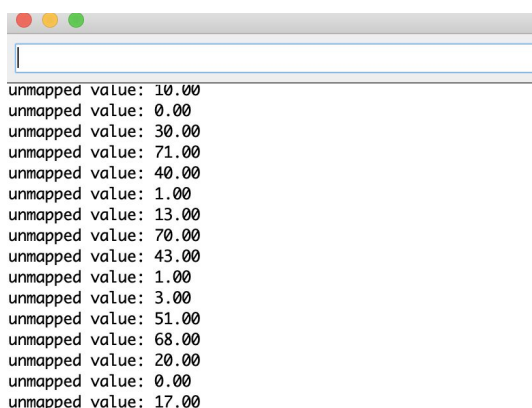
```
unmapped value: 31.00
unmapped value: 36.00
unmapped value: 32.00
unmapped value: 29.00
unmapped value: 34.00
unmapped value: 30.00
unmapped value: 26.00
unmapped value: 32.00
unmapped value: 28.00
unmapped value: 24.00
unmapped value: 30.00
unmapped value: 26.00
unmapped value: 22.00
unmapped value: 27.00
unmapped value: 23.00
unmapped value: 19.00
unmapped value: 25.00
unmapped value: 21.00
unmapped value: 17.00
unmapped value: 23.00
unmapped value: 19.00
unmapped value: 15.00
```

- 4) Now that you can see the `analogRead()` values, you will need to calibrate your device. Make sure to **UNPLUG YOUR LAPTOP** if it is charging. Move the device to a place where there is little to no electromagnetic field noise and open the serial monitor. If everything is working correctly, “unmapped value” should not fluctuate from equilibrium by more than 2. Take note of the highest “unmapped value” you see. In the case of the screenshot below, 26 is the highest “unmapped value.”
- Tip:** If the values are fluctuating from equilibrium by more than 2, try tapping the wire with your finger. This should “reset” `analogRead()`.



```
unmapped value: 26.00
unmapped value: 25.00
unmapped value: 25.00
unmapped value: 26.00
unmapped value: 26.00
unmapped value: 26.00
unmapped value: 25.00
unmapped value: 25.00
unmapped value: 26.00
unmapped value: 25.00
unmapped value: 25.00
unmapped value: 25.00
unmapped value: 26.00
unmapped value: 25.00
unmapped value: 26.00
unmapped value: 25.00
```

- 5) Replace the “_minimum_” number “26” on line 22 with the number you just found +1. In the case of the photo above, I would replace the number from line 22 with “27”.
- 6) Now, move the device to a place where you expect electromagnetic activity. I have found that TVs and power strips give off a strong signal. Lights, light switches, appliances, and wall outlets provide a medium strength signal. I recommend calibrating to a medium signal (10cm from a wall outlet or 60cm from a TV).
- 7) Looking at the serial monitor, you should now see that “unmapped value” is fluctuating. It may fluctuate very slowly, so be sure to watch the monitor for at least 30 seconds. Scroll around in the serial monitor and take note of the highest “unmapped value.” For better calibration, try holding the detector near various AC emitting devices, in case one device emits more than another.



```
unmapped value: 10.00
unmapped value: 0.00
unmapped value: 30.00
unmapped value: 71.00
unmapped value: 40.00
unmapped value: 1.00
unmapped value: 13.00
unmapped value: 70.00
unmapped value: 43.00
unmapped value: 1.00
unmapped value: 3.00
unmapped value: 51.00
unmapped value: 68.00
unmapped value: 20.00
unmapped value: 0.00
unmapped value: 17.00
```

- 8) Replace the `"_maximum_"` number `"40"` on line 23 with the number you just found. In the case of the photo above, I would replace `"40"` with `"71"`.
- 9) Upload the modified code with your new calibration numbers. Remove the `"//"` on lines 55 and 56. Upload the modified code with your new calibration numbers to the feather. You should see in the serial monitor that `"mapped value"` is 0 when there is no electromagnetic field noise, and oscillates closer to 1 as electromagnetic field noise increases.
- 10) Write down the `"_minimum_"` and `"_maximum_"` values that you found in steps 4-8. Make sure to keep track of these values along with the defined value of `"sample"` that you are currently working with.
- 11) Change the defined value of `"sample"` to 162 by removing `"//"` on line 2 and adding `"//"` on line 1. Upload the new code. Repeat steps 4-10.
- 12) Change the defined value of `"sample"` to 166 by removing `"//"` on line 3 and adding `"//"` on line 2. Upload the new code. Repeat steps 4-10.
- 13) Congratulations, the device is now calibrated!

Buzz Code

Now that we have calibrated the electromagnetic field detector, it is time to implement the code that will allow us to send data to Buzz.

- 1) Make sure you have downloaded the NeosensoryBluefruit library. You can learn more [here](#).
- 2) Upload [this code](#) to your Arduino IDE.
- 3) Update lines 32-38 with your own `"_minimum_"` and `"_maximum_"` values found during the calibration steps. `"min_low"` and `"max_low"`

correspond to "sample 162" values. "min_mid" and "max_mid" correspond to "sample 164" values. "min_high" and "max_high" correspond to "sample 166" values.

- 4) Upload the modified code to your feather.
- 5) Connect Buzz over BLE. You should now be able to sense electromagnetic fields!

Tips

If you are feeling small, constant vibrations when you shouldn't be, try raising all of the "_minimum_" values by 1 or 2.

If you aren't feeling vibrations until you get extremely close to AC emitting objects, try adjusting the antenna. I have found that forming a large loop with the antenna improves sensitivity. You can also try lowering the "_minimum_" and "_maximum_" values.

If you are feeling long, drawn out pulses, there is nothing wrong. Due to the nature of wave interference, you may feel long pulses if one of the "sample" values we used is extremely close to the actual AC frequency you are measuring. You can learn more about this effect through the analogy of beat frequencies, linked [here](#). We also discuss this in the "Theory" section.

If you feel seemingly random vibrations, make sure that you are not accidentally touching any part of the circuit with your hand. (For a fun test, try tapping the wire with your finger and see how Buzz vibrates!) You may also encounter unwanted electromagnetic noise if you are using a metal case to enclose the electronic components.

Theory

When I first set out to create an electromagnetic field detector, the first thing I looked into was an Adafruit triple axis magnetometer. The magnetometer worked for sensing extremely large electrical activity at a short distance, but it couldn't sense anything more than 5 inches away. I kept working with the magnetometer, but I was not satisfied with its performance.

A few days later, I was messing around with an oscilloscope in the university research lab where I work. I noticed, not for the first time, that the oscilloscope was picking up some noise from the various power cords lying around. As I moved the open-ended cable plugged into the oscilloscope closer to the power cords, the noise signal got stronger. Could this work as an electromagnetic field detector????

I started searching for examples of Arduino based EMF detectors that used a similar mechanism as the oscilloscope. There were a bunch of stupid articles about using EMF detectors to locate ghosts, so it was challenging to find anything. I eventually stumbled upon [this article](#) by Aaron Alai. The article outlines an EMF detector made out of only a wire, a resistor, an LED, and an Arduino. After messing around with different resistor values, I was able to get Aaron's design to work on the Adafruit nRF52 Feather.

Now, what to do about Buzz...

To begin, I took values that were picked up by the antenna, mapped them between 0 and 1, and sent the intensities individually to each Buzz motor using the `vibrateMotor()` command. This worked fine as a rudimentary EMF detector, but it did not have good enough fidelity to qualify as a sense. I was able to tell when there was weak, strong, or no electromagnetic noise, but I was not able to feel anything about the frequency or intensity of the signal. Additionally, due to timing issues (discussed later), Buzz was only being sent about 1 in every 200 `analogRead()` values. This caused my wrist to feel vibrations that were unrepresentative of the actual EMF values. For context, the code operated as follows:

- `analogRead()` collected ~10 values every millisecond

- `vibrateMotor(0, mapped_analogRead_value)` was called. "mapped_analogRead_value" was a direct mapping of a single `analogRead()` value, causing it to change 10 times per millisecond. As soon as `vibrateMotor` was called, the Arduino selected the most recent `analogRead()` and sent it to motor 0.
- `vibrateMotor(1, mapped_analogRead_value)` was called. The most recent `analogRead()` value was sent to motor 1.
- `vibrateMotor(2, mapped_analogRead_value)` was called. The most recent `analogRead()` value was sent to motor 2.
- `vibrateMotor(3, mapped_analogRead_value)` was called. The most recent `analogRead()` value was sent to motor 3.

As you can see, this code was extremely flawed. If I were to have a 2 millisecond long stream of readings from the antenna in a low noise area, say "25, 45, 20, 29, 21, 27, 35, 43, 20, 14, 25, 27, 21, 40, 21, 24, 31, 18, 24, 27" Buzz could be sent any of the 20 numbers depending on the exact 10th of a millisecond when it was called. On the other hand, if I were to move into an area with strong EMF noise, a I would pick up a new 2 millisecond long stream of readings, say "45, 51, 32, 31, 43, 21, 53, 56, 35, 39, 21, 42, 44, 49, 53, 61, 48, 48, 43, 29." Just like before, Buzz could be sent any of these 20 numbers.

In this hypothetical scenario, the average of all of the readings from a low noise area would be 26.6, and the average from the high noise area would be 42.2. This difference in averages is significant enough to differentiate between high and low noise areas. That said, individual readings from low noise areas overlap with individual readings from high noise areas. For this reason, individual readings are terribly inaccurate.

I tried to do a FFT on the individual `analogRead()` data, but it was too noisy to get accurate frequency readings. At times on the order of <5 milliseconds, I believe that many frequencies picked up by the antenna are just radio noise (or possibly microwave background...).

After realizing how noisy my signal was at low timescales, I decided that I needed to take an average. (Buzz has a maximum fidelity of 16ms anyway, so what use is a 0.1ms signal.) I decided to average 16ms worth of signal, corresponding to ~160 `analogRead()` values.

The 16ms averages turned out to be quite a bit more accurate than the direct `analogRead()` signal, but they still felt a bit random. After noodling around for a while, I realized that $1\text{sec} / 0.016\text{sec} = 62.5$, meaning that I was collecting a 60Hz signal at 62.5Hz... The "randomness" of the 16ms

averages was actually a $\sim 2.5\text{Hz}$ beat frequency. Given the sheer number of frequencies involved, (BLE frequency, Buzz frequency, Arduino frequency, AC frequency, etc...), I will spare the math. (Feel free to email me at maxladabaum@gmail.com if you want to know more about the math.)

I decided that I should lower the averaging number to something like 50 `analogRead()` values, and see if a FFT was viable. The FFT did work, but it just reported 60Hz all the time- BORING! I tried sending a combination of frequency and intensity to Buzz, but the result was still extremely boring since the frequency dedicated motor never changed. Additionally, I didn't like the fact that small surges and other interesting phenomena could be covered up by averaging.

Aside

From a design standpoint, I believe that the AC field detector should be as flexible of a tool as possible. Theoretically, if I design the AC field detector with only one purpose in mind, then users may miss out on sensing phenomena that my application is not specifically designed to expect. In other words, if I design the detector to only feel phenomena that I already know exist, then phenomena that I don't know exist, and phenomena that have not yet been discovered, will never be sensed.

For example, imagine that there is an unknown physical phenomena, something like a switch that surges for 3 milliseconds when it is flipped on and off. If I were to design the AC field detector such that it could only detect 60Hz frequency and intensity over long periods, a 3ms surge might be completely covered up. In this hypothetical scenario, an electrician wearing a Buzz with an AC field detector would never discover that the switch was causing tiny surges.

To further illustrate the concept of "sensory flexibility," as I like to call it, imagine a version of Buzz designed exclusively to listen to human voices. Deaf users would hear voices very well, but they may never know that their keyboard clicks when they type, or that they chew loudly. In this scenario, users would form a misrepresentative view of sound and think that only human voices make noise. On the other hand, if the "voice exclusive" version of Buzz were to be advertised as a voice sensor, rather than a sound sensor, deaf users would understand that other things do indeed make sound, but just do not register with the voice exclusive version of Buzz.

Following this notion, I am weary to advertise my project as a fully fledged electromagnetic field detector. Users may think that if their Buzz

doesn't vibrate, there is no electromagnetic field noise. In reality, there may be electromagnetic field noise, just not at a frequency that my device can detect. I do not want to give people an incorrect sensory intuition about electromagnetic fields, so I think it is appropriate to refer to my device as an AC detector or electrical grid detector, rather than as an electromagnetic field detector.

I got a bit into the weeds back there. Anyway, the bottom line is that I want this device to be as flexible as possible. Most of the signal processing design should be done with "sensory flexibility" in mind. I want to put very few bounds on what phenomena the user can feel- data sent to Buzz should be as raw as possible.

Theory behind the final setup

After testing various FFT, intensity, and a few other types of mappings, I decided that a moving average mapping was the best way to communicate electromagnetic field information through Buzz.

What is a "moving average mapping?" you may ask. A moving average is defined as a succession of averages derived from successive segments (typically of constant size and overlapping) of a series of values. In the case of my program, an average value is created over a 16.5 millisecond period. This value is then sent to Buzz, where motor 1 vibrates for 182 milliseconds (166ms delay increment + 16.5ms collection time) at an intensity corresponding to the average value. 162ms after the first average value has been collected, a new 16.5ms average value is collected. Buzz updates the motor 1 intensity to the updated average value, and the motor runs at the updated intensity for 182ms. (If you read carefully, you will notice that there is a 4ms overlap- this overlap contributes to the creation of beat frequencies.)

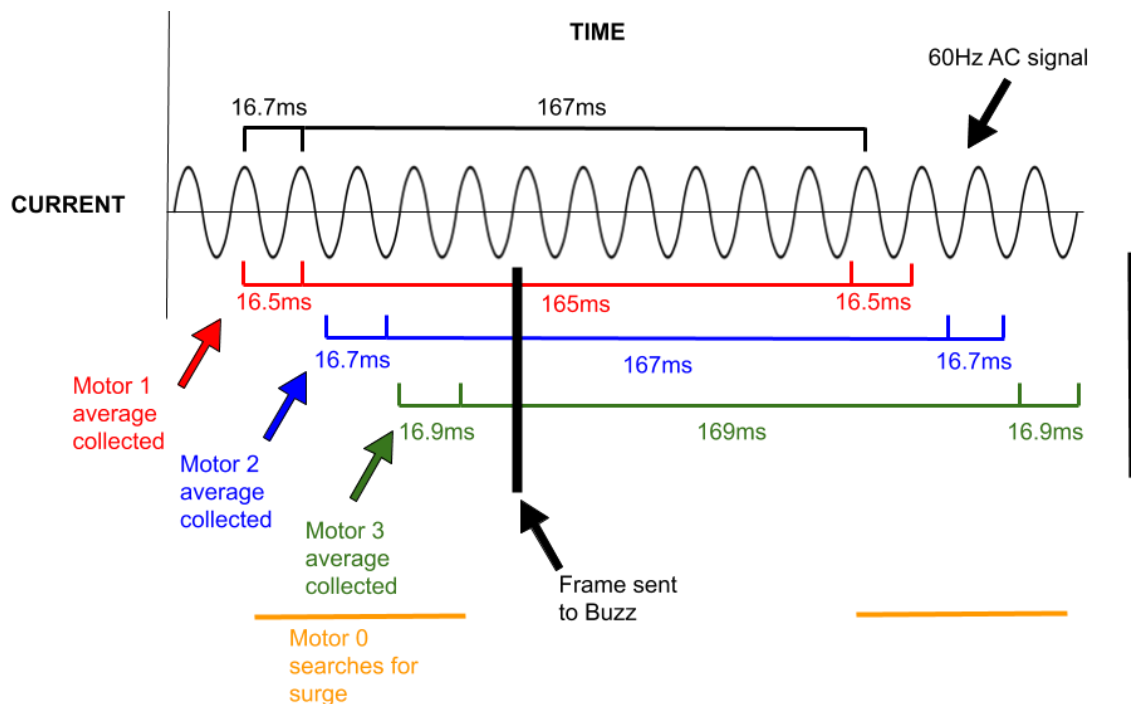
Motor 2 is run on the same mathematical principle, except for that its average value is created over a 16.7 millisecond period. The value is then sent to Buzz, where motor 2 vibrates for 182 milliseconds (166ms delay increment + 16.7ms collection time).

Motor 3 is run on the same mathematical principle, except for that its average value is created over a 16.9 millisecond period. The value is then sent to Buzz, where motor 3 vibrates for 182 milliseconds (166ms delay increment + 16.9ms collection time).

Motors 1, 2, and 3 display individual beating patterns in relation to how close they are to the actual AC frequency. Motor 1 collects data at 60.5Hz, motor 2 collects data at 60Hz, and motor 3 collects data at 59.5Hz. If an AC frequency is irregular or slightly off from 60Hz, a user should be able to intuitively feel the frequency change as the beat patterns change on motors 1, 2, and 3. In theory, this method of frequency identification through beats should be viable up until the beat frequency (absolute value of $AC_frequency - data_collection_frequency$) is no longer perceivable through the skin.

Motor 0 acts as a surge identifier. Motor 0 vibrates if the program finds any freakishly large `analogRead()` values that get constrained or silenced through the averaging process. Motor 0 can pick up a surge that is 0.1ms long. However, there are some points in the loop where no `analogRead()` values are being collected. During these periods, none of the motors, not even motor 0, will be able to pick up a signal. This is mostly a bluetooth issue, and could be fixed with directly integrated Buzz hardware.

Here is a graphic attempting to illustrate how the signal is mapped:



The idea behind this method of mapping is that users will be able to feel:

- Average electromagnetic field intensity mapped to the intensity of motors 1, 2, and 3.
- AC frequency mapped to beat patterns on motors 1, 2, and 3.
- Surges mapped to motor 0

Additionally, this form of mapping is nice because:

- Human brains are good at extracting information from overlaid frequencies (think about hearing and sight). Buzz is essentially just a sensory interface for touch, just like screens are an interface for sight and speakers are an interface for hearing. Screens communicate with RGB pixels, creating a complex image out of different intensity levels of a few light frequencies. A speaker creates a complex sound by combining multiple individual sound frequencies which your brain processes into a note. Following this idea, it seems natural that Buzz should communicate data by combining beat frequencies. You can think of this a bit like adding two colors together, or playing a chord on an instrument.

Outreach

A few of my friends from highschool work with dangerous amounts of electricity. I asked them if they would be open to answering some questions about the Buzz AC detector, and they were happy to talk. I talked to my friend Levi, a lineman, my friend Landon, a groundman, and my friend Jay, who constructs solar arrays.

I did not record my conversation with Levi. Levi's advice was that I promote the product through electrical unions. He said that unions have a lot of power, and that unions usually use their leverage to force companies to buy safety equipment for the employees. He also said that there are already ways to detect if a wire is live, (voltmeters and ammeters), but that he still thought the idea was good because many linemen get lazy about safety and do not double check if the line is live. He also noted that freak accidents can occur if civilians plug in strong generators and blow a transformer when lines are turned off for maintenance.

Levi stated that he did not think the Buzz AC detector should replace normal safety checks, but he did think it would be useful as an additional

safety check. His main assertion was that, when followed correctly, lineman safety protocols work well. However, many linemen fail to perform all of the tedious checks, and that's when accidents happen. Levi felt that the Buzz AC detector would serve as a last line of defense if linemen failed to do their safety checks correctly- the idea being that Buzz would vibrate before a lineman could accidentally touch a live wire that they nonchalantly assumed was off. (A quick, back-of-the-envelope calculation shows that a lineman's wrist should start buzzing a few feet away from a live wire.)

I recorded a short recap of my conversation with Landon. Here is the link to the recorded summary of our conversation: [listen to the summary](#).

It is useful to note that Landon was visiting for the weekend, so he actually got to try out the device.

I did not record my conversation with Jay. Jay's comments were somewhat similar to Levi's. Jay basically said that he thought the Buzz AC detector was a great "final safety check," but that he would not want it to replace current protocol. Jay expressed that even when he performs correct safety checks with voltmeters and ammeters, he is still worried that he may have made a mistake in his check. He expressed that a mistake could easily cost him his life, so he would always be interested in a device that improves safety.

Jay also mentioned that there is some paranoia surrounding the possibility that a line could get turned on after safety checks have already been performed. This usually occurs because of a communication issue between multiple workers (this is more of a concern for solar array workers, but not so much a concern for lineman). Since workers cannot be constantly performing safety checks, they have to trust that other workers have not turned on a line that was off minutes earlier when the safety check was performed. Buzz could help solve this issue, since the wrist vibration format does not require the worker to stop their work and take out a voltmeter/ammeter to test a line. Following this notion, Buzz would allow workers to be "constantly performing a safety check," and could thereby prevent accidents caused by previously tested lines being turned on.

Finally, I messaged my former highschool physics teacher about potential applications in the education sector. My teacher, Mrs. Heintz, said that she thought the Buzz AC detector would be a great tool for helping students learn about electromagnetism. She expressed that she thought the

price point was too high for teachers to purchase Buzz with their individual classroom budgets, but that district funding could still be on the table.

Despite proposing household electricians as a potential market for the Buzz AC detector, I was not able to talk to any electricians. I can say from my own testing that I am able to feel if a lightbulb is on, high dim, low dim, or off, and that I am able to feel if various appliances are plugged in.

Media and Files

Custom belt clip case (3d print file): [here](#)

Video showing extremely long beat pattern: [here](#)

Video showing my Preliminary Arduino EMF detector: [here](#). The detector flashes an LED instead of sending commands to Buzz.

Video showing the Buzz AC detector in action: [here](#)

Conversation with Landon: [here](#)

Github: https://github.com/maxladabaum/neosensory_emf_detector

Moving Forward

In order to eventually bring this product to market, a few things need to take place:

First, Buzz needs to be modified to contain an antenna that can pick up 50-60Hz. I do not think such a modification would be extremely expensive or time consuming, as modern antenna technology is very advanced. Cheap antennas with very small form factors are easy to come by.

Second, better software needs to be written. This should not be too challenging either, since the main problems with the current software are due to a) my low level of C++ knowledge b) bluetooth complications.

Luckily, these are not fundamental issues like the bit-rate being too large to transfer through skin, or issues related to sensory correlations being inconsistent. An additional upshot here is that antenna signal is very similar to microphone signal, so I assume most of the microphone data processing currently performed on the Buzz hardware could be tweaked to process antenna data.

Third, electrical unions would need to be contacted. The negotiation could be hard, but at least we would not have to deal with private distributors. As I see it, safety is an easy sell for unions, workers, and utility companies. Unions don't want to see workers die and get injured, workers don't want to die or get injured, and utility companies want to increase job safety so that they can pay workers less (lineman salary is high because it is one of the top 10 most dangerous jobs in America).

Finally, legal work may be the biggest hurdle. I have never dealt with anything in the safety sector, but I would assume that providing safety equipment is a huge liability.

Thank you!!!!
Max Ladabaum

Feel free to email me at maxladabaum@gmail.com or call me at 650-421-6268 with any questions!