Engr 523 - P3 - Audio Detection

Code Due: 11:59pm, February 15, 2019 Version: 2019.0

Demo Due: February 20, 2019

Goal

For this project, you will use an embedded system to first capture and then transmit audio data to a server using a UDP connection. You will also use filtering and signal power to determine if a given frequency is present. Finally, you will transmit a binary decision back to the Argon to be displayed on an LED.

Background

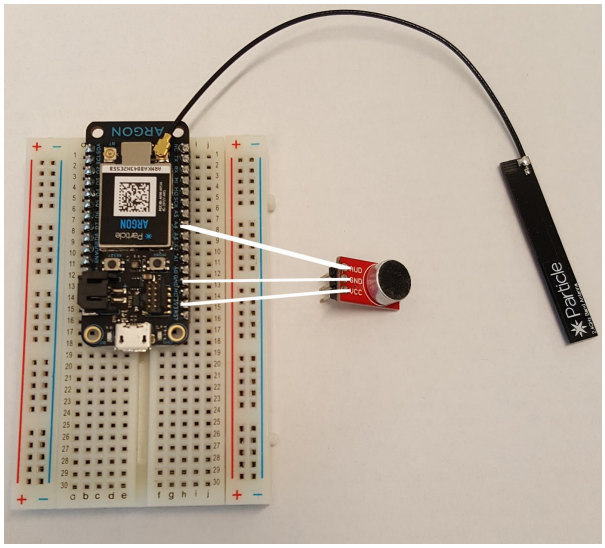
Microphone Setup

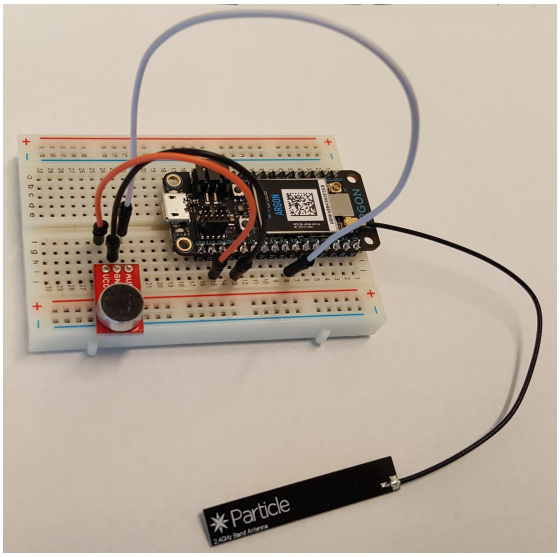
You should have received a microphone in class. If not, please see one of the TAs to acquire a microphone. If your microphone is new, you will need to solder three header pins to your microphone. There are a number of how-to-solder videos online. The TAs can help you access the FAB_LAB where soldering irons are kept.

Once the header is attached, you will need to connect the microphone to the Argon. The connections are shown below. For this, you will need to acquire at least 3 wires from the TAs.

Microphone	Argon
AUD	A4
GND	GND
VCC	3V3

These connections are demonstrated in the photos below.





Capturing Audio

With your microphone physical connected, we can now extract audio data from the microphone. To extract an individual sample, you will need to use the `analogRead()` function within your Argon code.

Documentation and a reference example of `analogRead()` can be found here:

https://docs.particle.io/reference/device-os/firmware/argon/#analogread-adc-

Recall that WAV data is not a single point, but a series of points equally spaced in time. Therefore, you will also need to cause a slight delay between individual samples (calls to `analogRead()`). For this, you will find the `delayMicroseconds()` function helpful. It's documentation can be found here:

https://docs.particle.io/reference/device-os/firmware/argon/#delaymicroseconds-

Another, slightly more accurate method for microsecond-scale delays is to manually track the number of elapsed microseconds using the `micros()` function. Then you can do the math to determine when the appropriate number of microseconds has elapsed.

https://docs.particle.io/reference/device-os/firmware/argon/#micros-

Setting LEDs

For this project, you will need to set the D7 LED on or off. The D7 LED is in the top right-hand corner of the Argon (near the USB port). For instructions on driving (or setting) the LED's value, please refer to the Particle documentation for an example:

https://docs.particle.io/reference/device-os/firmware/argon/#digitalwrite-

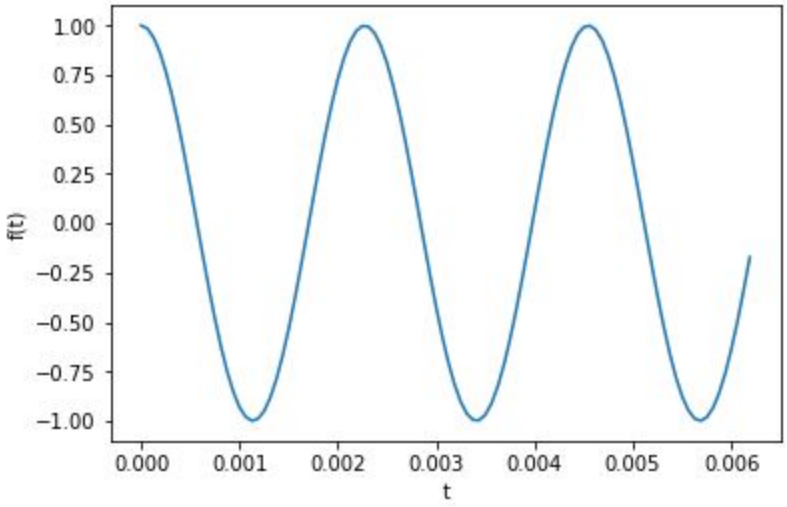
Low-Pass Filter

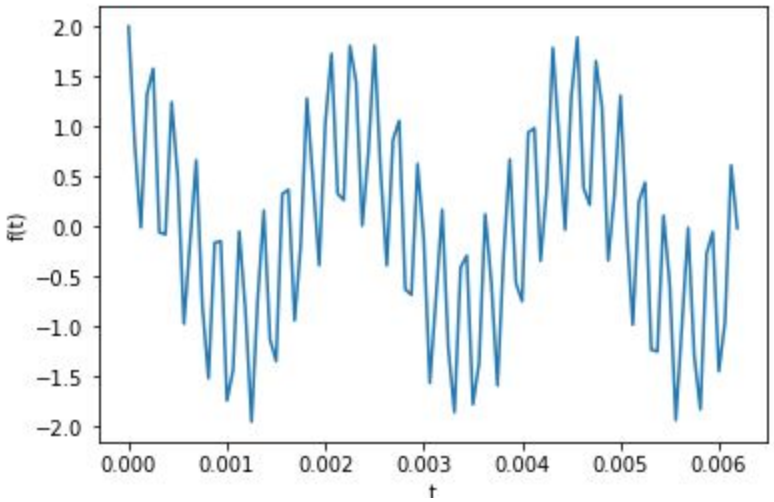
A low pass filter is used to cut out high frequency components from a time-series signal. There are various variants of low pass filters but a very simple variant of it is a moving average filter. Moving average filters (or rolling average filters or boxcar filters) helps to filter out short time fluctuations from a time-series data. It is a very simple approach, in which, as the name suggests a mean of neighbouring samples from a time-series data is used to estimate signal sample.

```
Y[i] = sum( Y[i-N : i] ) / N
```

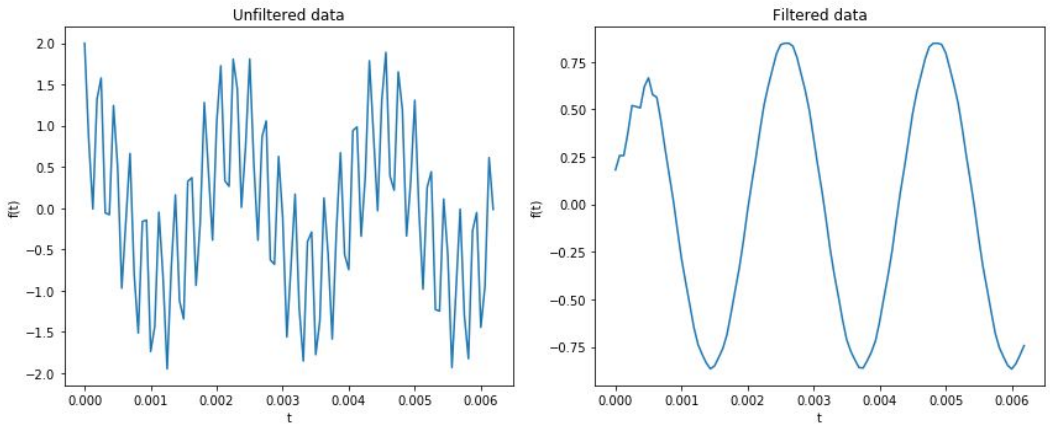
In the above formula N is the number of neighbouring samples that we are averaging over, in other words it is the filter length. A moving average filter is type of a FIR (Finite impulse response) filter, as successive outputs depend only on a fixed (or finite) number of previous inputs.

Time series data is often polluted with extra high frequency signals, called **noise,** and filtering is a very simple but an effective method of removing the noise. Here is a small example to demonstrate it. I generated a 440 Hz sinusoidal signal and added some noise to it by adding a very high frequency component to it. Here is the waveform of my clean 440 Hz and the noisy signal respectively.





Now I passed the noisy signal through a moving average filter, and out pops the denoised time-series, which is shown as below.



For all practical purposes, the audio that you will be recording will have high noise added to it due to mechanical imperfections (the dangling wifi-antenna, etc..). You can denoise your signal by using a simple moving average filter.

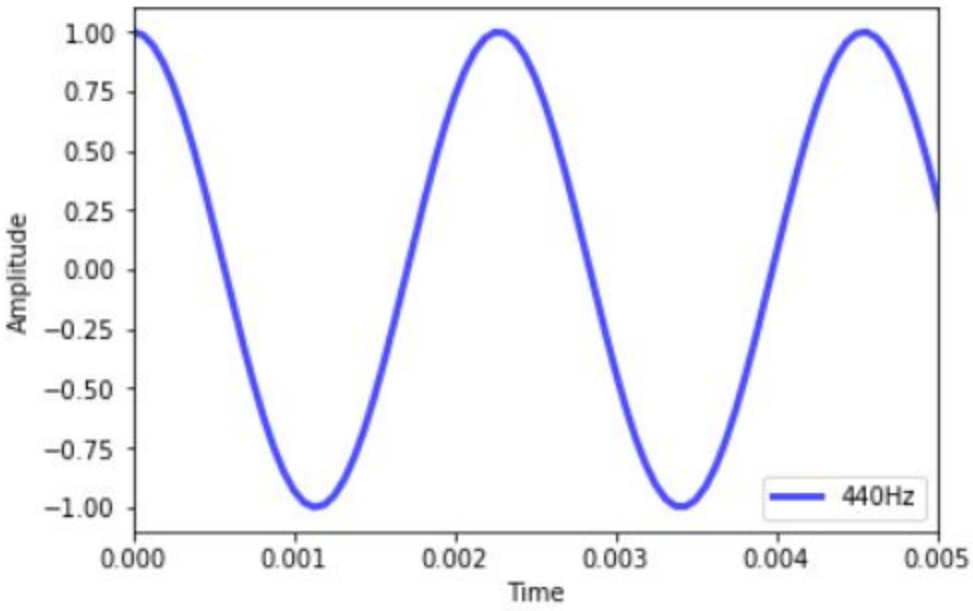
Signal Power

To calculate the “power” of a signal, we will be using a variant of the Root-Mean-Square (RMS) algorithm.

The calculation for for RMS on a list of input values, `xs = [x1, x2, x3, …]` is:

$$x_{\text{rms}} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \cdots + x_n^2\right)}.$$

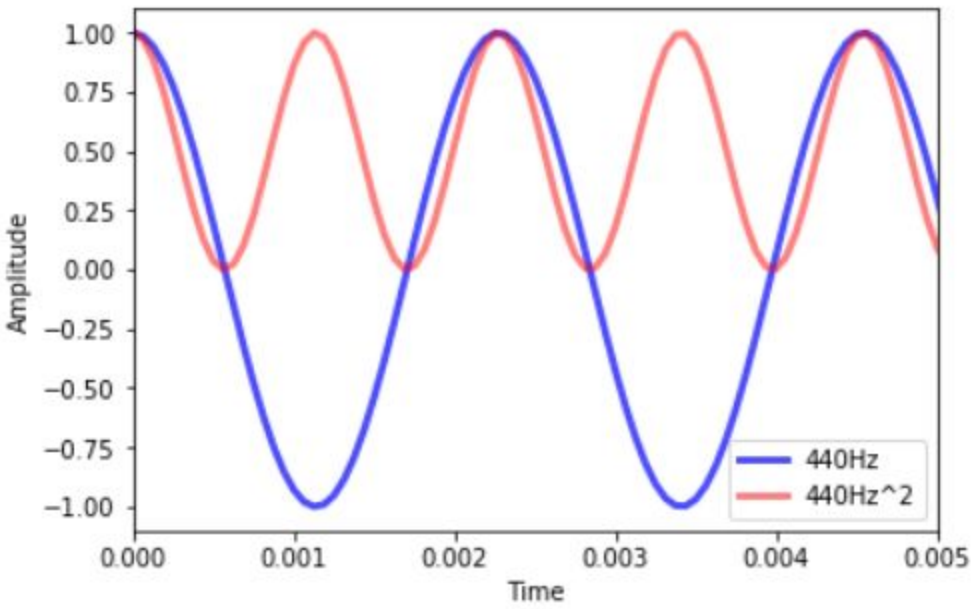
Unlike what the name suggests, you actually compute RMS backwards, Square->Mean->Root. To illustrate this graphically, let’s start with a 440Hz Cosine wave:



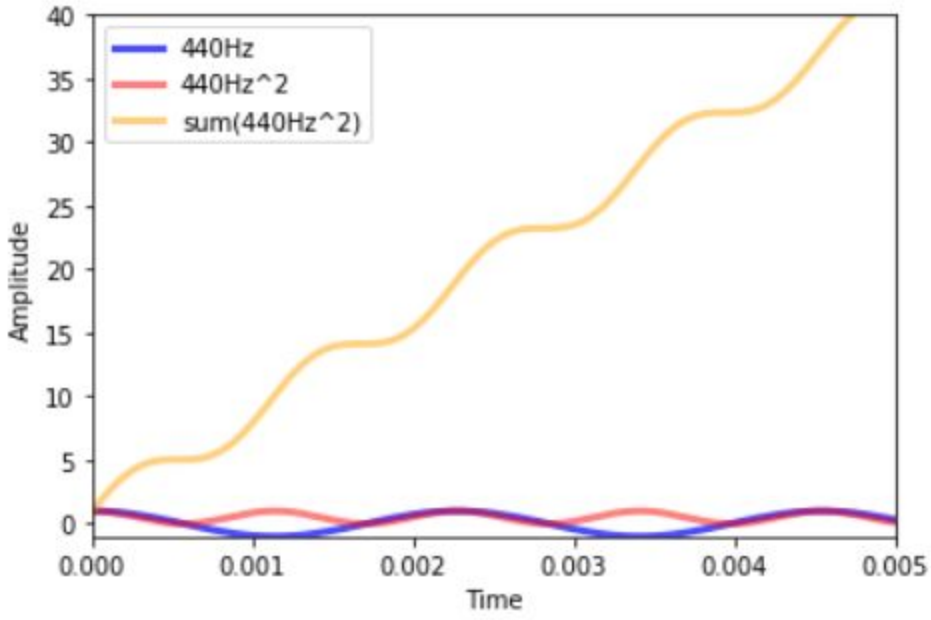
Notice how it oscillates above and below 0. If we add the values up, we’ll just get 0. The first step, Square, is to fix that. We square all of the points in our array. In python, this would be something like:

```
xs2 = xs ** 2
```

The result of squaring our input is something that oscillates twice as fast, but only above 0:

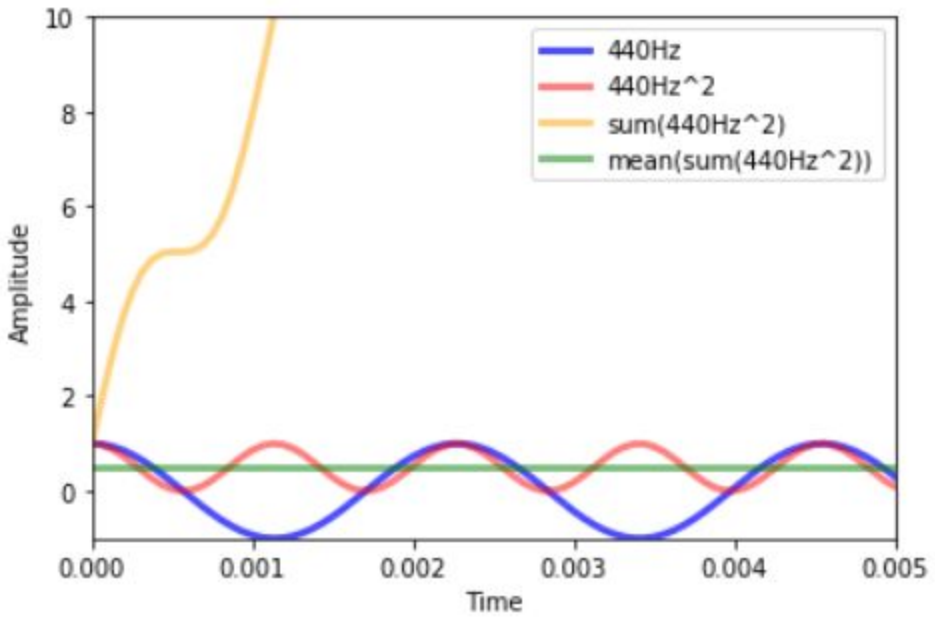


Next we need to compute the mean. That involves two parts, computing the sum, then the mean. The sum of all xs is computed by adding all the squared xs:



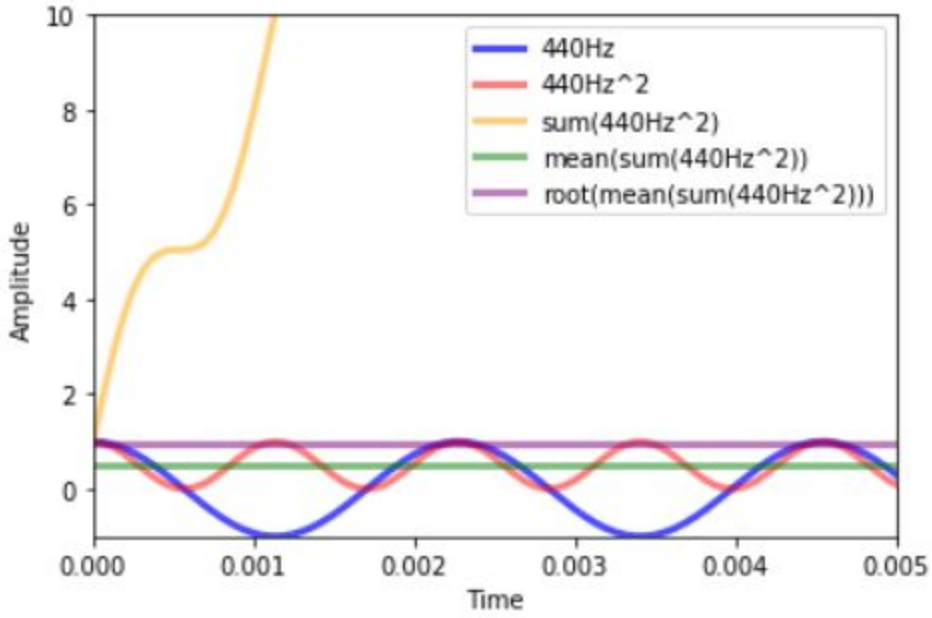
Once we have the sum, we can take the average by dividing by the number of points:

```
mean_square = sum(xs2) / len(xs2)
```



The final step of RMS is “Root”, or computing the square root of the mean. This leaves us a single value that is a measure of the “signal power”, or area under the curve of the signal.

```
rms = numpy.sqrt(mean_square)
```



For our project, we’re going to do a ~~R~~MS, or just MS. This means you only need to compute the mean of the square. So you will need to square all your WAV audio values, and compute the mean of the squared values. That will produce a single number, the mean.

To determine if there is a sound or not, you simple compare that number to a threshold:

```
def is_sound( mean_square):
      if (mean_square >= threshold): return “440Hz”
      else: return “no440Hz”
```

Assignment Description

Your assignment will be to program your argon to do the following:

1) **Argon: Capture the “MODE” button press**
2) **Argon: When the “MODE” button is pressed begin recording 16KHz audio data**
3) **Argon: Record a total of 1 second (160000 samples) of audio data**
4) **Argon: Stop recording, and transmit the recorded audio data to iot.lukefahr.org**
5) **Server: Save the raw audio data as raw.WAV**
6) **Server: Perform a 10-element (N=10) sliding-window low-pass filter on the audio data as described above.**
7) **Server: Save the post-filter audio as filter.WAV**
8) **Server: Perform an power-calculation on the audio data**
9) **Server: Print the MS value to the terminal on the server**
10) **Server: Threshold the power levels, determine yes440Hz / no440Hz**
11) **Server: Print the yes440Hz/no440Hz result to the terminal on the server**
12) **Server: Transmit yes440Hz/no440Hz back to the Argon**
13) **Argon: Listen for the response from the Server**
14) **Argon: If the server responds that the 440Hz has been detected, turn on the D7 LED (upper right corner, next to the USB port). If 440Hz is not detected, turn off the D7 LED.**

We encourage you to reuse as much code from P1 and P2 as possible.

To test your code, we have provided two sample WAV files to play while capturing the audio data. The first is tone_440Hz.wav which is a 440 Hz tone. The other is 4400Hz.wav, which is a 4400 Hz tone. These files are available on the Downloads page of the website.

Grading

The grading for this assignment will be based on these milestones:

- Correctly connecting your microphone (10%)
- Correctly capturing and transmitting audio data via UDP from the Argon (30%)
- Correctly filtering the audio data on the server (30%)
- Correctly computing the signal power on the server (10%)
- Correctly displaying yes440Hz/no440Hz with the Argon's D7 LED (20%)

The points for these milestones are all dependent on correctly following the submission instructions.

Code Submission

For this project, we will be switching to a code submission + demo, rather than code submission with video.

All code should be developed in a private IU Github repository where *the users susom, **ccfernan** and lukefahr are added as collaborators*. You must convey which version of your code is your "submission" by submitting the link to the commit on IU Github. This link will look similar to

https://github.iu.edu/SOIC-Digital-Systems/Spring-2017/commit/269670d11d10decb79905ff3cb4ba456a9c928c0

and when followed should lead to the commit summary page.

The submitted commit must contain:

- Your code, obviously. Please include a 'server' subfolder which contains the server-side code, and a 'client' subfolder for the Argon code.

- A Readme.md. This will specify a set of instructions for how to build and/or run the project, and we suggest that the process be automated via makefiles or similar tool. The readme must also contain the name and email of your partner.

Project Demonstration

For this project we are not having you go through the trouble of creating a demo video but instead, **you will be required to show a demo of your working project to one of the TAs during their office hours by the Wednesday following the code submission due date.** The exact date will be posted at the top of this document.

References:

- Wiki-page of Moving average filter.
- Wikipedia article on RMS, https://en.wikipedia.org/wiki/Root_mean_square.