Mains Hum and Audio Evidence Comparison Tool

Andrew Bajumpaa

Table of Contents

Link to Tool

Code for the tool, as well as documentation can be found at https://github.com/racoltdev/MainsHum.

Goal

The goal of this paper is to implement an audio extraction and comparison tool that can be applied to plain audio and video formats. Standard audio files often contain inaudible forensically valuable information. In this case, most of the data in an audio file, such as an mp3 can be considered noise. A band-pass filter can be applied across a file to remove all the noise and extract information on only certain key frequencies that may provide useful information. A known use case for this is to extract the background hum of electrical equipment, determined by the mains power, from evidence samples to determine the exact time when a file was recording[1]. This paper will also implement tools for extracting other frequency bands and comparing against other known backgrounds aside from just the mains hum.

Scope

The implementation provides a documented Command Line Interface (CLI) for interacting with it, as well as an installation guide and troubleshooting section. All information regarding usage of the tool is documented in the README.md file in the linked github.

The tool implements two modes of use. The first is for extracting frequency bands from samples. This is useful in the case that an investigator does not have access to the background signal that they would like to compare evidence against. The second mode is comparison, which compares an unaltered audio sample to a known background signal to determine if and where an audio file being examined as evidence matches with the background.

In addition to simply processing the data, the tool also produces a graph of the extracted signals, as well as its interpretation of the signal when in compare mode, so a user can visually compare parts of the signal manually.

Methods

Extraction

This implementation primarily analyzes .wav files. Although it accepts other formats as input, internally it will convert all data into a .wav representation using ffmpeg[2]. Some important characteristics about .wav files are needed to understand the methodology in this paper. A .wav file contains two important kinds of data. The first is the sample rate, and the second is a series of sample points over time that can be interpreted as the amplitude of audio data. While .wav files are a discrete representation of sound information, if the sample rate is high enough in comparison to the highest frequency that is being recorded, the .wav can be reconstructed as a continuous wave. The sample rate of a .wav file must be at least twice as high as the highest frequency in Hz that is of importance in the recording. Higher frequencies must be discarded prior to producing a .wav file. This minimum bound for the sample rate is known as the Nyquist rate. So long as the sample rate is greater than or equal to the Nyquist rate, a continuous wave can be extrapolated from a discrete .wav.

Standard human hearing extends to about 20kHz, meaning a good Nyquist rate for .wav files should be about 40kHz. The default sample rate of a .wav file is 42.1kHz, which is just above the Nyquist rate. The ratio between the highest frequency in an audio sample and the actual sample rate is called the Nyquist ratio.

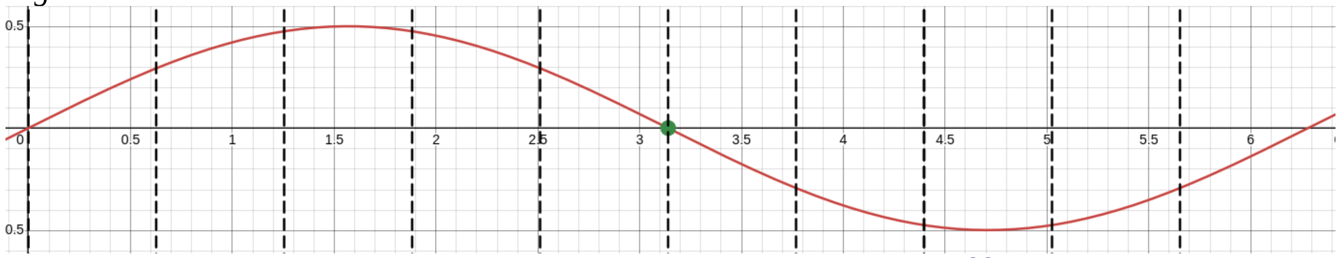
To extract key audio frequencies, a band-pass filter is used. To focus specifically on mains hum in the US, the frequencies extracted are 59-61Hz. A scipy implementation of a Butterworth band-pass filter is used, using sos to reduce numerical instability[3]. This is then fed through an sos IIR filter to produce a final output.

Unfortunately, with such a high Nyquist ratio, and such a low frequency band, the output is heavily artifacted. Through testing, a Nyquist ratio above 16 produces unacceptable levels of artifacting, providing us an upper bound on the Nyquist ratio. This is mitigated by first using scipy's decimate implementation, which filters out high frequency sounds, and then down samples by a specified factor. The decimate function itself has limitations. The factor by which to decimate must be a whole number integer, and it must be less than 13 to prevent distortion. This means if the sample's original Nyquist ratio is any factor greater than 13 by our upper bound for a target Nyquist ratio, the down sampling method must be performed in steps. Any method of integer factorization will produce some percentage of error. A custom factorization method based on logarithms is used in this paper, which for the specific use case of extracting a 59-61Hz band on 44.1kHz sample rate audio data falls within an acceptable margin of error for the target Nyquist ratio. While an upper bound of 16 has been established, a lower bound of 10 is detailed in the Comparison section. A full analysis of this method produces fascinating results, but I will not explain it in more depth in this paper.

Comparison

This mode takes two audio files. The first is the unaltered sample to be analyzed, and the second is the background signal that can be compared against and used to generate a timestamp of when the original file was recorded. This paper primarily focuses on analyzing the 59-61Hz component of samples, and the bg.wav file on the project github was generated by extracting the 59-61Hz component of a normal audio file since I did not have access to mains historical data.

Figure 1



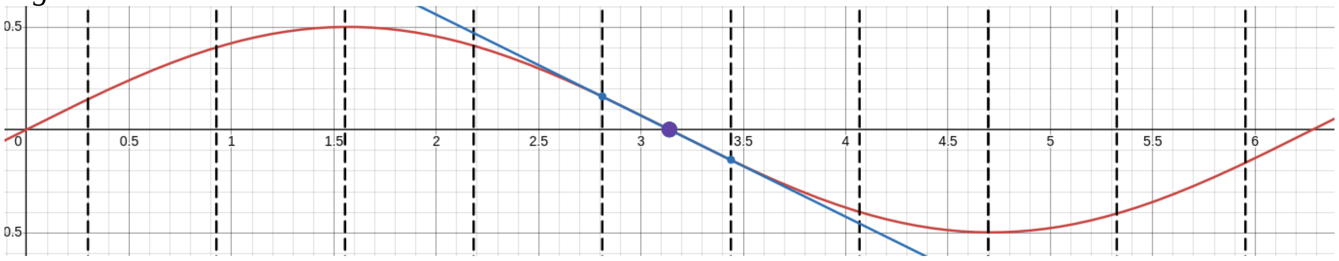
Figures 1, 2, and 3 were generated with Desmos[4]

A large challenge in this implementation was finding a good way of determining if two signals matched each other, even if the amplitude and phase were offset from each other. The method I devised, although flawed, was to represent both signals as a series of points when they each intersected with y=0. Then, compare the distances between these zeros on both samples, which is equivalent to comparing their wavelengths. If the differences in zeros are the same for a portion of both samples, then there is a match. When a sample falls directly on zero, zero detection is simple, and is shown in *Figure 1*. But sometimes no sample lands directly on zero, so some level of interpolation must be applied. I chose to implement a simple linear interpolation of the two nearest points to zero, although this has its own issues detailed in Future Work.

Assuming the Nyquist ratio is high enough, the area of a sine wave bounded by the two nearest sample points to zero can be assumed to be linear with extremely little error. Although lacking any real rigor, by visual examination I determined a Nyquist ratio of 10 to be a viable lower bound, preserving a reasonable expectation of local linearity with relatively little error. This bounds the desired Nyquist ratio for the down sampling method to be between 10 and 16.

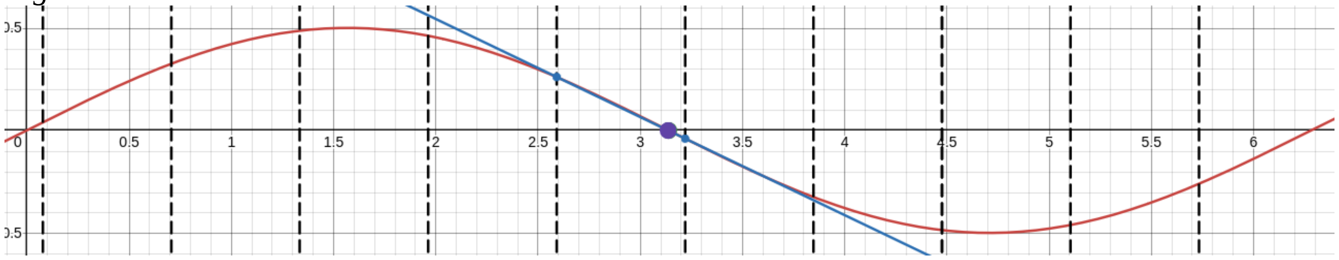
Figure 2 and *Figure 3* display the proposed linear interpolation method with a Nyquist ratio of 10. *Figure 2* displays the roughly best case for this method, with an error of less than 0.005 radians between the interpolated zero point, and the actual zero point of a sine wave. *Figure 2* shows roughly the worst case. In *Figure 2,* the error rate is less than 0.01 radians. These error rates are acceptable for this use case.

Figure 2



Error less than 0.005 rads

Figure 3



Error less than 0.01 rads

Now that wavelengths have been interpretted from both .wavs, comparing the wavelengths can be performed. Once again, this turns out to be rather complicated. I tried two seperate methods, which each have their own limitations, further discussed in Challenges, and I discuss ideas on new methods to try in Future Work. The first method is a simple detection of a subarray. This method treats the sample being investigated as a subarray of the background that has been recorded, and attempts to find a 1 to 1 match of the subarray within the background. The second method, smallest difference, again treats the sample as a subarray of background, but instead of searching for a 1 to 1 fit, it finds the fit with the least difference in wavelengths across the whole sample.

Results

The exraction method works as expected. When an audio sample is recorded on equipment that can accurately record 60Hz, a 60Hz component can be extracted. The bg_normalized.wav file is a file that has had its 60Hz component extracted and normalized to an audible level. Listening to this produces the expected 60Hz tone with little to no distortion or artifacts. Additionally, as shown in *Figure 4*, the resultant visualization of the .wav is extremely periodic, matching what is expected from a narrow band. This verifies not only that the band-pass filter used is applicable for this use case, but it also verifies the assumption that hidden data exists in audio files, and can be extracted for analysis.

Results for the methods of comparison implemented, however, are negative. In the vast majority of cases, the simple comparison returns false negatives, and the smallest difference method returns a false positive. Reasoning as to why the methods do not act as intended is detailed in Challenges.

Challenges

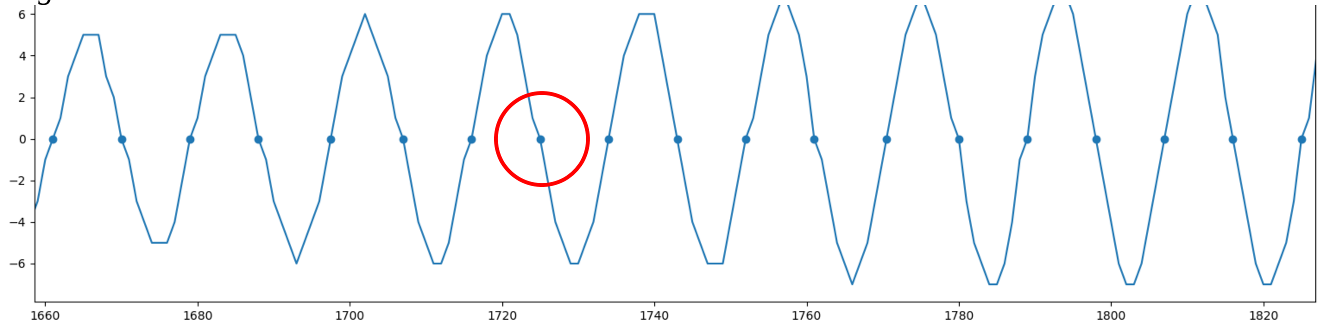
I faced many issues during the implementation stage of this project that I had not accounted for during the planning stage. The biggest challenge had to do with the precision of .wav representations.

Precision

Not only are .wav files discrete representations of a signal, they also only store amplitude as a 16 bit signed integer. Since I'm extracting such quiet frequencies from a sample, the generated output often only has an amplitude of 1 or 2 in the resulting .wav file. With higher precision representations, this would not be an issue. However, this makes many processing steps difficult to perform without complicated interpolation methods. For example, attempting to normalize the volume back to audible levels by simply multiplying the .wav by some factor doesn't work. If sample point 3 and 4 were originally both at amplitude 1 and sample point 5 was originally amplitude 2, an ideal amplifier would multiply sample 3 and 5 by some factor, and then appropriately interpolate sample 4 to be somewhere between them. Instead, the resultant normalized .wav produces extremely jagged jumps in amplitude between some sample points, introducing extreme distortion.

Additionally, this low level of precision makes my method of finding the zero points for amplitude quite inaccurate. In *Figure 4*, the wave section nearest the circled area is notably jagged near the zero point, making a simple linear interpolation method an inaccurate way of detecting zero points. Small differences in recordings would offset zero point detection significantly enough that comparison with this method is infeasible.

Figure 4



Zero detection is inaccurate due to 16 bit precision

Comparison

The second notable challenge I encountered was regarding the comparison methods used. While the simple comparison of wavelengths is extremely brittle to any differences in wavelengths between samples, the smallest difference method is too accepting of false matches. The smallest difference method also highlights a more fundamental issue with this approach. Comparing two samples based off of nothing but wavelengths is not an accurate method of comparison. Amplitude should also be accounted for in a similarly robust or fuzzy method.

Figure 5

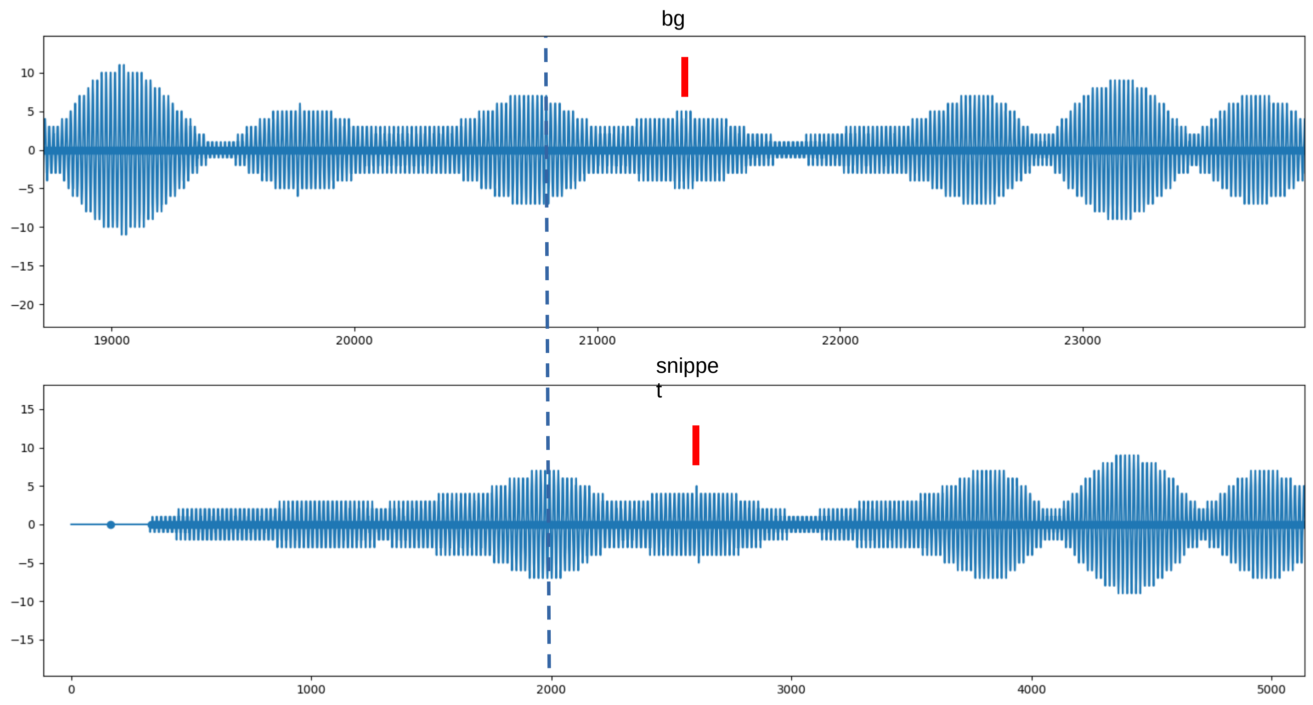


Figure 5 shows two charts displaying samples which should contain a match between them. Both bg and snippit originated from the same audio recording. Snippit was first sliced using ffmpeg to contain only a portion of the whole file, while bg was not sliced. Both bg and snippit had their 59-61Hz frequencies extracted. *Figure 5* shows this data, with bg being phase offset to visually line up with snippit.

It is expected that the beginning and ending of the snippet will be different due to the slicing process. This extends to around sample point 2000 in the snippet, and is signified with a blue dotted line. But even when trimming, there are a multitude of micro differences in the .wavs past the point of start/end trimming. The red bars indicates one such area. While the general shape of the wav is similar, there are 4 local peaks at 4 amplitude in bg, and only one local peak at 4 amplitude in snippet. This does not produce an audible difference, but significantly complicates design of any algorithm to determine matches between two samples.

Simple comparison of wavelengths are clearly not resilient enough against signal perturbations for comparing two different recordings that each contain the same background signal, so this method should not be used. A smallest difference of wavelengths method is closer, but returns too many false positives. When capturing such a narrow frequency band, of course there will be many close matches between the expected wavelength and the recorded wavelength. Instead, a fuzzy matcher that accounts for amplitude as well as wavelength should be implemented for this task. Ideas for such a method are proposed in Future Work.

Future Work

Factorization

Although not discussed extensively in this paper, a more in depth analysis of logarithmic based factorization should be analyzed. It is noted that for any given logarithmic base, integer or not, the maximum error rate is always exactly 1/3, or 33%. This error rate is far to high for almost any practical purpose, and a new nearest integer factorization method should be developed. This phenomonenon should also be investigated further.

Comparison

Another possible method for comparing two samples in an extremely resilient manner is to abandon the idea of representing the signal by its zero points, and instead try to fit a more complex line to a series of points. The current method is fitting a portion of the signal to a linear model using only two points per half wavelength. This does not hold enough context to perform much analysis. Instead, if a series of points within a half wavelength or a whole wavelength was used to fit a polynomial regression, then the coefficients used in that regression could be used as the basis for comparison. While it is not possible to fit a sine wave to a polynomial of finite degree, it is possible to fit portions of a sine wave. The smaller the portion being fit, the fewer degrees are needed in a polynomial to maintain a certain level of acceptable error. Here, a smallest difference method could be used against the regression coefficients. In other words, by comparing the polynomial coefficients produced by a regression fit across a whole or part of a sample, and then finding what portions of the two samples have the most similar coefficients, it could be determined which parts of the signals are the most similar. Finally, a threshold could be set for when to determine if two samples have a match or not.

Since the two samples are not guaranteed to have similar amplitudes, the sample being tested could have its amplitude normalized between 0 and 1, and each window of the background being compared against could be similarly normalized. This method would be extremely resilient towards differences in recordings and various other perturbations because it accounts for both wavelength similarities and amplitude scaling.

Conclusion

While the methods proposed and implemented in this paper fail to accurately compare signals to each other, and, as a result, fail to produce timestamps for audio evidence with missing or invalid metadata, they do display that some forensically valuable information is often hidden in plain audio formats. In particular, the mains hum is hidden in most recordings and can be extracted with relative ease. If it is not present, either a recording was made outside the presence of any electrical grid, the recording was made on high end equipment that effectively disregards mains hum, or the file was edited to remove it. These cases do not apply to the vast majority of recorded audio files.

I have no education in signal processing, so this was likely more complicated than it needed to be, but I did learn a lot doing this and I'm glad I got to work with audio data and audio processing hands on. Music and audio are one of my biggest interests, and learning more about how they work is always an enjoyable task.

References and Citations

[1]: Cooper, Alan J.; The Electric Network Frequency (ENF) as an Aid to Authenticating Forensic Digital Audio Recordings – an Automated Approach [PDF]; Metropolitan Police Service; Paper 3; 2008 Available: https://aes2.org/publications/elibrary-page/?id=14411

[2]: https://ffmpeg.org/

[3]: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html

[4]: https://www.desmos.com/calculator