Phoneme Identification

by Adam Burkhalter, Duong Chau, Micah Lee

Abstract

The project will take a look at a method to split up phonemes in recorded audio. The algorithm in question will be variable segmentation, and the authors will attempt to replicate it as much as possible. To approach this algorithm they built a tool to analyze WAV files as Fast Fourier Transform in order to determine the magnitude of specific phonemes. The project was able to successfully decompose a phoneme into its frequency model in respect to its magnitude which resembles the individual phoneme sequences.

Contents

Techniques

- Original Signal Plot
    - This project actively utilized discrete signal values and plotting those discrete signal values with respect to time. The calculations for the sample time duration utilized the audio sources sample rate to convert the sample duration to real time duration. The signal then was plotted with respect these values.
- Euler's formula
    - This project actively converted Cartesian coordinates to polar coordinates to allow for simpler data processing. Euler's formula was heavily used in the conversion process.
- Complex Exponentials
    - The project creates a data structure a complex number data structure in Java because it is not provided by the language. This data structure helps in the manner of finding and calculating the FFT of the signal.
- Discrete Fourier Transform
    - The discrete Fourier transform is used to determine a signal's frequency content. It reveals what a signal is composed of. Determining what a signal is composed of is extremely valuable is phoneme identification as there is a unique DFT signature for each phoneme.
- Fast Fourier Transform
    - The fast Fourier transform is the computationally efficient version of the discrete Fourier transform. It has a complexity of O(n * log(n)) compared to O(n^2) for the discrete Fourier transform.
- Hamming Windows
    - The fast Fourier transform inherently causes "power leakage". Power leakage causes the FFT to show frequencies exist, even though they don't. Windowing the signal segment helps mitigate this leakage.
- CD Quality Audio Recording
    - Audio was recorded in CD standard specifications: 44.1 KHz sampling rate and 16 bit quality. .wav files are created using this format and read using the same.
- Amplitude/Frequency Plot
    - The FFT returns an array of complex numbers. These complex numbers contain data that is useful signal decomposition, such as the magnitude of components signals.
- Audio stream manager singleton
    - A singleton manages all audio streams and provides single, unified interface to access them.
- Recorder
    - Opens static input line, managed by the system
    - Records at optimal length using the sample size
- Playback
    - Opens static output line, managed by the system
    - Plays back at optimal length using the sample size

Algorithm

Algorithm Overview

A signal enter the program is stored as an array of byte. The byte values will be converted into complex doubles. Then, the program apply FFT algorithm to the signal complex doubles in order to generate a new complex doubles list as the result of FFT. The results, then, will be converted back into array of double as is required to plotting.

Algorithm Components

Read in data file

Discrete signal data is read in from the .wav file. Processing the data ensures an O(n) space complexity by discarding data at each interval. This ensures O(1) data consumption. This data is then passed to the FFT processor for processing.

Hamming window

The data read in from the .wav file was passed directly to the Hamming Windowing algorithm. The algorithm is modeled by the equation x*[n] = x[n] * (.54 - .48 * cos(2 * PI * n / (N - 1))). Essentially, this algorithm minimizes the impact of power leakage at the signal segment boundaries.

Complex number class

A complex number class is a class that specify a data type that will be used to calculate the FFT. This class contains two main data: real value and imaginary value. The class provides several methods to do simple computational math such as: addition, subtraction, and multiplication. A complex number in the program can be initialized by a given pair of Cartesian values or a double values - the double value, d, will be used to compute sin(d) as real value, and cos(d) as imaginary value of the complex component.

Fast Fourier Transform

Referencing to the algorithm in the class textbook(page 112), an FFT function is developed based on the provided pseudo-code. This is a recursive method that takes in a list of complex numbers and returns the FFT of that list. The method split the even and odd indexes into 2 separate list and recursively calls the method on those 2 newly created lists. As for the returning value, it will generate a list of complex values that are summations of the even index values and the odd index values, with the odd index values multiplied by e^(-jk2PI/N).

getMagFFT

getMagFFT is a function that will be called directly when the data is being processed. This method takes in a double array of data return a FFT double array of the data. First, the double data is converted into a list of complex values data. Once the data is in a complex format, it is ready to be passed into the FFT function to generate another list of complex values data. The generated result then will be converted back into a double array. Now that the double array contains the magnitude values of all the complex numbers, it is ready to for plotting and displaying.

Plotter

Once the magnitude with respect to the FFT was generated, the FFT data was plotted. FFT data was plotted as lines, and since the FFT size is larger than the plotting window, the plot appears to be perfectly smoothed, if only a little be aliased.

Background

Technology continues to become more mobile and more hands-free as it advances, and so do the controls to manipulate it. Since hands-free technology fundamentally requires a different set of controls compared to traditional mouse and keyboard computers, another computer control system must be implemented. Since humans communicate primarily with speech, controlling a computer with speech should be intuitive as well. This assumption was made research began on how to make a computer understand human speech.

Phoneme identification is a very useful tool in human to computer communication, as it is a core component of voice recognition technology. Without it, it would be near impossible for any computer to interpret human speech. Humans understand speech is this way and are able determine what word is being said by the identification of phonemes. For example, the phoneme detector can determine that bale is different from ball because the ‘a’ sounds different in each word. The project then combines the phonemes nearly instantaneously and forms a mental understanding of the word.

Many devices are implementing some version of hands-free commands to control a device, and consumers enjoy it as it a less physically taxing control system. For example Google’s “Google Now” allows for voice commands to nearly completely native instructions. Voice recognition is a huge part of this, as it is the main layer between the user and the device’s capabilities (Smith).

Design

Audio Tool

The audio tool is designed to help aid in the phoneme splitting process by adding an easy way to record audio in the correct format, as well as saving the bytes for easy analyzing.

Recorder Object

When a user clicks the Record button, the application will create a threaded Recorder object. This object will be responsible for recording and saving the audio created by the user. In order for the recording to start, the Recorder's "start" method must be called, after which it will run everything by itself. The first step for the Recorder is to set up the utilities it needs to successfully record audio. In order to do this the Recorder will attempt to obtain a line from the system by creating an Info object, and passing it through an AudioSourceLine. Once this has completed successfully the next step is to start recording the audio. Essentially this will just be like file transfers, as bytes will be continually read until the recording has stopped. To receive the audio at the optimal rate, it must calculate the amount to read using the Audio's frame size: (AudioInputStream bufferSize / 8) * framesize. Then it simply continues reading bytes by calling AudioInputStream's read method. Once recording has finished the Recorder will clear out and close the line by calling the "clearLine" and "clearBuffer" methods. Finally the Recorder will write all the recorder audio to the main AudioInputStream managed by the application.

Playback

When a user clicks the Play button a new Playback object will be created. Playback is also a thread object and will be responsible for playing the audio back to the user. The design is the same as Recorder, and will only start as its start method is called and will complete the rest by itself, however Playback will also have a method in which a File can be passed and played. The first step is to setup the utilities necessary to play audio back to the user. In order to do this the object will need to create an Info and pass it through a DataSourceLine. Next the playback will call its "playback" method which will again calculate the optimal rate to play the audio back. From this point the Playback object will write bytes to the DataSourceLine until all the audio has been played.

Plot Signal

Signal will be a much simpler algorithm than Plot FFT, as it will simply plot the raw data from the WAV file, or bytes chosen. Essentially this button will loop over all the bytes that make up the signal, and plot each byte on the graph.

Plot FFT

After the getMagFFT function is called during the data-processing process, plotFFT button is ready to be clicked and will perform the following tasks when clicked. When plot FFT is clicked by the user, the program converts the FFT magnitudes values into window points. These points are registered to the drawing window. Once the window is created and displayed on the user's screen, all the points that were registered are plotted and connected to create a consecutive set of lines, which form the segmented signal. Thereafter, a graph of FFT is generated. This is a linear process and prior data processing is needed in order to complete the formation of the FFT plot. If the user clicks on this button without having the getMagFFT function called previously, the program would not create a graph, and crashing would be a possibility.

User Guide

Audio Tool

Description
The audio tool is a simple tool in order to record audio in a format that can be used in the phoneme splitter. At a high level view the recorder will take in audio through an input line, and store it as bytes, which represent the sound the computer just heard. The tool will also play the audio back, and will allow the user to save out the audio in the target format and as bytes (text document).

Recorder
When a user hits the record button, the application will create a new Recorder object. The audio will be recorded in standard CD format: 44.1 kHz, 16 bit quality. The recorder object is a threaded object that will open up the InputLine, and read the data coming through the user's microphone. This data will be stored in the static AudioInputStream.

Playback
The playback aspect of this tool will take the audio from the AudioInputStream, and load it into a SourceDataLine, which will open and play as the user clicks play. The user will also be able to stop the playback, by clicking the same button.
The user may also opt to play their own file. The tool will only be able to play a few select formats. If the user selects Choose File, a selection window will appear and they will be able to select their own file to playback.

Split Phonemes (Not yet implemented)
This button will split the phonemes of the selected audio automatically, and save each phoneme as a separate audio file. This will not be 100% accurate as phoneme splitting is a very difficult process and there are a lot of chances for error to occur.

Plot Signal
This button will plot a segment of original signal processed by the FFT application. Each time a segment is plotted, the signal is move 1/10th of a second. This helps verify that the raw audio is captured well and represented fairly. It is particularly useful to view the raw phonemes and what the unique phoneme signal looks like. By comparing the phoneme signals, it is quite easy to see the differences between the two.

Plot FFT
This button will plot a segment FFT of the current audio in question. Each time the FFT is plotted, it will work out to 1/10th of a second for each plot. This will help analyze the audio by displaying how many signals make up the compound signal which will ultimately help in determining which phoneme the application is trying to split. The user progress the signal shown by clicking the Show Next button.

Alternative approach

An alternative approach to reconstruct the word is by comparing the signal with individual letters, instead of phoneme. This would not be a very efficient way to approach due to the fact that the same alphabetical letter, in American English, is pronounced in different ways. As an example, the words “ball” and “bale” both contain the same letter “a” but are pronounced significantly different. Therefore, if the program is made to compare the input signal with how the letters are pronounced in the alphabet, incorrect results or errors would be generated.

Limitations

The assumption is that this project is only working with the English language, which contains 42 phonemes. The project is also assuming that audio streams will be primarily composed of the phonemes and not noise. This will reduce, if not completely remove the need to filter out noise thus improving the phoneme detection rate. There is also an assumption of using a single accent is going to be used, eliminating pronunciation differences further increasing the phoneme detection rate.

Findings and Results

Data Generation

Using the signal processing tools described previously in this document, signals segments and their FFT were able to be plotted. Screen shots were taken of the plotting windows and compared against similar images.

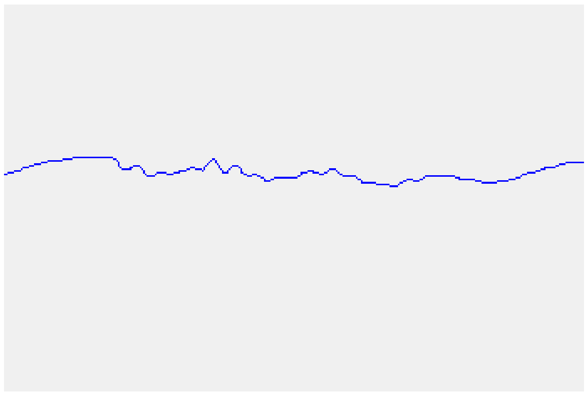
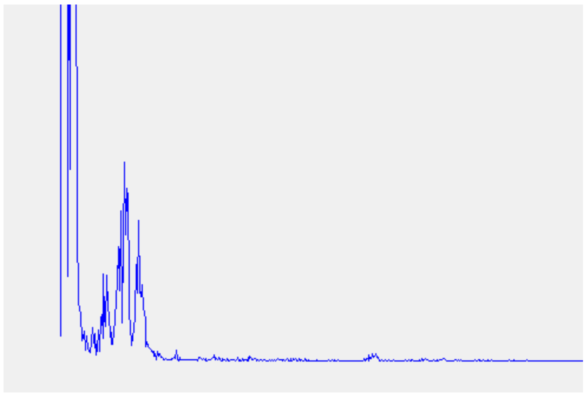
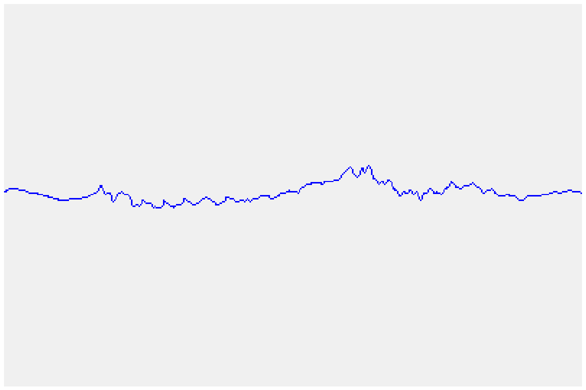
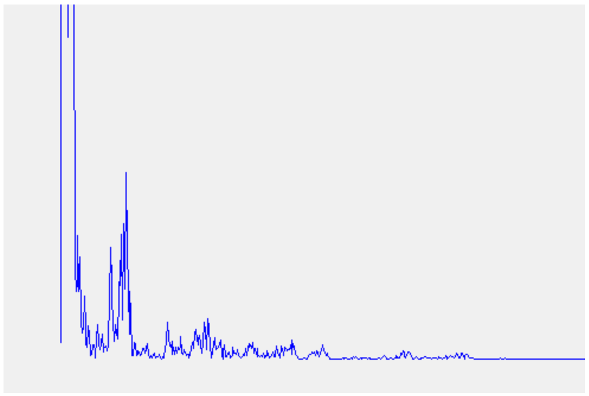
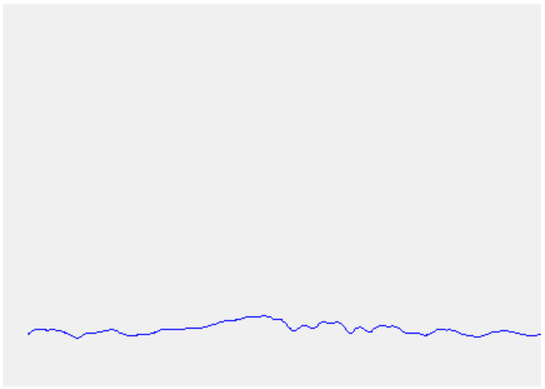
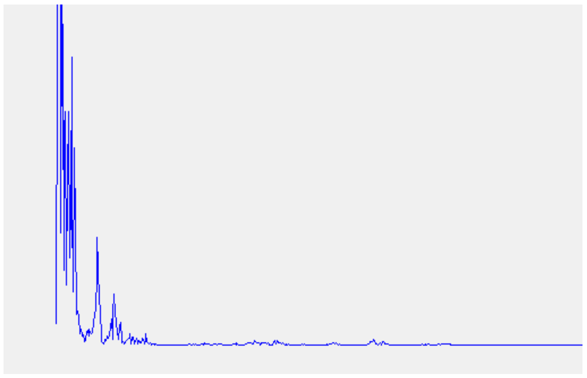
The project members decided to visually compare the spoken phoneme /A/ with the other to determine if the FFT algorithm produces correct results. The recording was done in a quiet room with very little noise to minimize its effect. Two of the project members recorded speaking the phoneme /A/ at different times to ensure they didn't attempt to mimic each other. The plotted signals are similar, but not obviously so. The FFTs revealed the content necessary to generate the /A/ phoneme and showed that the frequency content was indeed similar. The members noted that the primary similarity is the presence of two distinct frequency peaks. These peaks could note the unique frequency signature of the phoneme. Table I shows these results.

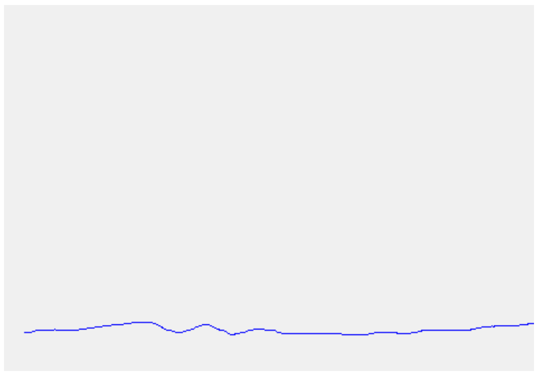
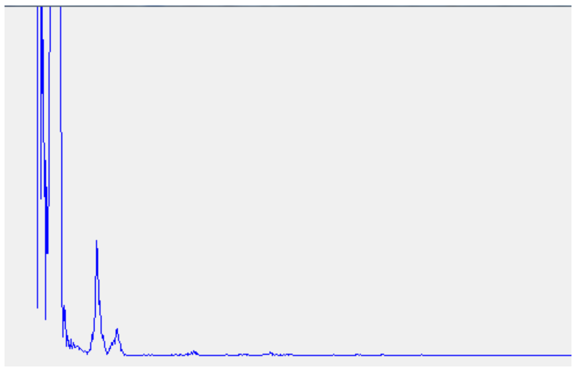
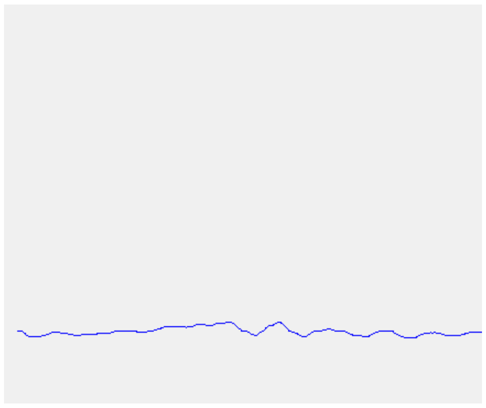
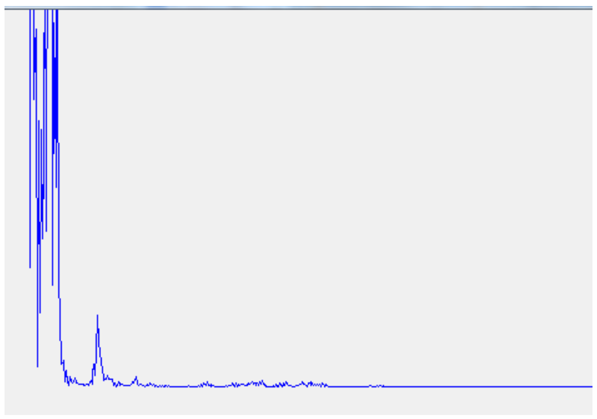
Data Analysis

When comparing the separate phoneme sounds, it is also very apparent that each produces their own unique sequence. At first glance all of the "a" phoneme sounds look similar, however upon closer inspection there are very distinct peaks at which each phoneme can be distinguished. Given this information the authors are able to theorize the isolation of each phoneme in a word or sentence represented as a signal.

Due to time constraints, the project members were unable to perform expansive data analysis. The project members theorize that averaging multiple person's pronounced phonemes could generate a truly accurate base line phoneme to compare against. Once this baseline is established it can be used to determine if phonemes are of a specific type. The theorized algorithm is as follows: difference between the baseline phoneme and the phoneme being processed can be taken is fed into a standard deviation function. All standard deviations are averaged together in an unspecified way, possibly using a geometric mean. If the mean standard deviation is below a certain value, then the phoneme type has been determined.

Table I: Signal Segments and Their FFTs

Signal Segment	FFT of Segment
 Figure 1: /A/ phoneme signal graph generated through Adam's voice	 Figure 2: /A/ phoneme FFT graph generated through Adam's voice
 Figure 3: /A/ phoneme signal graph generated through Micah's voice	 Figure 4: /A/ phoneme FFT graph generated through Micah's voice
 Figure 5: /a/ signal graph generated through Adam's voice	 Figure 6: /a/ phoneme FFT graph generated through Adam's voice

 Figure 7: /ar/ signal graph generated through Adam's voice	 Figure 8: /ar/ phoneme FFT graph generated through Adam's voice
 Figure 10: /aw/ signal graph generated through Adam's voice	 Figure 9: /aw/ phoneme FFT graph generated through Adam's voice

Problems

The project was not able to complete as proposed due to the time constraint. On top of that, the ambition of this project is extremely high. The concepts and the materials that were presented during the researching phase of this project were very difficult to comprehend for all authors. The time that was spent to fully understand the project was longer than expected, which lead to a limitation on time for implementation and testing.

During the recording it was also very difficult to achieve identical and similar recording environments as many variables had to be taken into account: volume, position of the microphone, accents, and tone. These variables, even though may seem to have an insignificant affect on the signal, convolute the signal a substantial amount. Another aspect is human error, as background noise also affects the signal in a very momentous way. Even just breathing during a signal can modify it in a way which makes the phoneme unrecognizable.

Future Work

For future work and what potentially can be done if the project was not limited by time, the splitting phoneme button could be implemented to complete the job that is promised in the project proposal. The next logical step in regard to continuing this project is to implement the use of transient. With the transient, the border of the phoneme will be easily determined and

recognized. Then a process of understanding what kind of phoneme it is and what it means will be taken to make guesses of the actual word loaded into the program. After understanding what the actual word is, the program will reconstruct the word.
Beyond the promised project proposal, the project can take this tool a step further to implement a form of speech recognition, which can be useful in many modern day technologies. Being able to take this project to understand sentences at a time would be the next huge step, however one step at a time is a must.
Furthermore, this project can use the method of standardization to compare the difference between one voice versus another. This will further expand the boundary finding to another level.

References

Smith, Matt. "A Look at the Future of Hands-free Computing." *A Look at the Future of Hands-free Computing*. Digital Trends, 21 Mar. 2013. Web. 12 Dec. 2014. <http://www.digitaltrends.com/computing/a-look-at-the-future-of-hands-free-computing/>.

Jaynath Kumar Talisila and Pradip Sircar. “Parameterizing Speech Phonemes by Exponential Sinusoidal Model.” *Parameterizing Speech Phonemes by Exponential Sinusoidal Model.* PDF file.

"English Phonemes, Spellings, and Meaningful Representations." *English Phonemes, Spellings, and Meaningful Representations*. N.p., n.d. Web. 12 Dec. 2014.

Stiber, Michael, Bilin Zhang Stiber, and Eric C. Larson. *Signal Computing: Digital Signals in the Software Domain*. Bothell: U of Washington Bothell, 2014. Print.

Link to open source:

https://github.com/micahrlee/AudioProject