Simple Voice Commands

Introduction:

Speech is a set of specific wave modulations that form words. Voice assistants these days use complex deep neural nets to predict speech based off lots of vocal data. Before deep neural machine learning, hidden markov models were used as an algorithm to detect speech and before that simpler methods of statistics were used. More information on vocal processing can be found here: <https://en.wikipedia.org/wiki/Speech_processing>. This task heavily involves signal processing. This document aims to give an educational example for a simple signal (voice) processing that can be made and utilized in MATLAB (this example code can be applied to many other languages).

**Capturing a Signal**

First things first, an input is needed for us to process it. An incoming signal in this case is just any way we can measure an amplitude change in time. Depending on the application, a simple breadboard microphone with the appropriate microcontroller can achieve this, but for simplicity, in this tutorial MATLAB was utilized in conjunction with a headset microphone connected to the computer for direct or recorded input.

Pre-Recorded: (see figure 1 below)

* This method is simpler on the coding end, (both methods are included in speechTesting.m). First an audio file will be needed. This can be recorded from any simple program that can output wav or mp3 files.
* After loading the file using MATLAB’s audioread() function, the data and ample rate variables are returned in to data (the audio data) and Fs (sample rate) respectively.
* In this example we are taking the 2 channel (stereo) data and only extracting 1 side of it (mono) to make the task of processing simpler.

Active input: (see figure 2 below)

* This method is much more practical, therefor after this section it will be used for all of the following in this tutorial.
* Using MATLAB’s audiorecorder() library we are able to take a live audio capture in the program from the system computers main connected microphone. The audiorecorder() outputs an audio captured object, (more info on the details of this data structure can be found here: <https://www.mathworks.com/help/matlab/ref/audiorecorder.html>). In this utilization we give it three inputs: Fs, a predetermined sample rate (we use 44100), nBits, the amount of bits per sample (8 here), and NumChannels, or the selection of whether we want to record in mono or stereo (it is set to 1 here, mono, for simplicity).
* Taking this object, we call recordblocking() for 5 seconds and we print out some messages to the user in the command window so they know when recording has started and stopped. This is more of an active utilization of a microphone because we are telling it when to listen to us instead of it always listening and then parsing out our speech looking for a keyword.
* Finally, calling getaudiodata() will return us a new matrix of the audio data or an amplitude over time matrix. The return of this function is the same as our audio read from recorded input so the rest of the code can work for either case.

\*\*NOTE there is code included to plot the signal over time. Uncomment it for some visualization.

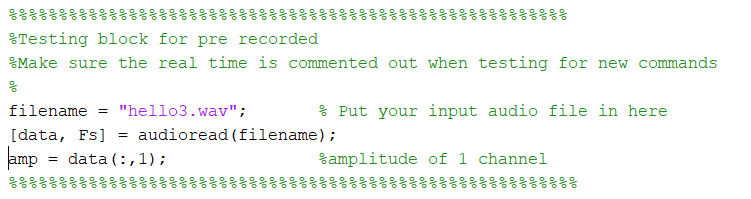


Figure 1: Pre-Recorded files

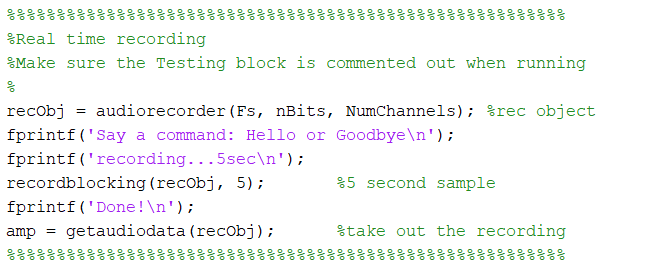


Figure 2: Active input

**The Fast Fourier Transform**

If you are a Purdue engineering student well into their work chances are this is review; each distinct signal has certain frequency characteristics that compose it. For analysis it is very useful to be able to separate this information in a simple way, hence, the Fourier transform is needed. For those who have never seen it, it is simply a method of math to convert a complicated wave into the most basic sinusoidal parts at different frequencies. The normal Fourier transform is very expensive on a computers cpu computation wise, therefor, the fast Fourier transform was created. We will be using the built in fft() library in MATLAB and almost all modern coding languages will have a fft library as well.

* Begin by simply calling the fft() function with the amplitude data as the input from the recorded audio. Time data is not needed because we are moving from the time domain to the frequency domain.
* The return might give us values that are harder to calculate so we will take the absolute value with abs() of the spectrum divided by the length of the signal.
* For simplicity, we will look at a single side of the spectrum for analysis, therefor we will cut down the dual sided spectrum to a single sided spectrum with the next line.
* The result is a list of amplitudes at certain frequencies but the frequency at which amplitude occurs is still needed. This will be denoted by the new variable called band.
* Finally band will be cut down to represent 0-20000kHz or the range of our human hearing.

\*\*NOTE there is code included to plot the amplitudes over their frequencies. Uncomment for some visualization.

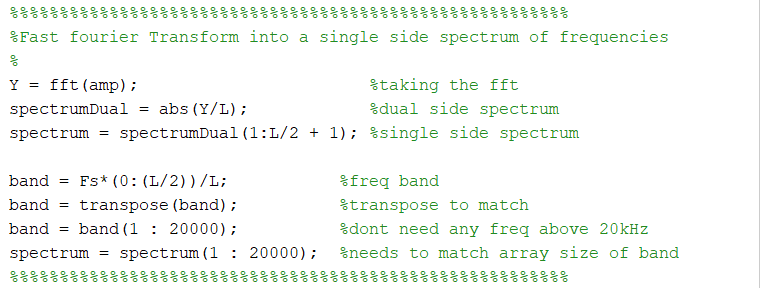


Figure 3: FFT MATLAB adaptation

**Analysis (simple statistical methods)**

To maintain simplicity, the analysis being used is not much more than some basic statistics run on many test cases. The example excel booklet shows some of the tests that were used to build up to determination. Depending on the environment or microphone these values might be different enough to make a significant difference so the code will have to be recalibrated. The following methods are used to collect key features or analyze them.

\*\*\*NOTE\*\*\* As testing went on, the decision to normalize the amplitude data was made. Normalizing the data gives a more absolute scale over multiple tests to help account for large variances. The normalize() function was called in MATLABs built in library before any of the following was carried out.

Finding Peaks (findingPeaks.m):

This function takes in the data of spectrum amplitudes over a frequency band and a number specified as n. The function is O(n) as it goes through every point collected in the spectrum. It collects the amplitude and the frequency at that amplitude for the greatest values of amplitude n times. So, if n is 40, the function will return the 40 highest amplitudes and the frequencies at which they occur; they are the most prominent in the spectrum. A higher n means more detail, but it also means that it is more complex to later analyze due to capturing more features. The variables stored in MATLAB, seen in fig 4, were taken to excel; multiple n values were tested for finding a useful method; the test data for the words hello and goodbye can be seen in the example excel data book.

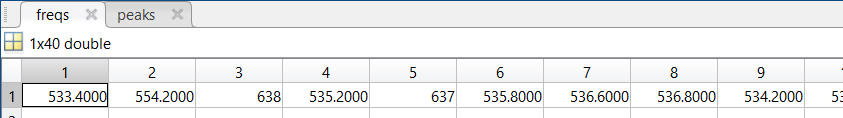


Figure 4: example n=40 data from MATLAB

Peak Density (peakDensity.m):

This function takes in the amplitude and frequency band data much like finding peaks. It is useful to uncomment the plot for the frequency and spectrum band when testing to see what amplitudes pass any certain threshold value. In short by setting a threshold value in thresh this function returns how many peaks are above that specified value. These densities can then be analyzed after numerous tests to find correlations between different command words at different threshold values. In the case of this example program the words are hello and goodbye.

Statistics

The included example excel data booklet shows a multitude of different tests that were run in experimentation to determine a difference between the two key words. From making this project it was found that the normalized peak density test was the most useful in disparaging a difference. Edit or move onto extrapolating these tests for your own purposes.

**Determination**

After performing several tests with the two key words hello and goodbye, the clearest difference could be seen in the normalized peak density data. There is a clear separation that with a threshold value of 1.2, (on the normalized scale), set to our microphone it appears that the word hello tends to have below 1800 peaks above the thresh value while goodbye tends to be upwards of 1800. Running a few more tests to account for false positives in detecting the right key word a slight correlation was seen with one of the stat variables included in the function analyze.m. The frequency at the smallest amplitude in the finding peaks vector was useful in detecting false positives for hello. This variable and the average frequency in the freqs finding peaks vector were useful for detecting goodbye false positives; in either case more tests were run to find correlations between the statistic variables.

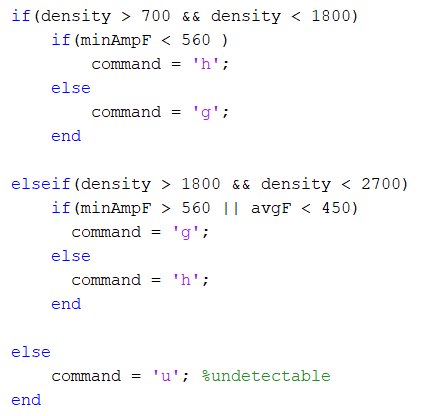


Figure 5: Determination case logic