Tap To Tempo, BPM Detector and Signal Modifier

Austin Votypka, Keon Garrett

Department of Electrical and Computer Engineering
University of Rochester

ABSTRACT

Understanding the importance of BPM (beats per minute) selection in music production is a crucial component of creating a musical piece. Our work focuses on designing a Matlab program that alters, or simply detects, the BPM of a signal. Since BPM change is equivalent to changing the speed of a signal, which innately effects pitch, the program can also alter pitch independently.The program improves upon similar plug-ins found in Logic Pro X and allows users to input an audio signal to adjust the BPM of that signal, with various methods of input. We utilized various audio signal processing tools such as peak detection, frequency filtering, full/half wave rectification, fourier transforming, etc to develop a function/program that can detect and alter the BPM of and audio signal. This paper outlines how we have implemented each of these tools and shows how we developed our user-interface for efficient functionality .

Index Terms— BPM, pitch, alteration, detection

1. INTRODUCTION

The BPM measure of a song or sample is the foundation for what it is built upon. Changing the BPM can drastically alter how your final product sounds and, because of this, many artists want to have the ability to change the BPM of an audio signal. The BPM is simply a way to measure the tempo of a piece of audio, usually musical. The higher the BPM value, the 'faster' the tempo of the audio.

It is easy to see why this is such a big factor in how a piece of audio sounds. By looking specifically at a musical piece, if your BPM is 120 and you want to add in a new instrument then you must be playing with a tempo equal to 120 BPM or one of its "harmonics", such as 15, 30, 60, or 240. Trying to include a new instrument at a different BPM will not be pleasing to the ear and is very noticable. This is why presently, where sampling is so popular in music, being able to adjust an audio signals BPM is a very useful tool to have.

Many different DAWs, digital audio workstations, and plug-ins have BPM detectors and/or signal modifiers. One such DAW being Logic Pro X and its plug-ins; BPM Counter and Time and Pitch Machine. However, one feature that Logic does not have is being able to determine what you want to change your BPM to by what is called tap to tempo. This feature, which is present in other DAWs/plug-ins, basically allows the user to press out, usually on a keyboard, their desired BPM in real time.

In this paper a Matlab program that detects a signals BPM and can alter it, along with the pitch, is described. This paper goes over how the user interface runs, how the pitch and speed are modified, and how the BPM detector, itself, works.

2. PROGRAMS OVERALL PROCESS

The Matlab program runs with a user interface that allows the user to choice how they run the program. The user initially can decide whether they want to "Alter" an audio signals BPM, or just "Find" a BPM that they have either in their head or from an audio signal without altering.

2.1. Alter

2.1.1. BPM

If the user enters "Alter" the program will immediately ask them to input the name of the audio file they would like to change. This file must be in the Matlab path that the program is running in, or else they will be asked to reinput their file name. Once the file is loaded in the user will either enter the BPM of the audio file or run the BPM detector on the audio file to determine the BPM. After this, the program will prompt the user to decide how they would like to change the BPM, if at all. The choice the user has are; "Clap", "Tap", "Number", "Song", or "No Change".

If the user enters "Clap" the program will run a function where the user will clap out their desired BPM. This program sets up a device recorder in Matlab and uses the computers built in microphone to record since sound quality does not play a factor in BPM calculation. The device will record for 10 seconds and after the recording is finished this audio signal is sent through the BPM detector.

Once the BPM is output, the user will have the choice to either keep this tempo or try again if they wish.

If the user enters "Tap" a similar process will occur, except the BPM is tapped out on the space bar. The function associated with "Tap" will bring up a figure prompting the user to begin tapping. The function runs for 15 seconds and counts up the number of space bar taps in this time. After the 15 seconds is over the function multiplies the total number of taps by four to obtain the BPM. It does this because 15 seconds is one fourth of a minute. Again, after this process has happened the user will be asked if they want to try again or keep their current BPM. If they want to keep the BPM then they enter the value displayed to them before proceeding.

The "Number" option is available if the user already knows the number that they want to change the BPM to. This option is the only one that is available in Logic Pro's Time and Pitch Machine. The user is told to enter the BPM number and this becomes the new BPM.

If "Song" is entered by the user then the program will access a function that contains a bank of a bunch of different songs. These songs all have different BPMs, varying from 75 to 175. The different BPMs are displayed and the user chooses what BPM they want to listen to. Once they enter a number, the song that correlates to that BPM will play for about 15-20 seconds to give the user a good idea of what that tempo is like. They will then make a decision on this tempo, depending on the BPM, or they can listen to a different BPM before proceeding.

Lastly, the user has the option to enter "No Change" to keep the BPM the same. This choice is available because the program can also alter the signals pitch. So, if a user wanted to just alter the pitch, and not the BPM, they would enter "No Change".

2.1.2 Pitch

After a decision has been made regarding the BPM, this value is stored. The user will then choose to change the pitch as well, or not. Once an option has been selected the program runs one of two functions adapted from the third homework on the audio signal. These functions alter the speed and pitch of a signal independently, one function for mono signals and one for stereo. Next, the signal gets normalized and is bounced out of Matlab for the user to use.

2.2. Find

Back when the program is first ran, if "Find" is entered by the user they will first be promoted to make a choice concerning how they want to find the BPM. The options they have are alike those of "Alter", but with one difference. The options are; "Tap", "Clap", "Song", and "Analyze".

The "Tap", "Clap", and "Song" options run the same functions as they do in "Alter". However, the only difference is that once the user has settled on a BPM the program just outputs this "found" BPM and that is all.

If a user is simply just trying to find the tempo of an audio signal, without altering it, then they would select "Analyze". This choice will prompt them to enter the name of an audio file for analyzation. Once they enter the name, the program runs the BPM detector on the file and then outputs this value back to the user.

3. BPM DETECTOR

The foundation for Logic Pro X's BPM Counter works by reading a audio signals impulses and determining the BPM from them [3]. The BPM detector in this program also works in a way that is analogous to Logic's method. It's process was adapted from a similar program developed at the MIT Media Lab [1]. It contains 4 steps that breaks an audio signal down into its envelope, to better view impulses or peaks in the audio amplitude, and then utilizes convolution in order to determine the BPM.

3.1. Filterbank

The first step in the BPM detection process is to send the audio signal into a filterbank. The filterbank will split the audio signal up into different frequency bands for BPM analyzation. Analyzing an audio signal as, in this case, six different frequency bands instead of just one gives more accurate results. This is because if the whole signal was analyzed then the "amplitude impulses" of the drums would tend to overpower other instruments. Being able to analyze different frequency bands allows the program to focus on other instruments, as well as the drums. This is also similar to having six different signals, with the same tempo, to analyze and compare with, which will innately increase the validity of the final output.

Once an audio signal is in the filterbank, the fourier transform of the audio signal is taken to take it into the frequency domain. Next the frequency bands are constructed. Using a choice of frequency bands that produced the best results, 0-500 Hz, 500-1000 Hz, 1000-5000 Hz, 5000-10000 Hz, 10000 - 20000 Hz, and 20000 Hz - sampling frequency, the program calculates which samples of the audio signal these different bands represent. Once these indexes are determined the corresponding samples are loaded into a six column matrix, where each column represents a different frequency band.

The filterbank process is now complete and this matrix is ready to go into the next step.

Figure 1 shows the spectrogram of one of the frequency bands. In the spectrogram, the yellow colors represent high energy content of that frequency and blue represents little to no energy content. Knowing this, it is easy to determine what frequency band is being shown, even if it was not told to you.

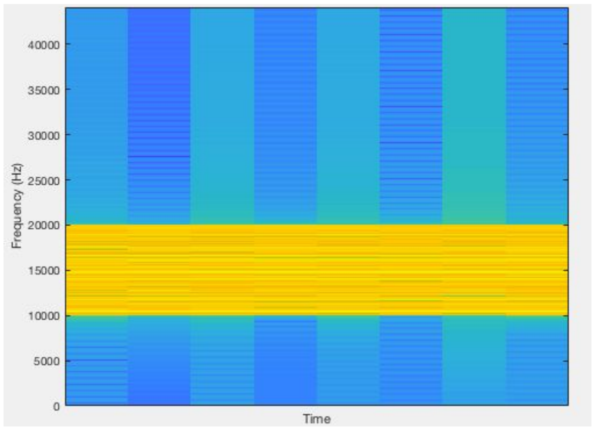

Fig. 1. Spectrogram of 10000-20000 Hz frequency band

3.2. Signal Smoothing

Once the signal has been split into frequency bands, it can now be "smoothed" into its envelope. An audio envelope is represented in Figure 2. Getting the signal in this form helps the program to analyze the amplitude peaks/impulses. In Figure 2, the point where the "attack" and "decay" sections meet would be a peak. These peaks are not nearly as discernable when the signal is not in its envelope which is why this step is important for accurate BPM detection. In order to achieve this we must full wave rectify the signal and then low-pass filter it with a hanning window [7].

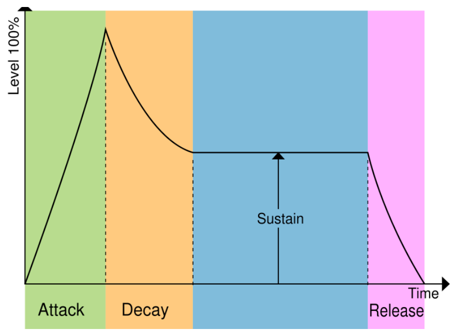

Fig. 2. Graph representation of a sound envelope [6]

3.2.1. Full Wave Rectification

The first step in reducing each frequency band down to its envelope is full wave rectification. Full wave rectification is similar to taking the absolute value, except the concept encompases both the absoute value or the negative absolute value [2]. In this part of the program, the positive full wave rectification is used, which is equivalent to just taking the absolute values of each frequency bands.

This gives us the basic shape of our envelope and must be done before low pass filtering. The reason for this is because if a signal, with positive and negative values, is sent through just a low pass filter then the output will be very close to, if not all, zeros [7]. This happens since, with positive and negative values, the mean of all the values will be close to zero [7]. Full wave rectifying the signal before low pass filtering guarantees that the mean will not be close to zero and likewise for the corresponding output. The shape of the envelope is now sent to the hanning window to be smoothed.

3.2.1. Hanning Window Application

When people think of signal smoothing the first thing that would come to mind is windowing. This is a process that can be utilized as a low pass filter [5] which is important for only viewing a sounds envelope, since the shape of an envelope is independent of the frequency content. The program utilizes a hanning window to perform smoothing because it has a more drastic side lobe roll off than a hamming window. As a result, when the window is applied to the signal the peak will be more pronounced and the other components of the envelope less so than if a hamming window was used.

The signal also only needs half, the positive side, of the hanning window applied because after full wave rectification the signal is completely made up of positive values. Therefore using the negative side of the hanning window is unnecessary and would only go to increase computation time. The signal has now been low pass filtered into a smoothed envelope [7] and the inverse fourier transform of the frequency bands is taken before proceeding.

3.3. Amplitude Peak Accentuation

Now that the audio signal's frequency bands are in an envelope form, the next step is to highlight amplitude changes. Specifically the changes that culminate in an amplitude peak, the largest ones. These peaks will correspond to a “beat” in the audio signal. In order to accomplish this the frequency bands will be differentiated and half wave rectified in the time domain.

3.3.1. Differentiation

The first step is differentiation by finding the amplitude difference between each successive sample in each frequency band. Doing this will allow the program to store variables that correspond to the changes in amplitude between samples. By comparing these the program will be able to more easily determine which differences correspond to that of a “beat”.

It is obvious that when this method is performed there will be some negative differences, corresponding to decreases in amplitude. The point where the amplitude increases from one side of a sample, but decreases from the other side would be considered a local peak. By combining differentiation with half wave rectification, the program can determine which of these peaks corresponds to the “beats” of the signal.

3.3.2. Half Wave Rectification

The process of half wave rectification is similar to full wave rectification, but with one major difference. In full wave rectification the values all remain intact, whether they were positive or negative before. However, in half wave only the positive values remain and all the values that were negative, before rectification, become zero [4].

This process is performed on the difference values, so the only values that are retained are ones where the amplitude was increasing. Doing this makes it easier for the program to notice when a peak happens, because as soon as a zero value is read it knows that the previous value was where a peak occurred. Now that our peaks/impulses in each frequency band have been highlighted the final step may commence.

3.4. Combfilter Impulse Response Convolution

With the audio signal now in envelope form, where the peak points have been highlighted, the only step left to take is that of combfilter impulse response convolution. During this process, vectors that represent a combfilter’s impulse response are created. Each impulse response corresponds to a different BPM, from 60-240. What this means is that the impulses happen at points, over time, that would correlate to a certain tempo. Each of these impulse responses are convolved with every frequency band of the audio signal and then the resulting energy for each band is summed. Whichever convolution, between a certain BPM and the signal, results in the highest amount of energy, after summation, is what the program outputs as the signal’s tempo.

The accuracy of this program is determined by the bandlimits, set for the frequency bands. By changing these limits the program is able to predict the BPM for either better or worse, this effect is shown in Figure 3. Setting the limits as; 0-500 Hz, 500-1000 Hz, 1000-5000 Hz, 5000-10000 Hz, 10000-20000 Hz, and 20000-sampling frequency Hz gave very accurate outputs, these limits resulted in the correct output plot shown in Figure 3. These limits decided upon to take into account the frequency ranges of different types of instruments.

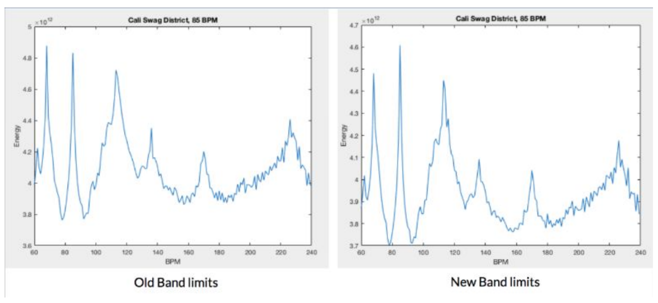

Fig. 3. Effect Changing Bandlimits Has On Output, Left Plot Gave Incorrect BPM Output While Right Gave Correct

4. CONCLUSION

By updating, and combining, functions from the third homework with the BPM detector and user interface described in this paper, a superior program to Logic’s BPM Counter and Time and Pitch Machine was developed. Not only does this program include both of the processes of each Logic program, it improves upon them. This Tap To Tempo, BPM Detector and Signal Modifier allows a user to alter an audio signals BPM or pitch independently, just like in Logic. However, the user has more alteration options to decide from than just entering a number. Being able to tap, clap, or compare to other songs to alter the tempo of an audio signal are all features that Logics software does not provide. With improvements in the BPM detector algorithm, for computation time, this program could surely rival other such plug-ins in the music industry today.

5. REFERENCES

[1] Cheng, K., Nazer, B., Uppuluri, J., & Verret, R. (2001). Beat This. https://www.clear.rice.edu/elec301/Projects01/beat_sync/beatalgo.html

[2] Full-wave Rectifier. (n.d.). https://www.eecs.tufts.edu/~dsculley/tutorial/diodes/diodes3.html

[3] Logic Pro X: BPM Counter. (n.d.). https://support.apple.com/kb/PH27653?locale=en_US&viewlocale=en_US

[4] Marivani, S. (2012). DIODE CHARACTERSITIC AND THE HALF-WAVE RECTIFIER. https://web.sonoma.edu/users/m/marivani/es231/units/experiment_04.shtml

[5] O'Haver, T. (2018, June). Smoothing. https://terpconnect.umd.edu/~toh/spectrum/Smoothing.html

[6] Park, J. Adabox_1280px-ADSR_v2.svg.png. https://learn.adafruit.com/assets/67594

[7] Rose, W. (2014, July 23). Electromyogram analysis. https://www1.udel.edu/biology/rosewc/kaap686/notes/EMG analysis.pdf