MIDIoke

transforming human singing to digital music

Georgios Kyziridis, giorgos.zapata@gmail.com , s2077981
Geerten Verweij, geertex@gmail.com, s1420062

January 19, 2018

Abstract

MIDIoke is made to transform human singing to a MIDI signal. It uses fast Fourier transformation and autocorrelation to get the fundamental frequency from the input. It outputs a MIDI file that can be used in other software. MIDIoke's limitations are; the constant note length, the lack of directly streaming MIDI and the lack of consonant filtering.

1 The Idea

The goal was to create a way to transform singing to a MIDI signal and to be able to have an instrumental representation of what was sung. We had several steps in mind for the implementation. Firstly the pitch needs to be extracted from an input signal. Then this has to be transformed to a MIDI signal. Then that signal needs to be transferred to some software or a device that can play MIDI.

2 Implementation

2.1 Pitch Detection

Pitch detection procedure started with a simple implementation of fast Fourier transformation for the raw input signal. In the early stages of developing MIDIoke we followed the naive approach of simple fft according to the standard formula below.

$$f(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi} dx$$

That standard-initial approach of fast fourier transformation was producing too many different frequencies that did not match the users sung input melody which was a significant obstacle in usage of the midi-output representation. Fast Fourier

transformation detected many other harmonics apart from the fundamental frequency, of the voice-singing melody, which were distorting the midi output. In order to avoid that we tried to implement an upgraded fft by convolving the original raw signal and the reversed one using autocorrelation. The result of this approach was significantly better than the old one with respect on the clarity of segmented pitches we extracted for the midi output. The autocorrelation function is a measure of similarity between a signal and itself delayed by τ as the formula below.

$$\varphi_x(\tau) = \int_{-\infty}^{\infty} x(\tau)x^*(t-\tau)dt$$

We finally used the that approach in order to extract the midi output which included fewer harmonics that distort the original midi signal.

2.2 Generating MIDI

Once a solid pitch estimation in Hertz is acquired it must be converted to a MIDI note. To get the MIDI pitch value we use this standard formula:

$$midiValue = 69 + 12 * \log_2 \frac{freqValue}{440}$$

The input frequency is also limited to be between 20 and 4000 Hz. This is done to remove any noisy surrounding frequencies. The window of 20 to 4000 Hz matches the expected input from the user. The energy of the signal is also calculated by taking the root mean square of a chunk of input. This energy value is then used to determine if the input is loud enough to be considered wanted input. A threshold value of 10 (there is not really a unit of measure for this) was found to do the job but this is of course dependent on the hardware used. MIDI notes will only be generated if the energy of a chunk of input surpasses the threshold. The MIDI notes are outputted to a file which the user can then load into any software that takes MIDI files, for playback.

3 The Result

3.1 The Experiment

Our experiment with MIDIoke was quite simple. We sang a basic melody, put the output file into a MIDI sequencer and then checked if the result matched our intended melody. As it can be observed in Figure 1, we can see the actual pitch distribution according to what was already sung. We can also see the power(energy) of our sung melody according to the pitch. Moreover, we can define that the energy is independent from the pitch. The green line in Figure 1 is the default-constant threshold, MIDI notes will only be generated if the energy surpasses this threshold. Figure 2 shows the outputted file loaded into a sequencer. The time axis is reversed compared to Figure 1. You can see that the pattern of the pitch in both images match.

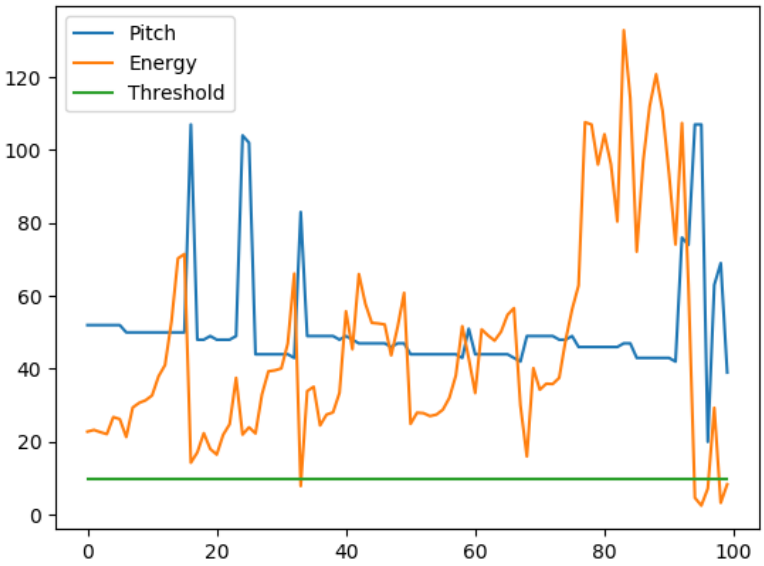

Figure 1: This is what the user sees when using MIDIoke, the values on the axis have arbitrary values

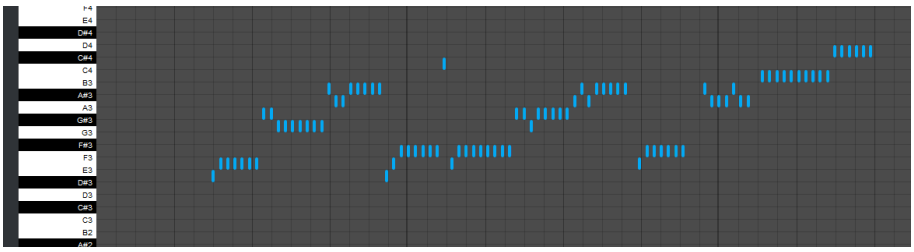

Figure 2: MIDI pattern resulting from the recording from Figure1

3.2 Performance

The pitch of the produced MIDI file did match the melody that was sung when we demonstrated MIDIoke. Occasionally some extra notes are added that do not fall within the melody which are probably created by consonants or background noises.

3.3 Limitations

The current version of MIDIoke has a set length for each note, when a singer produces a longer note this will simply become a sequence of the same note with short durations. It would be much more desirable to have the length of the output notes match the length of the input notes. This will be discussed in the Would Haves section below.

4 Would Haves

Our final implementation of MIDIoke does not contain everything that we initially wanted to put into our project. Here are some parts that are missing which we would like to add to MIDIoke in the future.

4.1 Matching Note Length

To get the note lengths in the output to match the note length of the input MIDIoke would have to keep track of the last sung note and only output if a note ends. Then it could output that note with the correct length. Implementing this would not be very complex. However the reason why it is not implemented is twofold; Firstly is of course the time constraint on a project like this. Secondly, and more importantly, it is because in the final form we would want MIDIoke to be this would not be needed. In the final form we want streaming MIDI, which we will now discuss.

4.2 Streaming MIDI

By streaming MIDI to other software that does the audio synthesis based on the MIDI signal, MIDIoke would work just like a MIDI-keyboard as input. And if this works the matching note length problem would be fixed as well. The notes played by a MIDI-keyboard are only played as long as the key for that note is pressed. If this could be implemented for MIDIoke the length of the notes produced would match the length of the notes that are sung. Another big advantage streaming MIDI would give is the possibility to have live synthesis of the MIDI output of MIDIoke. This would make it truly like karaoke. The user would then hear a live version of what he/she is singing generated by a synthesizer of choice.

4.3 Portable Hardware Version

If the code for MIDIoke could be executed on an Arduino or Raspberry-Pi which would than output the MIDI signal through USB the MIDIoke project would truly be in it's ultimate form. Then the user could take the MIDIoke-device to any place and plug it into a digital audio workstation to sing some synthesizers. The code for MIDIoke might be too heavy for an Arduino but perhaps with the right hardware on the Arduino and some optimizations it might be possible. This would be preferable over running it on a Raspberry-Pi since the Arduino tends to have less latency. But this would be a whole new project in and of itself.

5 Conclusion

The result of our work gives the opportunity to users to convert their singing melody into a midi.output which can be used as an input to every digital audio workstation and then be processed according to the user's demands. The MIDI signal of the output is sufficient with respect to the fundamental frequency. The output signal will include some frequencies that are useless, which might come from the users consonants. Making the note lengths match the input would be a very nice improvement for future versions of MIDIoke.

6 References

- Correlation and Autocorrelation using FFT
- Fast Fourier Transformation
- Signal Processing Python Documentation
- Simple Frequency Estimation Methods in Python