AUTOMATIC BEAT GENERATION FOR MUSIC PRODUCTION AND PERFORMANCE

Terahn Harrison
terahnharrison@gmail.com

Chris Ellard
chrislellard@gmail.com

Sam Johnson
SamDLJohnson@gmail.com

ABSTRACT

This project investigates a means of producing output generated both from pre-recorded audio files and from real-time input with a microphone or other input device. Specifically, we aim to detect the pulse, tempo, or rhythm of audio input, which is then used to reciprocate with new sounds in a way that matches or relates to the original. This involves building software with both audio-tracking and audio-creating capabilities.

To do this, part of the code, which is written in Python, includes creating an interface where the user need only select the timbre or sound characteristic of instruments, and letting the script handle generating and mapping their temporal locations for the output. Musical rhythm, or "beat", is of interest, and so although there may or may not be a clearly distinguishable rhythm in the input sounds, drum-type samples are sufficient in creating the output.

1. INTRODUCTION

Much of the mainstream music production these days follows a very simple formula of taking a sample and laying drum patterns on top it. It is not a stretch to assume that this process could be mimicked by an algorithm using some MIR techniques. Beat tracking and tempo induction are commonly used in DJ software to aid the DJ with mixing songs into a seamless stream of music by matching tempos. These same techniques could be used on samples to find their tempo, and then the extracted info could be used to generate a new audio file with drums added on top of the original sample, creating a beat.

In this paper, we propose a drag-and-drop music production tool that takes as input a drumless audio sample and a set of drum samples, and outputs an automatically generated audio file, which we will refer to as a "beat", using the original samples provided. To develop this tool we will use Python 3 with a GUI using Pygame.

We will assume that all input samples will have a steady tempo. The genres we will target in this project will be variations of hip-hop and electronic music (ex. trap, lo-fi, house) as these genres tend to have a steady tempo with programmed drums. This will allow us to mimic these genres more accurately compared to genres such as rock and jazz that primarily use real drummers. The input samples that are used in this project have been downloaded from a free sample website called Looperman.

Further work could include developing the ability to use the tool in a live setting. For example, instead of using a preexisting audio sample as input, a musician could play an instrument directly into the tool, and then it would start to automatically accompany the musician once it has sufficient info on the tempo the musician is playing at.

2. RELATED WORK

There has been extensive work in the areas of tempo induction and beat tracking since the 1990s. Most of these early solutions, however, were evaluated using popular music with drums [1] [2]. Masataka Goto, who worked on numerous beat tracking systems in the 90's, published a paper for beat tracking for music with or without drum sounds in 2001 [3]. Since then, there have been many papers published exploring the area of beat tracking and tempo induction [4] [5] [6].

These techniques have been used extensively in different applications. For example, most DJ software will analyse your music library, extracting the tempo from each song in order to aid the DJ in choosing songs to play. These same techniques are used in music streaming services to suggest songs with a similar tempo. The idea of using beat tracking for automatic musical accompaniment has been explored, but no actual tool has been developed [7].

3. PROJECT DESCRIPTION

3.1 Team Member Roles

Writing of the report will be divided equally among members with the Project Manager having the final decisions on the quality and content of the report.

Terahn Harrison: Project Manager, Developer, and Lead Writer
Chris Ellard: Developer, Tester, Writer, and Proofreader
Samuel Johnson: Developer, Tester, and Writer

3.2 Timeline

Date Range	Objective
Oct. 17	Submit project design specification
Oct. 21 - Oct. 27	Begin planning out python code
Oct. 28 - Nov. 4	Develop code including user interface
Nov. 20 - Nov. 22	Test and evaluate working system
Nov. 23 - Nov. 30	Add findings to report
Dec. 1 - Dec. 7	Finish report and prepare for presentation

4. APPROACH

4.1 Front-end

The front-end of the application is an interface with various rectangular buttons that can be clicked on and toggled. The display is divided into three sections, each of which is labelled in bright, capital letters. The section on the left side has one button for the input sample at the top, and three buttons for the drum samples at the bottom. The drum pattern section covers the upper right side of the display, and shows three rows of sixteen buttons–representing the three drum types, hi-hat, snare, and kick–that can be toggled on or off. The output section below contains a button that generates an output track for the audio file selected for the input sample based on the selected drum pattern.

When the application first opens up, all of the drum pattern buttons are initially off, and no 'play' button is shown until the user clicks on the 'generate' button. Clicking on the drum pattern buttons will play that drum sample sound once.

The application uses the python libraries Pygame and Tkinter. Pygame is used for the visual component and button interaction, and Tkinter's filedialog is used as an additional interface to help with searching for and opening files or audio samples. All of the buttons are constructed from Pygame's Sprite class, with size and location parameters. This includes width, height, as well as x and y coordinates relative to the upper left corner of the display surface window. A 'sprite' essentially acts as another image that is shown overtop of the display surface that can move or function independently from the display, and can interact with user input. The image of each button sprite is updated via pygame's `blit()` function.

The main function first initiates an Interface object, which acts as the main window for the application. The interface class initializes the main display surface, sets the internal clock, and sets default or 'placeholder' path names for drum samples, in case the user tries to generate a drum track without choosing any. After the interface is created, its `run()` function is then called, starting the main loop for the application. If at any time the user closes the application window, the interface's `self.running` variable is set to false and the loop breaks. The `run()` loop has four core steps:

1. `self.CLOCK.tick(FPS)`
2. `self.input()`
3. `self.draw()`
4. `pygame.display.flip()`

The first step in the loop controls the rate at which the display is updated, which is currently set to 60 frames-per-second.

The input function first reads the mouse position (x,y) relative to the origin (upper-left corner at (0,0)). With this information, all of pygame's events are checked via a for-loop. If the `mouse click` event type is `True`, then it

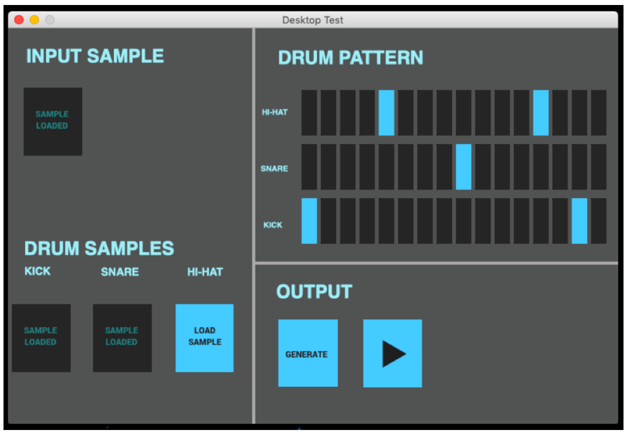

Figure 1. Screenshot of the GUI

checks whether the mouse is currently over a button sprite. If the button is one of the drum-sequence buttons, its image is updated as either `on` or `off`. Otherwise, if the button is one of the input sample buttons on the left side, the `OpenFile()` function is called which opens up a file-opener window (using tkinter).

While the input function includes updating or 'blitting' the appearance of the buttons, the `draw()` function then blits these button images to the display surface.

The last step, `display.flip()`, reveals the final result of the screen display image to the user. For illustration purposes, this step is shown here as part of the `run()` loop, however, in our application, we have placed this as the last line inside the `run()` method.

4.2 Back-end

The back-end engine was designed as a single Python function that takes as input a set of audio samples and the drum sequencer values from the front-end, and then returns a single output sample. The set of input samples consists of a drumless audio sample and a set of drum samples. The drumless audio sample should be preferably greater than 5 seconds in length to allow for more accurate audio analysis. This drumless audio sample is the sample in which the drums will be placed on top of. The set of drum samples should consist of single hits of each drum type to be added onto the drumless audio sample. This set of drum samples can be infinitely large, however currently the front-end is configured to work with 3 drum samples: a kick, snare, and hi-hat.

Once all of the input samples are loaded, the engine's next step is to run a beat tracking algorithm on the drumless input sample. This will return an estimation of the tempo, as well as a list of the estimated beat indices. The engine currently uses Librosa's beat tracking function, which implements the algorithm outlined in a 2008 paper by Daniel P. W. Ellis [8] [9]. The beat tracking algorithm used in the engine can be replaced with any other beat tracking algorithm, as long as the algorithm takes as input an audio sample and returns a tempo estimation and a list of estimated beat indices.

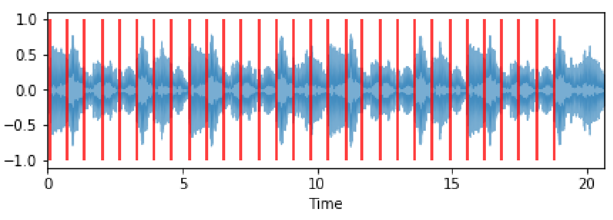

Figure 2. Estimated beat indices for a certain recording

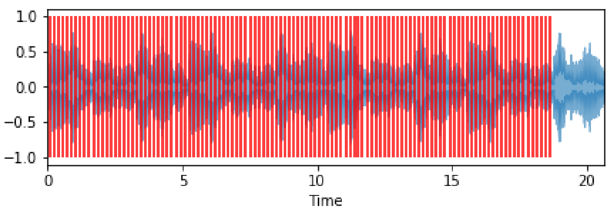

Figure 3. Drum indices subdivided into quarter notes

The next step consists of applying heuristics to the set of beat indices in order to improve their accuracy. This step is not necessary for successful execution of the engine, however it can help improve the end results. Currently we have implemented a heuristic to help normalize the tempo of the input sample. This compares the tempo estimation to an upper and lower BPM threshold. If the tempo estimation is faster than the upper BPM threshold, then the list of beat indices will be halved by removing every other value. If the tempo estimation is slower than the lower BPM threshold, then the list of beat indices will be doubled by adding the midpoint between each pair of adjacent values to the list. This helps normalize the list of beat indices of any songs with exceptionally fast or slow tempo estimations, such that all tempos fall in the range of 70-140 BPM.

Once any available heuristics have been applied to the beat indices, the list of beat indices will be subdivided into quarter notes. Assuming each value in the list of beat indices corresponds to the placement of a beat in the input sample, quarter notes can be estimated by calculating 3 evenly spaced numbers between each pair of adjacent indices. This gives 4 notes for each beat: the beat index from the original list and 3 evenly spaced indices leading to the next beat. This subdivision step returns a list of the estimated placements of each quarter note throughout the input sample.

The next step is to calculate the placement of each individual drum sample using the list of values taken as input from the drum sequencer. This input array is a list of boolean values of length 16*n where n is the number of different drum samples in the sequencer and each 16 values corresponds to the 16 notes a certain drum type can be placed at. A value of True corresponds to a note being turned on in the drum sequencer. Furthermore, each index of the pattern can be treated as a quarter note, and therefore we can map this pattern to the list of quarter note placements generated in the previous step by looping the pattern until its length is equal to the length of the estimated quarter note indices. This mapping can then be used to generate a new list containing the placement of each individual drum sample by looping over both lists, taking the quarter note placement value if the corresponding drum pattern value is True. This step returns a list for each drum sample containing the estimated note placements of each drum hit for that drum sample throughout the entire drumless input sample.

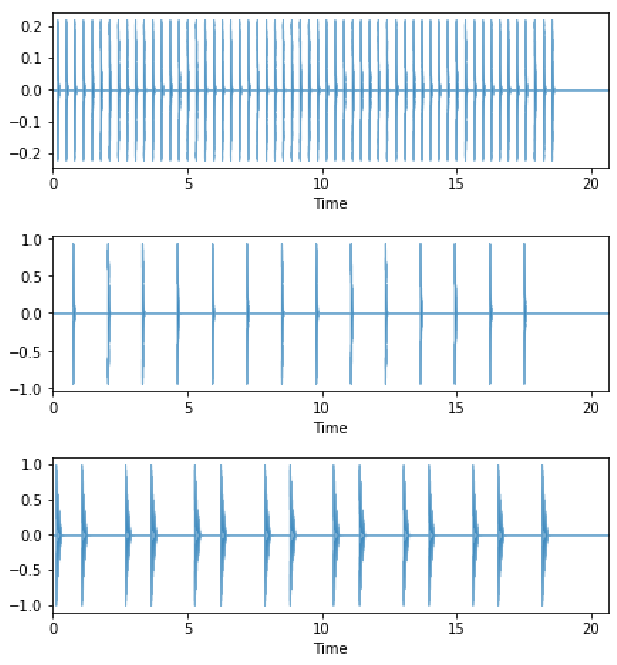

Figure 4. Individual drum tracks generated for each drum sample

Now that the placements for each drum sample have been calculated, an actual drum track can be generated for each drum sample. This is done by initializing an empty list with a length equal to that of the drumless input sample, and then adding in the drum samples at each index present in the corresponding drum placement list.

The final composite output track t_{output} can be created by summing the original input sample t_{input} with each drum track generated in the previous step t_i where $1 < i < n$ and n is the number of drum samples. The volume of each track can be modified by multiplying the track by a number v, where numbers between 0 and 1 will lower the volume and numbers greater than 1 will raise the volume.

$$t_{output} = t_{input} * v_{input} + \sum_{i=1}^{n} t_i * v_i$$

5. EXPERIMENTS AND EVALUATION

Evaluating our algorithm posed quite a challenge as it didn't produce numerical results or have a reputable work to be compared to. Therefore, in an effort to evaluate the algorithm, we tested it using a set of 10 input samples. Each output sample's correctness was then evaluated by ear. If the drums seem to fall in the correct logical times then the output sample is deemed to be ‘Good’. If most of the drums fall into the correct logical places, but there are minor errors such as some out-of-place drums or the drum pattern is at double or half tempo, the output sample will be deemed as ‘Needs Improvement’. If the drum placements are mostly all incorrect, then the sample will be deemed as ‘Bad’.

Sample Name	**Tempo Estimation**	**Evaluation**
87_mura_masa_1	86.13 BPM	Good
89_mura_masa_2	89.1 BPM	Good
90_lofi_piano	45.33 BPM	Good
90_mura_masa_4	89.10 BPM	Good
93_mura_masa_3	95.70 BPM	Good
120_guitar	117.45 BPM	Good
120_house	80.75 BPM	Bad
130_nick_mira_1	161.5 BPM	Bad
140_lofi_rhodes	136.0 BPM	Good
140_nick_mira_2	69.84 BPM	Good

To truly evaluate our algorithm we would likely be required to conduct a user study. Unfortunately, due to time constraints we weren’t able to conduct one at this stage. However, a user study would be conducted as follows:

1. Gather a sample of people that have different musical tastes and preferences
2. Instruct them to use our sampling application to create a snippet while we observe them
3. Once they get an output, have them rate it on a 10 point scale in categories of sound quality, listenability, and how it compares to their expected output.

With this information, we would get a better sense of how our algorithm performs while also getting feedback on the usability and intuitiveness of our user interface.

An additional evaluation that would aid in the assessment of our algorithm would be to use a source separation tool such as Spleeter to split a sample track into its different components. A new input sample would then be created by combining the components that don’t feature drums, thereby creating a drumless version of the original sample. The new drumless sample would then be put through our algorithm and assessed on how it compares to the original sample that included the drums. This method of evaluation would provide a new perspective that could be used to examine how our tool performs in relation to the original sample.

6. CONCLUSION

In this paper, we have proposed a tool that utilizes music information retrieval algorithms to facilitate the process of music production and performance. We also wanted this tool to have an easy to use interface to enable use by users with no knowledge of the underlying algorithms. While beat tracking is used extensively in the field for music analysis, using it for music generation is a rather unexplored area. Since our tool is used to generate new unique audio samples, accurately evaluating the performance is difficult. Nonetheless, evaluation by ear has proven that our tool produces good results for the majority of tested audio samples. The main factor in the performance of the tool is the accuracy of the beat tracking algorithm used, and therefore our tool should improve with the introduction and implementation of more accurate beat tracking algorithms in the future.

We believe this area of using music information retrieval algorithms in music production and performance could be very useful to musicians, and we hope our project can show a glimpse of what is possible.

Application code can be found online at `https://github.com/terahn/GaBI`

7. REFERENCES

[1] M. Goto and Y. Muraoka, “A beat tracking system for acoustic signals of music,” in *Proceedings of the second ACM international conference on Multimedia*, pp. 365–372, ACM, 1994.

[2] M. Goto and Y. Muraoka, “A real-time beat tracking system for audio signals.,” in *ICMC*, 1995.

[3] M. Goto, “An audio-based real-time beat tracking system for music with or without drum-sounds,” *Journal of New Music Research*, vol. 30, no. 2, pp. 159–171, 2001.

[4] M. E. Davies and M. D. Plumbley, “Context-dependent beat tracking of musical audio,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 3, pp. 1009–1020, 2007.

[5] J. Laroche, “Efficient tempo and beat tracking in audio recordings,” *Journal of the Audio Engineering Society*, vol. 51, no. 4, pp. 226–233, 2003.

[6] F. Krebs, S. Böck, and G. Widmer, “Rhythmic pattern modeling for beat and downbeat tracking in musical audio.,” in *ISMIR*, pp. 227–232, 2013.

[7] M. E. Davies, P. M. Brossier, and M. D. Plumbley, “Beat tracking towards automatic musical accompaniment,” in *Audio Engineering Society Convention 118*, Audio Engineering Society, 2005.

[8] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, “librosa: Audio and music signal analysis in python,” in *Proceedings of the 14th python in science conference*, vol. 8, 2015.

[9] D. P. Ellis, “Beat tracking by dynamic programming,” *Journal of New Music Research*, vol. 36, no. 1, pp. 51–60, 2007.

[10] M. Goto and Y. Muraoka, “Real-time beat tracking for drumless audio signals: Chord change detection for musical decisions,” *Speech Communication*, vol. 27, no. 3-4, pp. 311–335, 1999.

[11] S. Dixon, "Automatic extraction of tempo and beat from expressive performances," *Journal of New Music Research*, vol. 30, no. 1, pp. 39–58, 2001.

[12] M. E. Davies and M. D. Plumbley, "Causal tempo tracking of audio.," in *ISMIR*, 2004.

[13] M. A. Alonso, G. Richard, and B. David, "Tempo and beat estimation of musical signals.," in *ISMIR*, 2004.

[14] R. B. Dannenberg and C. Raphael, "Music score alignment and computer accompaniment," 1985.

[15] R. B. Dannenberg, "Real-time scheduling and computer accompaniment," 2002.

[16] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on speech and audio processing*, vol. 10, no. 5, pp. 293–302, 2002.

[17] C. Raphael, "A bayesian network for real-time musical accompaniment," in *Advances in Neural Information Processing Systems*, pp. 1433–1439, 2002.

[18] D. C. Correa, J. H. Saito, and L. da F Costa, "Musical genres: beating to the rhythms of different drums," *New Journal of Physics*, vol. 12, no. 5, p. 053030, 2010.

[19] K. Seyerlehner, G. Widmer, and D. Schnitzer, "From rhythm patterns to perceived tempo.," in *ISMIR*, pp. 519–524, 2007.

[20] A. Cont, J. Echeveste, J.-L. Giavitto, and F. Jacquemard, "Correct automatic accompaniment despite machine listening or human errors in antescofo," 2012.

[21] M. F. McKinney, D. Moelants, M. E. Davies, and A. Klapuri, "Evaluation of audio beat tracking and music tempo extraction algorithms," *Journal of New Music Research*, vol. 36, no. 1, pp. 1–16, 2007.

[22] N. J. Bryan and G. Wang, "Musical influence network analysis and rank of sample-based music.," in *ISMIR*, pp. 329–334, 2011.