BeatVOX Project Final Paper

John Kasun, Walter Kopacz, Connor Dowd, and Joshua Chan
School of Music
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Emails: johnkasun@live.com, walterkopacz10@gmail.com, connor.dowd377@gmail.com, cchan92@gatech.edu

I. Introduction to Context

A popular tool for the modern producer is the virtual drummer. This tool is used to create a computer generated drumbeat that sounds as if it were created by a real drummer. This is advantageous especially to the ”bedroom producer” who may not have access to a recordable drum kit or may not have the ability to play drums. It is also faster than programming a MIDI drumbeat and often sounds more realistic. However, while virtual drummers are very good at giving us a drumbeat very rapidly, they are not very good at interpreting what we as the user want the drumbeat to sound like. Different virtual drummer applications use various different kinds of vague settings for the user to adjust to try to convey to the virtual drummer algorithm what they want the drumbeat to sound like. For example, Logic Pro X's virtual drummer makes use of several different ”drummers” that play different styles of music with different characteristics. It also makes use of a 2 dimensional X-Y plane with the two dimensions of loudness and complexity allowing the user to pick a point on that plane to describe those two parameters of their drumbeat. While

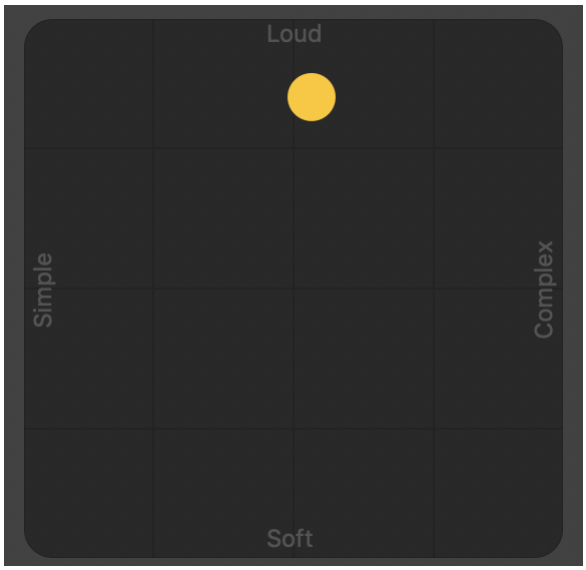

Fig. 1. This is the XY axis controller in Apple's Logic Pro X's virtual drummer

these choices can give you something that fits within the context of the producer's idea, they do not give you much control beyond that. If you have a specific rhythmic idea in your head, it is impossible to convey that to the virtual drummer. However, if someone were to simply make vocal percussion noises or ”beatbox” the drumbeat to a human drummer, a reasonably skilled human drummer could recreate the beatboxing on their drum kit.

This brings us to our solution to this problem. What if we could beatbox a drumbeat to a virtual drummer and it would understand it? This is exactly what we are trying to create. The human voice is in many ways the universal instrument and this concept could elevate its capabilities even further. While sitting down at a drum kit and playing a drumbeat is not intuitive for most, creating vocal percussion is so natural that many children can even do it. For this project we are going to attempt to create an algorithm and functional software solution that will take in the input of a person creating vocal percussion, interpret the input, map the input to drums found in a traditional drum kit, and then output a MIDI sequence that corresponds to the interpretation of the vocal percussion.

II. Applications

This idea and algorithm could be applied in various different ways. The main application of this software would be in the scenario described earlier in the introduction, as a tool for songwriters and producers who do not have access to live recorded drums but would like to quickly add a realistic sounding drum beat to their project. Currently these producers would have to spend time programming their own drum patterns using MIDI which is often challenging and does not always yield the best results for several reasons. Creating beats in MIDI can be challenging especially if you are trying to create a complex drum pattern or are inexperienced with this style of production. Creating a drum beat using solely MIDI can also lead to very robotic and overly quantized sounding drumbeats which is not suitable for many styles of music such as rock, folk, and jazz. Virtual drummers as discussed before are suitable for getting something that matches the context of the piece but are very hard to fine tune to fit a specific piece or a specific rhythmic idea that the artist might have in their mind. This tool could prove useful even for artists who are adept at programming MIDI drums as an extremely fast way for creating a customized drum beat, leaving them time to focus on other ideas.

Within the application of songwriting and music production there are a few different routes that this kind of technology could take. As stated before it could be useful in the creative process, as the performative qualities of beatboxing would better convey the artist's original intent, and preserve uniqueness in the performance. However, it also is useful as a tool for eliminating the process of manually drawing MIDI notes, making it more efficient while preventing the rigidity of quantized notes. While implementing this algorithm as a plugin would conform best with traditional music production convention, it could also be implemented in the form of a standalone application. This is the avenue we took. BeatVOX is opened as an external application and communicates to other applications, such as a MIDI drum sampler or DAW, via a virtual MIDI port.

III. Related Work

There are numerous products on the market that share similarities to this idea. The first of which, and perhaps closest contender, is the Dubler Studio Kit. This product, which recently graduated from Kickstarter, also deals with the conversion of vocal input to drums – it is marketed as a live "voice-to-MIDI controller." Dubler allows the user to select the type of input style (i.e. beatboxing, clapping, vocal percussion, etc.) and proceeds to assign MIDI values to the unique variations of sounds it receives. Once configured, the user can trigger MIDI values in real-time by replicating the input sounds. Unfortunately, this software requires specialized hardware to function properly and is set at quite a high price point. Another relatable line of products are drum-replacement plugins such as Addictive Trigger and Drumagog. These tools analyze and detect time-based onsets for an audio file and can replace the original hits with samples. A producer's typical use for this would be to swap out live-recorded drum clips, which may be too dull and/or uninspiring, for other pre-recorded samples while still preserving the original performance's rhythmic integrity. The downside of such a plugin is its inability to process an audio file with numerous instrument types (Kick AND snare) occurring simultaneously. Lastly, Melodyne is a plugin and standalone software that allows the user to manipulate the harmonic and rhythmic characteristics of an audio file. The process through which it functions was of interest to our team, as it shared some parallels to our initial implementation prospects: Audio exists on a DAW's track, audio is read into and analyzed by plugin, algorithms manipulate file and generate a new waveform, and then upon playback only the generated waveform is output to the remaining signal chain.

IV. Feature Set

Fortunately, all the fundamental features the team had projected to include were able to be developed. This includes an audio recorder for input, onset detection and support vector classification for analysis, and MIDI streaming for output. Currently, the application is designed to function with kick, snare, and hi-hat sounds – this is a slight redirection from the original plan which included more drums. Implementation features were also added such as a sample accurate metronome, the ability to change audio devices and the MIDI output in a settings menu, and numerous instances of error catching and handling. Lastly, A simple, yet effective GUI was developed that leads the user through the intended procedure of use. It also allows the user to manipulate finer details such as sample rate and buffer size.

The analysis components, aka the algorithm, utilized various DSP techniques in their processing. For the onset detection, the following techniques were performed to compute a novelty function: Fast fourier transform, flux, root mean square, and half-wave rectification. Afterwards, an adaptive threshold with a median filter was created and applied. As for the classification, the main procedure for analysis was computing the MFCC (Mel-frequency cepstral coefficients) of the audio signal – the code for this is a direct port of Librosa's MFCC function.

Due to time constraints, the decision was made to abandon all stretch goals including the plug-in port. Development on the fundamental features simply took longer than expected and the team did not want to stretch themselves too thin. Regardless, all necessary goals were reached.

V. Method

A. Tasks/Roles

The development of this application was split into two main categories: the algorithm and the implementation. Two team members were assigned to each category. The algorithm team focused on the analysis of the recorded audio, which involved onset detection and support vector classification. This was done initially in Python and later ported in C++. The implementation team focused on building the framework that encases the algorithm, dealing with user interactions and handling inputs and outputs. This was developed exclusively in C++.

B. Timeline

Once the scope of our abilities was realized, there were significant changes to the timeline. Here is a brief, updated view of how development proceeded – highlighting major events – starting from October:

- **October 2nd**
  Basic GUI designed and programmed
- **October 9th**
  Recording functionality complete
- **October 16th**
  Metronome complete
- **October 23rd**
  Onset Detection complete in Python
- **October 30th**
  Onset Detection port to C++
  MIDI Output complete
  Audio I/O Settings Menu complete
- **November 6th**
  Onset Detection bug fixes

Classification complete in Python
Error handling for implementation

- **November 13th**
  Onset Detection bug fixes
  Classification ported to C++
  Extensive testing of features
- **November 20th**
  Onset Detection bug fixes
  Classification bug fixes
  Accuracy tests for algorithm
  General implementation complete
- **November 27th**
  Onset Detection bug fixes
  Supplemental documents complete
- **December 4th**
  Standalone application complete

As can be seen, the application did not fully complete development until December 4th. This is in stark contrast to the original timeline, which suggested production would conclude on October 30th.

C. Resources

A variety of resources were used for development. One of the primary sources was JUCE, which is a C++ framework specifically for audio application, plugin, and GUI development. Depending on the operating system, the team members used Microsoft's Visual Studio or Apple's Xcode as IDE's. The other primary tool utilized was Python. The team used this environment to easily construct and test the algorithm before having to implement into C++, a language the team had much less experience with. Within Python, several functions from Librosa, an audio processing library, were utilized to conduct further testing, trials, and easy implementation. When implementing into C++, the team had to manually reconstruct these Librosa functions since they are not cross-platform. Lastly, the classification portion of the algorithm used LIBSVM, which is an integrated software for support vector classification, for machine-learning applications. LIBSVM has both Python and C++ versions, so the implementation of this library was not an issue.

VI. Challenges

Our team confronted numerous challenges – much more than expected – during the development of BeatVOX. The first deriving from our team's general inexperience with C++. A significant amount of time was spent investigating the general layout and basics of the language, and it was not until after one to two weeks that a clear direction and logic map had been solidified, at least pertaining to the first few goals. Fortunately, this did not impact the long-term timeline, as the remaining portions of project could be solved via permutations of basic techniques.

Due to the application's reliance on external factors (i.e. user audio devices, sound cards, external files, MIDI ports, etc.), error handling proved to be a challenge as well. Identifying possible errors involved much hypothetical discussion and experimentation/discovery. JUCE provides a convenient class called "AudioDeviceSelectorComponent" which aided in the handling of the audio device errors; however, the application still had to display to the user that an issue was present and appropriately disable portions of the application that would be affected by said issue. Since BeatVOX operates in a "GUI informs engine" manner, this led to another challenge faced: updating the GUI when engine processes had completed. Initially, this problem was encountered during the development of the audio recorder and was resolved via JUCE's "Timer" class. Although, as the application expanded, more GUI parameters required processing updates, enough to warrant a more practical solution. After much investigation, the implementation of JUCE's "ActionListener" class resolved the issue.

Another, and perhaps most frustrating, challenge was the process of implementing the algorithm (developed in Python) into C++. While this would have been inherently difficult, the process was significantly exacerbated admittedly due to the mistake of relying on Librosa's MFCC function for much too long in the classification code. Since Librosa is not cross-platform, the team had to dissect the MFCC function and rebuild it in C++. This oversight caused a setback of about two weeks, but fortunately the team was able to successfully complete the port. As for the onset detection, the main point of frustration was due to the differences in error handling between the macOS and Windows operating systems. the MAC team members were not receiving notice of numerous "vector-bounds" errors that were present in the code. This led to vigorous trials of debugging, tweaking, and cross-checking between the team members.

The final notable challenge was determining proper testing protocols. The team's inexperience made it difficult to devise methods to effectively evaluate the code's performance – in fact, the team's first forms of accuracy-testing involved simply "eye-balling" the values. This, of course, was unacceptable and the team searched for a more precise solution. Eventually, Python began to be used as testing environment, with the team printing vectors from the C++ code and loading them into various Python comparison functions. For example, the classification port was verified by comparing an audio file's C++ MFCC values to its Librosa MFCC values via a difference function ([A1 – B1, A2 – B2, . . .]). It was found that every test case resulted in diff values of 1E-4 or lower, mathematically affirming the accuracy of the port.

Some areas of development were surprisingly less difficult to construct than anticipated. These included the audio recorder, MIDI buffer and MIDI streaming configuration, and the implementation of LIBSVM. With regards to the recorder and MIDI, JUCE provided useful tutorials and classes (i.e. "AudioAppComponent," "MidiBuffer," and "MidiOutput") that allowed these procedures to be relatively simple, at least compared to original expectations. LIBSVM shared the same relative simplicity, with nearly all the relevant code existing within a small for-loop.

VII. Measure of Success

In our proposal paper, we outlined three points of criteria to measure our success:

- *Application can convert an audio file of relatively straightforward beatboxing into a MIDI file that is intuitively representative of the beatboxer's intention.*
- *GUI must be able to be easily interpreted by the user, with a straightforward design and properly labeled parameters*
- *All knobs provided in the GUI function properly and can elicit discernible changes in the generated MIDI file.*

With regards to the latter two, while time restraints did not allow for a survey or general poll, the team considers these to be fulfilled. The application's GUI has minimal controls with a disable/enable system that leads the user through the stages of use. Additionally, each button and slider are clearly labeled and have intuitive functions. It can also be seen that every included GUI parameter does in fact influence the state of the application in a significant manner. Again, an external and quantifiable measurement like a survey would be ideal for this and would be pursued if development is continued. Nonetheless, the team is satisfied with the results.

Concerning the first and most important criterion, the team considers this to be a success as well. While developing the algorithm in Python, the team was able to achieve a precision of 94 percent, recall of 88 percent, and f-measure of 91 percent for the onset detection and an accuracy of 85 percent for the classification. Furthermore, after implementing into C++, the team achieved difference values of 1E-4 or lower for all test cases when comparing the Python and C++ algorithm results. As for the audio recorder, functionality was verified via inputting pure sinusoids of various frequencies and graphically, as well as audibly, analyzing the output. The length in samples of the recording, as manipulated by the tempo, BPM slider, and sample rate, was also investigated and confirmed to be accurate. The MIDI output was tested by predefining vectors representing drum patterns and analyzing the result when streamed to external MIDI software. The team is confident that the application's overall performance, while of course not flawless, is up to par with the projected criterion. This was concluded via the team's own internal testing and judgement. Like the previous measure of success, time restraints did not allow for external testing and surveys but would be pursued if development is revisited. The team encourages users to view the demo video available on BeatVOX's GitHub for further proof of functionality.

VIII. Novelty of Work

The core element and skill of beatboxing is the performance, which captures moments that ordinarily might not have been included or realized in a written beat. These unexpected moments are what gives music its flavor, and is also what can encourage progress when there is a creative block. Tools that attempt to capture or interpret the creative intent of the user are bound to be applicable in music creation as players are always coming up with new ways to improve or bring character and uniqueness to their performances. BeatVOX's true novelty arises from its utility, speed, and ability to preserve the "humanity" of a performance.

IX. Deliverables

The deliverables can be obtained via downloading the files available under the "Released" branch on BeatVOX's Github. This includes a "README" tutorial document, the C++ source files (.cpp and .h), the JUCE project file (.jucer), an "AdditionalFiles" folder, and an executable. The downloaded folder must be placed in the user's Documents directory, all files must remain in their original folder configuration, and files must not be renamed. For Windows users, the program can be launched simply by opening the executable. Assuming JUCE is already installed, MAC users will have to open the .jucer file with Projucer, export into Xcode, then run the program from there. Unfortunately, the team was not able to properly create a macOS application file, despite numerous attempts.

X. Conclusion

In summary, our group's goal was to develop a standalone application that can convert an audio recording of beatboxing into its appropriate MIDI drum counterpart, which can then be manually passed into any drum sampler. This has a wide variety of uses within the songwriting and production realm. The development of this application utilized Python as well as the JUCE framework and consisted of two main components: the algorithm and the implementation. We were able to get the essential features working, but were not able to look into our stretch goals. Overall, while we were aware of some of the challenges ahead of us when we first came up with the idea, we fell short of truly realizing the complexity of the project as a whole. Fortunately, we were able put forth the necessary work to complete the task, and gained a substantial amount of knowledge and experience doing so. The standalone application fulfills the measures of success established from the proposal and has great potential if revisited in the future.