YouRap

Authors: Jiahao Zhou, Saransh Agarwal, Danielson Joseph: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of detecting the percentage of a user's rap (voice input) that is on beat. Currently, rappers train by either recording themselves and self critiquing their performance, or by asking another rapper to provide feedback. This process requires a lot of time and knowledge or some form of guidance, both not easily available to all. Our system aims to provide a rap learning environment that is easily accessible in terms of cost and availability.**

***Index Terms*— Digital Signal Processing, Music Technology, Noise Filtering, Onset Detection, Python, Rhythm, Tempo, Web Application Development**

I. Introduction

We developed a rap music teaching web application and a noise filtering microphone. The web application allows users to pick out background tracks and then rap over the beat. The user will then obtain feedback pertaining to the accuracy of their rapping. This accuracy is based on the number of beats that they were able to hit in their rap. This project is aimed towards those who are not yet good at rapping but wish to become better. Our project is important because rap is currently the most popular music genre in America but very few people know how to rap. There is a lack of teachers or software for learning. From our research, we were not able to find such a tool but the closest competitors would be Digital Audio Workstations like Logic Studio or Fruity Loops. Our goal is to determine whether a user rapping into a microphone is rapping on-beat given a backing track beat. The system must allow the user to choose a backing beat of their choice and then rap into a microphone. The user must then be given an accuracy score after they are done rapping. This will be displayed in a webapp. In addition, we will have a noise filter which allows our users to practice rapping in non-studio settings.

The rapping will also be recorded for playback to the user. The interface will also allow the user to choose or switch to different tracks. Upon recording, the interface will display a visualizer for the voice. The user will be able to pause. After they hit stop, the user will be presented with an accuracy score based on the number of beats they were able to hit. Hitting a beat is when a word is spoken at the same time that the beat is supposed to hit. Thus, the user will have an approximation of how accurate or on-beat their rapping was.

II. Design Requirements

Rap encompasses a wide genre of music that is different from individual artist to artist as a result of varying rap styles and schemes. For the purposes and scope of this project, we will consider common rapping styles based in 4/4 time signature in which the rapper hits at least one out of the four beats with an emphasized word. In this paper, a single beat will refer to one out of the four beats that comprise a single measure in 4/4 time signature. This is the predominant rap scheme that most commercial artists today use.

As such, one of the most important requirements is that we can identify single syllable sounds that coincide with a beat at 100% accuracy because all words are based on syllable sounds. We will measure this with a metronome beat at different beats per minute (BPM). A metronome is a tick at predetermined intervals. The interval will determine the BPM. We need to achieve 100% accuracy on these sounds. Next, we want to achieve 95% accuracy on detecting on-beat rapping. We define on-beat rapping as rap in which each word coincides with a single beat. In other words, the sample will consist of words in which the syllabus are spoken at the same time a beat is supposed to happen. Since this is the most basic form of rapping to a beat, we want an accuracy as close to 100% as possible. However, human speech naturally varies and thus it makes it hard to test this accuracy. According to a German paper on speech tempo variation, speech tempo varies about 5% in natural German speaking[5]. Even though this is in German, we can assume that English speakers naturally vary too. Thus, we chose 95% to account for this fluctuation in testing speech. Next we will test speech that is 95% accurate but has been shifted by half a beat. This means that none of the beats should be hit and the accuracy should be 0%. Our program needs to be able to identify the fact that the sample has been shifted. Thus, we want an accuracy less than 5% to account for natural variations in speech. Lastly, we will test on professional rapper's acapella samples. We need to obtain an accuracy greater than 90%. This is because we wish to account for the 5% but in addition, due to the fact that it is not slow speech, the program will have a harder time detecting accuracy. Thus we aim for 90%, which is just slightly lower than 95%.

We will deem accuracy to be detecting that a user is on-beat in three out of four beats in a measure. This is because human speech and rapping varies from word to word. In rap, certain rapping schemes may have the rapper hit different beats. However, a sense of flow is achieved by consistently hitting the first and last beats or more in between. Thus, averaged across a measure in 4/4, the rapper will still sound on-beat. In addition, if there is no sound detected for an entire measure,

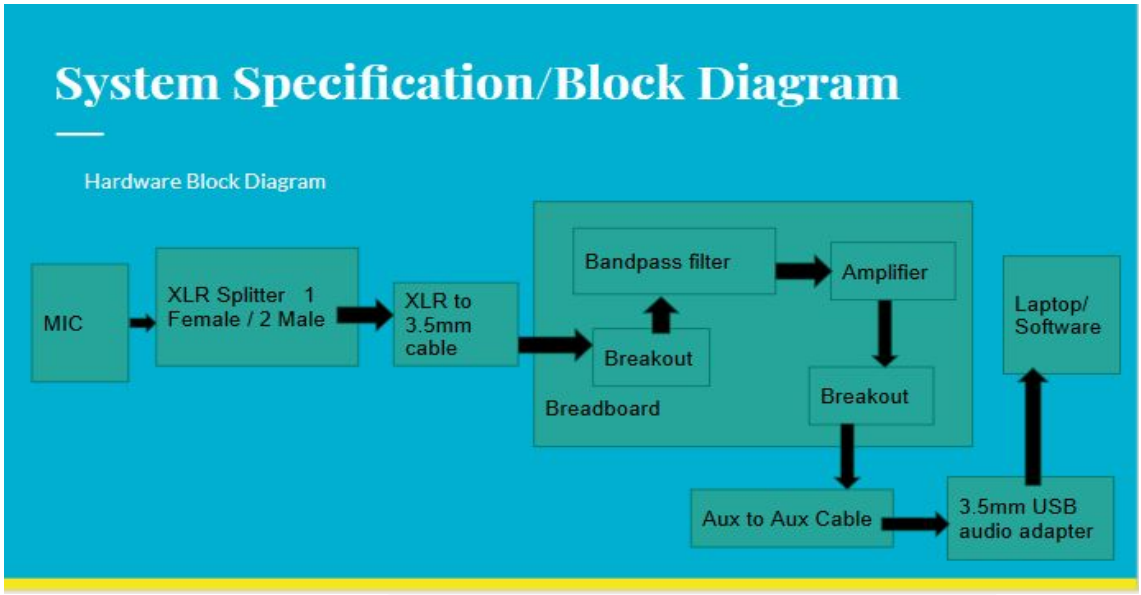
we will assume the user is pausing on purpose and that measure will not count in the overall accuracy rating.

In order to procure such tests, we will obtain wav samples of metronomes at BPMs ranging from 10 to 200 BPM in intervals of 10. We will be using microphones and MATLAB to generate the vocal samples.

III. Architecture and/or Principle of Operation

The system will consist of a hardware and software component as detailed below. The first diagram is a microphone filter that will help us eliminate noise from the input signal in real time.

The software component has two parts: Digital Signal Processing and Software Systems. All of our signal processing algorithms for beat detection were done using SciPy, a Python library. At the time of the design review, the plan was to use the MATLAB Engine API for Python which did not end up working. The Software Systems aspect is a webapp that allows the user to visualize and control the system in real time. Originally we had planned to give real-time feedback on whether the user is on-beat or off-beat. However, we changed to a percent score out of 100 displayed after a user is done rapping. We realized that detecting a successful case required us to wait for a minimum of two cycles of 4/4 beats, followed by repeatedly waiting for another cycle to end, making the feedback not really suited for real time, as it is delayed by definition. Furthermore, feedback given during a recording of a verse was deemed to be a distraction, interfering with the "flow" of thought to voice by our test subjects. This is because when a user is rapping, especially when freestyling (without written verses, straight from brain to mic", they cannot break focus from "musically" responding/flowing to the beat, unable to multitask. Post recording feedback, on the other hand, was welcome and understood.



Base Software Stack Image taken from reference [7]

IV. Design Trade Studies

A. *Microphone Filter*

The circuit filter was tested using the ECE oscilloscopes in the ECE lab, we used that to see if we got our desired frequency range for the circuit. We are going to use the circuit as a filter, so we have a clear starting frequency range and because filtering done in hardware is much faster than filtering done in software. The microphone output is fed into a laptop, so it helps to clearly know what frequency range we are dealing with. We built a circuit that can filter between 40Hz and 14Khz. A high pass filter was used to cut-off frequencies below 40Hz and allowed the higher frequencies to pass. A low pass filter was used to cut-off frequencies above 14kHz and allowed the lower frequencies to pass. The combination of high pass and low pass filters formed a bandpass that would just allow frequencies between the band of 40Hz and 14kHz to pass. We filtered between 40Hz and 14Khz because German microphone manufacturer Nuemann, states that above 14kHz voice gets an airy feeling and not much musical information is contained. And below 40Hz, is the sub bass range and contains little to no musical or voice information[3]. Nuemann has been in the microphone manufacturing business for decades, and their microphones are common in many professional audio studios.

The equation used to determine the frequency cut-offs for both the low pass filter cut-off and high pass filter cut-off is

$$f_{cut\text{-}off} = \frac{1}{2 * \pi * Resistance * Capacitance}$$

Fig. 1. Frequency Cut-Off Equation

The filters were implemented as passive filters instead of active filter, which could give the bandpass filter steeper slope cut-offs at 40Hz and 14kH, because resistor and capacitor components for passive filters were easily available from the 18-500 lab. Also, the initial desire was to make the filter circuit self contained in terms of not requiring additional connections when in use. So, that the filter circuit could be simply connected to any microphones and easily connect to the laptop without use for external power connections like a voltage supplier. Passive filters can be made with just resistors and capacitors, active filters use transistors which usually require connections to external power sources like a voltage supplier. The desire was for the system to be easily used by a consumer, so that if given the circuit they would just require a microphone and laptop with the web-app on it to use the system. The system was able to work without an external power source at first. However, when connecting the filters on the circuit and connecting the circuit to the laptop, the volume of microphone recording on the laptop was very low. An op-amp was added to increase the output signal of the filters and the volume of the microphone recording on the laptop. The addition of the op-amp required the use of a voltage supplier to power the op-amp, causing additional connections. The voltages applied to the op-amp, could probably be replaced with batteries, however, with the demo coming just keeping the voltage supplier seems like the safer choice.

The initial plan for the XLR Splitter was to have the input into the XLR be voice and music, the splitter would then output the signal to the circuit so that voice filtering was done there and then that signal would go to the laptop. And for the other output of the splitter, it would be outputted straight to the laptop with the use of a XLR to 3.5mm adapter and a USB to audio adapter, the output would be sent to the laptop to filter on the background music of the input. We decided to focus on the voice filtering and keep the extra XLR to 3.5mm cable and extra USB to audio adapter as spare parts. The XLR Splitter was kept in the system and used to extend the microphone cord so it would be easier for users to use the microphone.

Testing was done using oscilloscopes and function generators found in 18-220 lab.

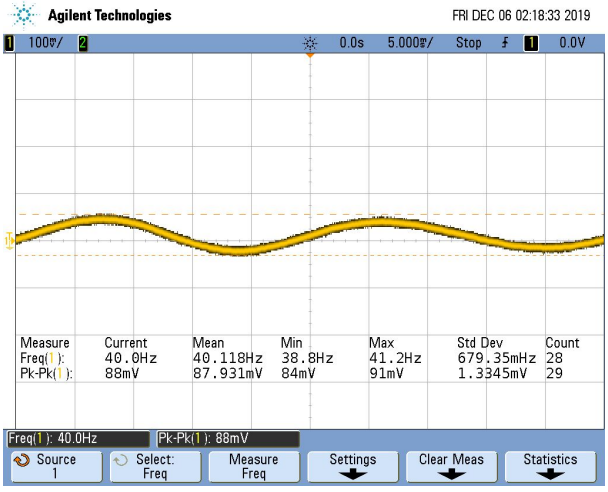


Fig. 2. Oscilloscope image of Bandpass filter at 40Hz

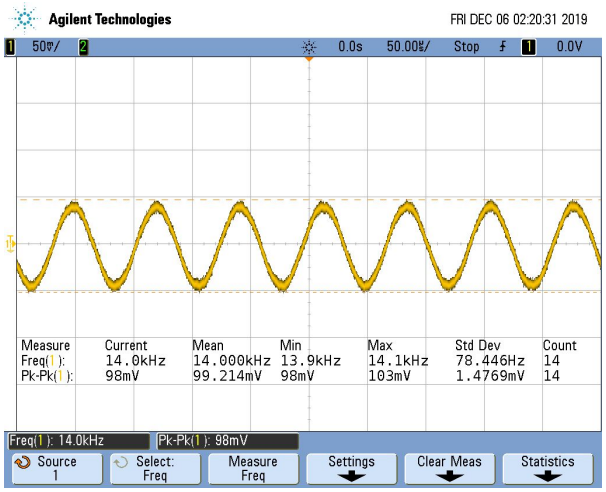


Fig. 3. Oscilloscope image of Bandpass filter at 14kHz

At 40Hz and 14kHz the system of low pass filters and high pass filters reduced the peak to peak voltage of the output signal of the system to approximately 10 percent of the peak to peak voltage of the input signal to the system.
So, if the input signal is 1 volt peak to peak, the output signal at 40Hz and 14kHz is approximately 100mV.

B. WebApp

React as it is most suited for single page applications we are building. Furthermore, the code can be converted to React-Native, a linked framework for creating Android and iOS apps easily, improving the extensibility of our app.

We are using Wad.js and Recorder.js libraries as they are wrappers around the HTML 5 Web Audio API allowing a lot of the processing to be done in the frontend. Furthermore Wad.js allows us to use the microphone input without having to implement it ourselves and effects.

There is a complication of using React and wad.js as it is designed to be used with Node.js and its package manager called NPM, however, due to the nature of using Django Rest Framework to serialize our backend. The frontend is completely independent from the backend, and either can be switched in the future, as long as the API remains the same. This makes our project modular and more future proof.

We are sticking with Django and not moving to a purely Node.js implementation as we have prior experience with it and it allows for flexibility in choosing the backend due to its robust ORM. Furthermore, the admin panel and API views created with the django stack, allows for the feature of uploading and managing beats and raps to be done without addition code.

C. Signal Processing

There are many algorithms available for analyzing music and detecting beats. However, because we are giving the user real-time feedback, we needed one that was fast and efficient. Machine learning (ML) algorithms are quite powerful and offer some of the best accuracy when it comes to audio signal processing. However, they have one major caveat and that is the training required. The strength of any ML algorithm no matter how good is heavily reliant upon the data it is trained and tested upon, and this requires both resources and time. For voice and speech recognition data is quite good. However, in the case of rapping there is a dearth of quality rap vocal databanks, much less data that has been preprocessed for training. Tempo detection is also relatively well-researched, which means we were able to find good algorithms without needing ML.

When it came to comparing the different algorithms we wanted ones that were fast, but still accurate. We ended up using a form of onset detection. First, the signal is preprocessed by filtering unwanted noise and frequencies. Next, peak picking will find the places where a user is rapping to the beat. The program will return

D. MATLAB

MATLAB was our choice initially for developing algorithms and performing most of the calculations. MATLAB is a programming language and a computing environment wrapped into an integrated development environment (IDE). It also features extensive GUI and visualization features. MATLAB is often the go-to for mathematical models such as signal processing. Our data consists of audio signals which meant that having a convenient way to visualize and interpret the data was important for decisions made regarding the algorithms. Being both a language and an IDE meant that all these features were integrated together and makes it easier to work with the data. Human visualization is an important and often overlooked aspect when it comes to many signal processes as it gives us an intuitive understanding of the data we are working with, allowing for easier debugging and decision making. Languages such as C/C++ may be faster and more optimized, but not by much when dealing with simpler algorithms. Since our real-time output is data averaged over the last 3 seconds, we have much more time to process and MATLAB was able to run all of our algorithms within the time frame. In addition, MATLAB also has a MATLAB Engine can be run on the backend server, allowing us to directly deploy our algorithms without complicating things.

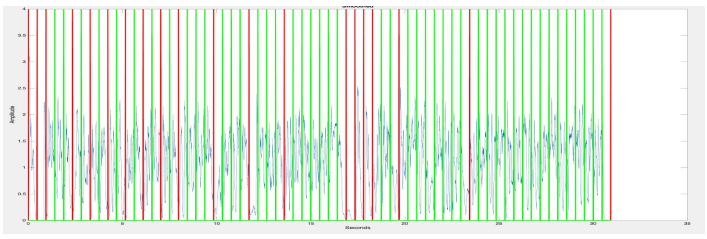


Fig. 4. MATLAB has a powerful and convenient GUI built-in. This is an example of the onset algorithm running on Kanye West's song "Stronger".

E. Python

Our final system ended up using Python due to a problem with the backend. The original plan was to run MATLAB on our backend with the MATLAB Engine API for Python. However, the MATLAB Engine did not run for Python 3 despite documentation claiming it could. Thus we switched over to making to implementing everything in Python. However, since most of our development occurred in MATLAB this was not much of a problem. MATLAB still provided many of the tools and was very convenient for development. As for Python itself, it is slightly slower than MATLAB, but with optimizations and just-in-time compiling, it can still meet our standards. Python also has many libraries that can mimic MATLAB such as NumPy arrays and Matplotlib plotting. The switch was also facilitated by the fact that the team members had more experience with Python.

V. System Description

A. Microphone Filter

We are planning to use the TONOR Dynamic Karaoke Microphone, and replace its cord with a XLR Splitter Cable (we are using the XLR Splitter Cable as a cable extender so the user can move the microphone easier without accidentally disconnecting something). The splitter cable is a XLR Female to Dual XLR Male cable. And then we plan to connect a TISINO XLR to 3.5mm cables to one of XLR Male connections of the splitter. The TISINO XLR to 3.5mm cable goes into a TRRS 3.5mm Jack Breakout which feeds the output of the mic into a bandpass filter in the breadboard circuit.

The bandpass is formed by connecting a group of high pass filters with a group of low pass filters. The number of high-pass filters used is four and the number of low-pass filters used is four. The high pass filters have their circuit configuration, the same resistance value of 400000 ohms and capacitance value of 10 nanoFarad is used in the same configuration for each high pass filter, that is so each high pass filter has a cutoff frequency of approximately 40Hz. The low pass filters have their own configuration, each low pass filter uses the same resistance value 11,330 ohms and capacitance value of 1 nanoFarad. The resistors and capacitors are used in the same configuration for each low pass filter, that is so each low pass filter has a cutoff frequency of approximately 14kHz. The output from the breakout connected to the microphone is fed into the first high-pass filter and the output of that high-pass filter is fed into the input of the next high-pass filter and so on until the output of the fourth high-pass filter is fed into the first low-pass filter. The output of the first low pass filter is fed into the input of the next low pass filter and so on until the output of the fourth low pass filter. The group of high pass filters and low pass filters connected together form the bandpass filter.

From there the output of the bandpass filter of the circuit feeds into an LM741 op-amp to amplify the output signal by approximately by a factor of 4.7. From there the output from the op-amp feeds into another TRRS 3.5mm Jack Breakout, then the output is transferred from the breakout with a KabelDirekt - Two Sided Aux Cord. One side of the cord connects to the breakout and the other side connects to a UGREEN USB Audio Adapters and is then fed into another USB port on the laptop. Filtered voice is going to be used by the webapp to make the beat detection work

B. WebApp

The system uses Model View Controller Architecture with Django as the main framework, and backend. React as the frontend, configured to work with HTML5 canvas for visualizations. Wad.js and Recorder.js are wrappers around HTML5 Web Audio API that are used for audio processing and storage. Communication between the front and backend was achieved using the Django Rest Framework responsible for serialization of data from the backend to the frontend, and vice versa.

The microphone and associated filter is plugged into the user's computer on any USB port and is detectable by the webapp. Permission has to be approved, which the user is prompted for in the web-app directly.

The Signal Processing part was originally done in MATLAB to make use of the optimized functions available, and was supposed to be part of the backend by using the MATLAB Engine API for Python, allowing one to use MATLAB as the computation engine for functions running in the backend. However, due to a discrepancy in documentation, it did not work with our version of Python, and the workarounds we tried to get it to work also failed. As a result, we refactored the code to Python using SciPy to allow for complete integration and the cost of performance.

C. Signal Processing

The digital signal processing algorithms will obtain rap vocal input data from the USB microphone filter and the backing track BPM from the backend. As our stretch goal, we will also determine the BPM of any backing track the user selects, in which case we will also be receiving the backing track WAV file. The core on-beat detection will be performed with onset detection, with the final accuracy based on how often the user hit at least three out of four beats and factoring in pauses.

a) Onset Detection

On-beat detection will be performed by analyzing the onset of a signal. In rap, and human speech broadly, there are distinct lulls and dips in the signal. The opening and closing

creates large and sharp onsets in speech that makes onset detection a good indication of speech tempo. Thus, we will be using this to perform vocal beat analysis.

The onset of a beat is the rising edge of the signal as the sound approaches its max amplitude where the attack happens. When one listens to rap, the beat is intuitively placed at the start of a word where the signal is strongest since that is the loudest part of the word.

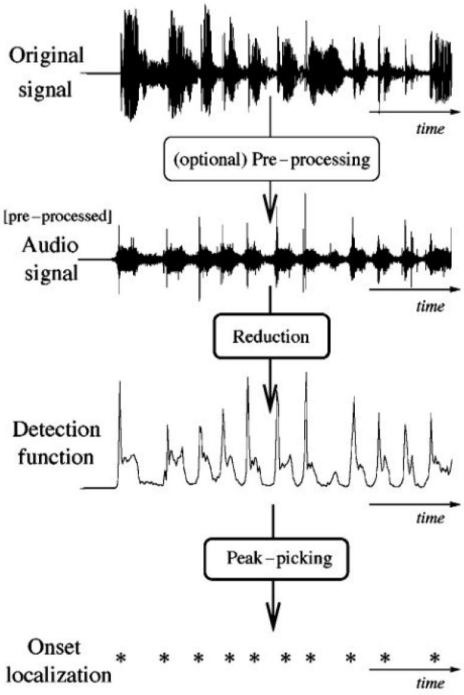


Fig. 5. Example of onset detection methodology [4].

In the onset detection algorithm, pre-processing is optional, however performing pre-processing helps remove noise and accentuate features we want. This pre-processing will occur in the microphone filter mainly for speed.

Next, normalization and reduction will create a clean detection function that allows the peak-picking algorithm to find where the onset location occurs[4]. A moving average filter was used to smooth out the signal before peak picking. The peaks were chosen by finding sudden increases in energy and the final amplitude reached was above a certain threshold. This signified that an onset was occurring, which is when the user was rapping.

b) Accuracy Score

After the peaks have been chosen, they will be compared against the BPM received from the backing track. If there is a peak that falls within +/- 8% of when a beat is supposed to hit, we count that as on-beat. If at least three out of the four beats in a measure are on-beat then that measure is determined as on-beat. In addition, pauses will be taken into account. If no sound above a certain threshold is detected for an entire measure, then that measure will not be counted in the final accuracy score. That threshold will be calibrated based on the microphone's ambient background noise when there is no sound input. The accuracy given back to the user will then be the percentage of on-beat measures. If the user ends the recording before the last measure is over, that measure will still be counted in the final score if at least two out of the four beats has elapsed, and the user has missed no more than one of those beats. In other words, in the last measure the user will have had to hit two out of three, or three out of four beats to be on-beat.

VI. Project Management

A. Team Member Responsibilities

Refer to Gantt chart.

B. Budget

Budget Spreadsheet Attached at the end of the report.

One XLR to 3.5mm cable and one USB to audio adapter is kept as spare parts. The XLR Splitter in the system was used to extend the microphone cord so it would be easier for users to use the microphone.

Two 150000 ohm resistors, and a 100000 ohm resistor put in series were used to get a resistance of 400000 ohms for each high pass resistor. There are four high pass filters.

One 10000 ohm resistor, one 1000 ohm resistor, and a 330 ohm resistor put in series were used to get a resistance of 11330 ohms for each high pass resistor. There are four low pass filters.

One 10 nanoFarad capacitor was used for each high pass filter. There are four high pass filter.

One 1 nanoFarad capacitor was used for each high pass filter. There are four low pass filter.

One 10000 ohm and one 47000 ohm resistor were used for the one op-amp

So, a total of 26 resistors were used and 8 capacitors were used on the circuit

C. Risk Management

Since we pivoted from chord detection to beat detection, our whole timeline was delayed by four weeks. In hardware if our noise filter does not work, we will use the Shure microphone which we are benchmarking against.

We had originally decided to let the user input the beat and rap over it. Sinec we do not have that much time left, we reduced our scope to just letting the user rap on our provided beat. Remaining risk is the software stack not working as planned. We are using technologies with great reviews and good documentation and a lot of successful applications, some already used by us before.

VII. Related Work

We did not find a project that used all of our components for this use case. There are companies like Bose and Sony that focus on noise filtering for their audio products like headphones. There are a lot of projects on Github which use React for Audio Visualization similar to reference [6]. Lastly, we did not find another rap beat detector, which was a novel implementation of the signal processing algorithm..

VIII. Summary

A. Results

We met or exceeded all of our expected accuracies except for on-beat rapping. We aimed for 95% but only hit 90%. However, we believe this was due to the poor quality of the samples tested. We recorded the rapping ourselves on a bad quality mic. In addition, our quality of rapping and the way in which we

Input Type	Expected Accuracy	Avg. Accuracy
10, 20, 30 . . . 200 BPM Metronomes	>99%	100%
80, 100, 120 BPM 4x4 Drum Beat	>99%	100%
On-beat Rapping	>95%	90%
Shifted Off-beat Rapping	<5%	3%
Commercial Rap Acapella Samples	~90%	93%

Fig. 6. Results of testing the algorithm on different inputs.

created the samples was perhaps not as clear as it could have been. However, the other accuracies were exceeded. In particular, commercial rap acapella achieved 93% accuracy, which is in line with what one would expect from professional rappers.

B. Future Work

We plan to include a more robust UI with more features and better performance by deploying the algorithm to a cloud service like AWS.

C. Lessons Learned

Integration takes time and will be ridden with bugs. Plan ahead of time.

It's important to verify every outside component/dependency by writing a unit test, instead of assuming they will work.

Each of our 3 team members focused on completely different fields of Computer Engineering leading to a very independent and disconnected workflow from each other. While this is beneficial in terms of ease of dividing work, getting help from each other was often not possible, and we were isolated in each of our contributions. In retrospect, reducing the breadth of our project and focusing on working on fewer aspects together would have been a more fun and rewarding experience.

References

[1] A. Robertson, A. M. Stark, and M. D. Plumbley, "Real-time visual beat tracking using a comb filter matrix," *Proceedings of the International Computer Music Conference 2011*, University of Huddersfield, UK, 2011.

[2] https://blog.usejournal.com/react-on-django-getting-started-f30de8d23504

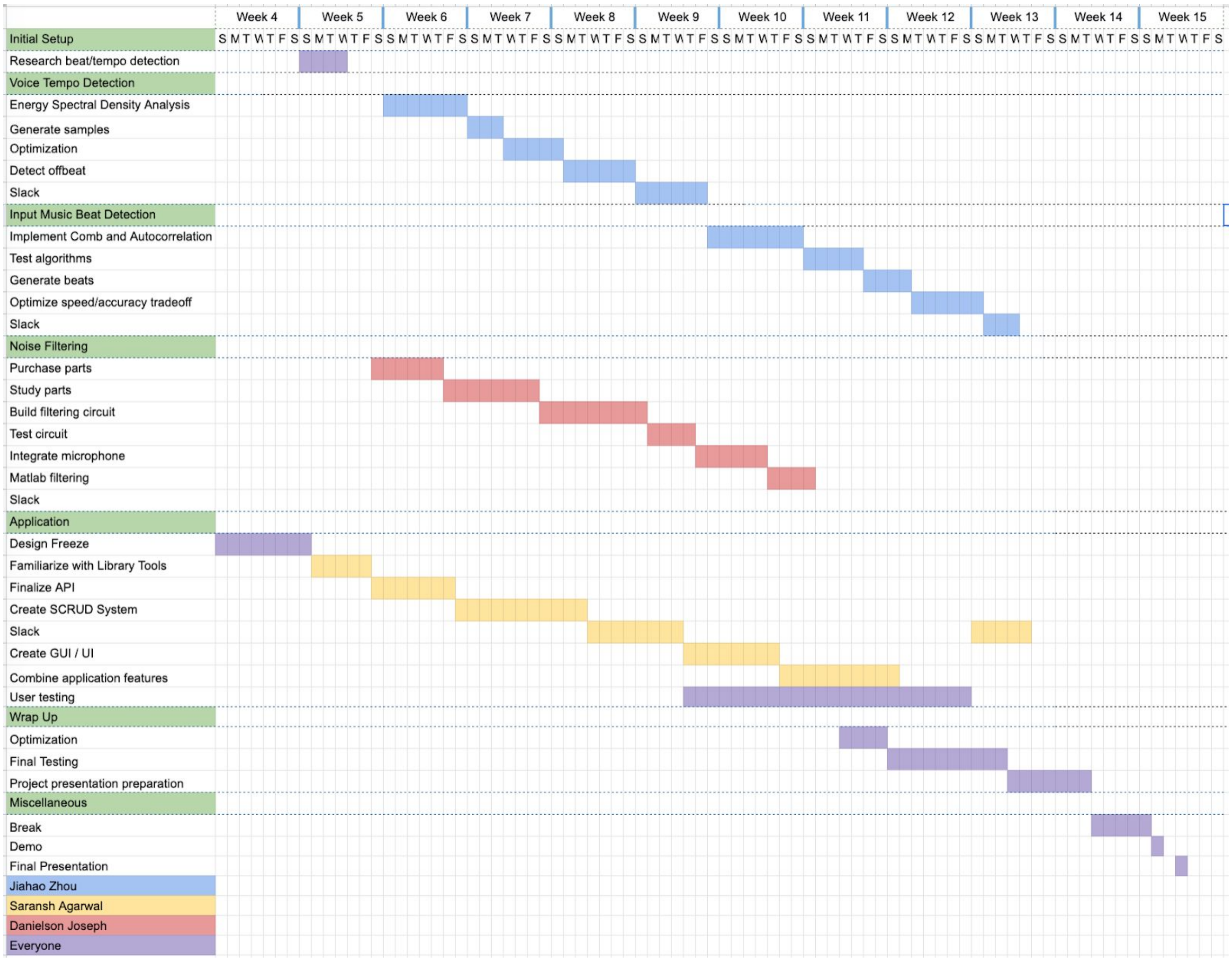
[3] https://www.thebroadcastbridge.com/content/entry/7759/how-microphone-frequency-response-relates-to-recorded-sound

[4] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies and M. B. Sandler, "A tutorial on onset detection in music signals," *IEEE Transactions on Speech and Audio Processing*, 2005.

[5] Trouvain, Jürgen. "Tempo Variation in Speech Production." PhD diss, 2003.

[6] Nash, Phil. "Audio Visualisation with the Web Audio API and React." Twilio Blog, Twilio, 21 Sept. 2018, https://www.twilio.com/blog/audio-visualisation-web-audio-api--react.

[7] Garner, Bennett. "React on Django: Getting Started." Medium, Noteworthy - The Journal Blog, 20 May 2019, https://blog.usejournal.com/react-on-django-getting-started-f30de8d23504.


Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
Week 15
Initial Setup
Research beat/tempo detection
Voice Tempo Detection
Energy Spectral Density Analysis
Generate samples
Optimization
Detect offbeat
Slack
Input Music Beat Detection
Implement Comb and Autocorrelation
Test algorithms
Generate beats
Optimize speed/accuracy tradeoff
Slack
Noise Filtering
Purchase parts
Study parts
Build filtering circuit
Test circuit
Integrate microphone
Matlab filtering
Slack
Application
Design Freeze
Familiarize with Library Tools
Finalize API
Create SCRUD System
Slack
Create GUI / UI
Combine application features
User testing
Wrap Up
Optimization
Final Testing
Project presentation preparation
Miscellaneous
Break
Demo
Final Presentation
Jiahao Zhou
Saransh Agarwal
Danielson Joseph
Everyone

Budget Table			
	Parts to be Bought		
Amount of Part	Parts	Cost	Source
1	OneOdio Headphones	$32.99	Amazon
2	UGREEN USB Audio Adapters	$17.98	Amazon
1	TRRS 3.5mm Jack Breakouts (pack of 3)	$7.58	Amazon
1	KabelDirekt - Two Sided Aux Cord	$7.99	Amazon
1	TISINO XLR Splitter Cable, XLR Female to Dual XLR Male	$12.99	Amazon
2	TISINO XLR to 3.5mm cable	$19.98	Amazon
1	TONOR Dynamic Karaoke Microphone	$19.99	Amazon
1	Shure SM58-X2U Microphone w/ XLR to USB Adapter	$199.00	Amazon
1	Breadboard	Free	ECE Department
8	Capacitors (values to be decided later)	Free	ECE Department
26	Resistors (values to be decided later)	Free	ECE Department
1	Laptop	Free	ECE Department
1	LM741 Operational Amplifier	Free	ECE Department
		Total Budget	$600
		Total Cost Without Shure Mic	$119.50
		Total Cost	$318.50
		Money left	$281.50