Physics 4BL Lab 12 - 2019 Fall Quarter

Sound to Sheet Music Conversion

- **Sydney Walsh**
  - Electrical Engineering
  - Python Coding
- **Jeremy Lim**
  - Mechanical Engineering
  - Data Collection
- **Kian Mohseni**
  - Mechanical Engineering
  - Recording

Introduction - Overview of Experiment

Objective

- Construct sheet music from audio data

Execution

- Record piano melodies at a specific BPM
- Isolate frequencies from audio and the times they were present
- Perform fourier transform on isolated notes
- Round frequencies and duration values to notes

Why?

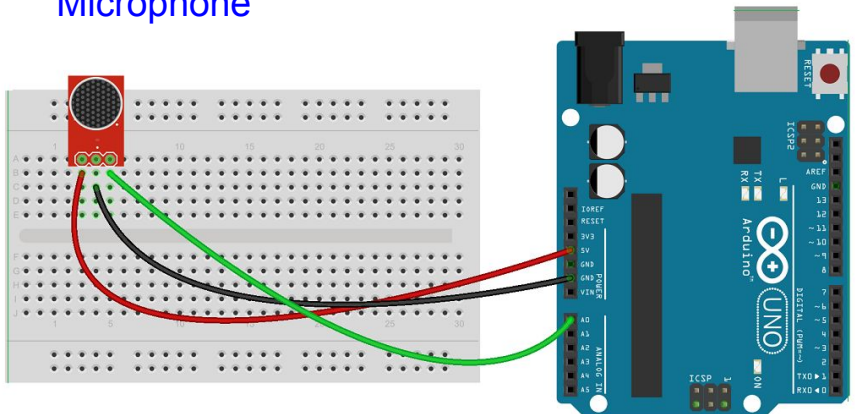
- Expansion of Week 4 Lab
- Relevant musical applications

Introduction - Overview of Experiment

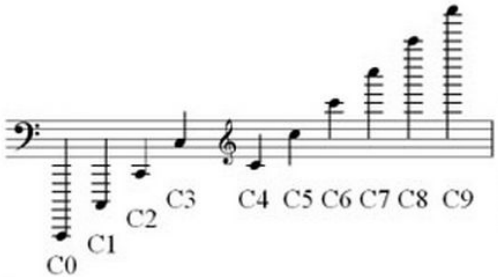
The only materials required for this experiment are an Arduino, a Microphone, and a Piano/Synthesizer.

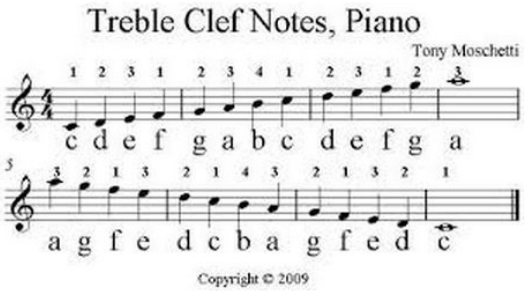
Microphone

Arduino



Piano





Theoretical Background and Predictions

Theoretical Background

- Using Fourier Transforms, we could convert our sound waves, which are a function of time, into graphs that show us the peak frequencies of each sound file and the intensity of each peak, allowing us to determine which notes were played. This should create a pattern of equidistant harmonics showing up on the Fourier Transforms for each note.
- For every octave (12 half steps), each note's frequency increases by a factor of 2.

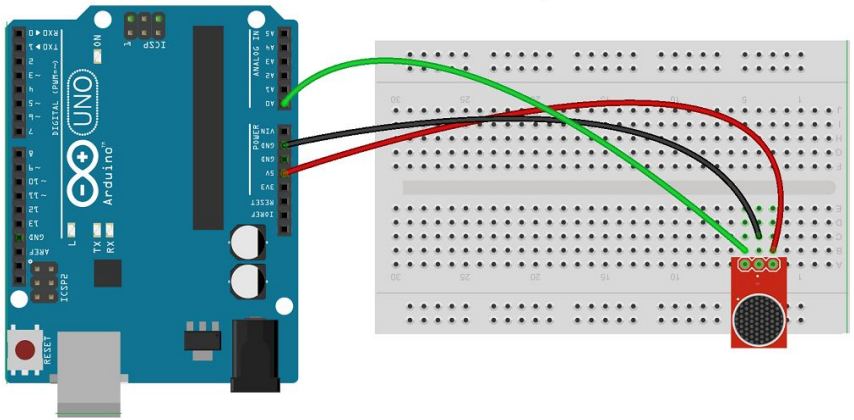
Expectations

- Results would mostly match software predictions

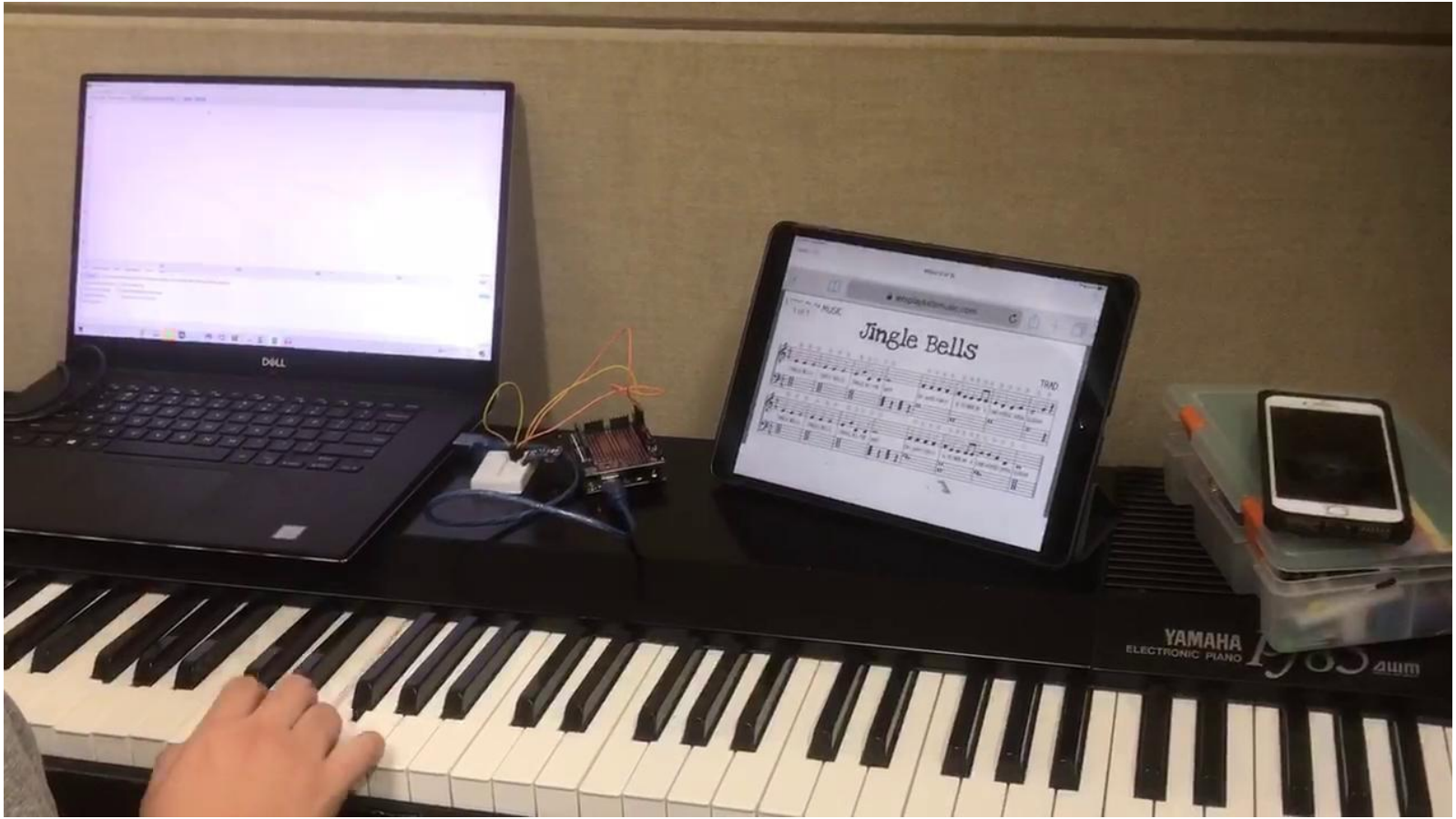
Error Sources

- Background noise
- Limitations of software thresholding
- Accuracy of playing
- Piano tuning

Conceptual Drawing of Experiment



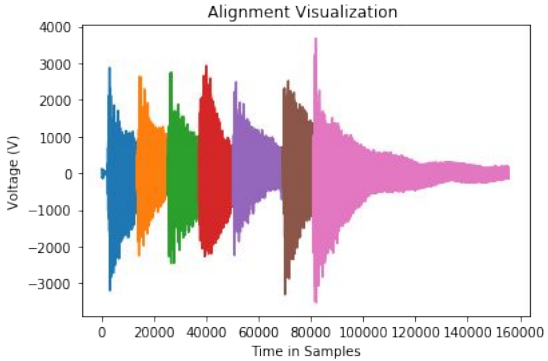
Video of Running Experiment

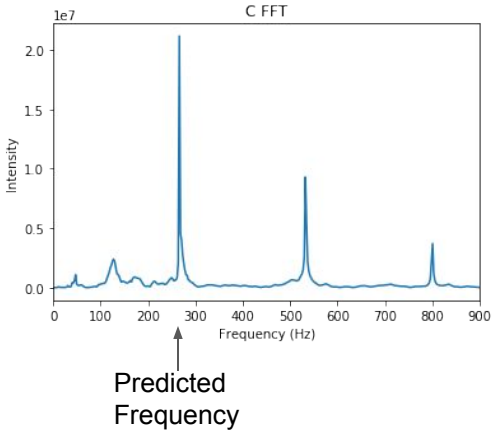


Data Taking and Analysis Method

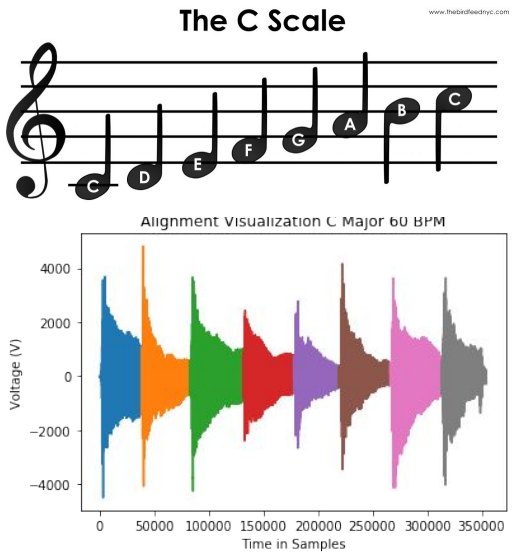
Program Execution

1. Truncate data and guess alignment of notes
2. Perform fourier transform on sequential chunks of determined notes
3. Predict resonant frequencies by looking for a pattern of harmonics in FFT
4. Map notes by rounding frequencies to closest theoretical match
5. Timing and duration determined by sequence of note chunks, length of chunks, and sampling rate
6. Determined thresholds for partitioning data and deciding on resonant frequencies

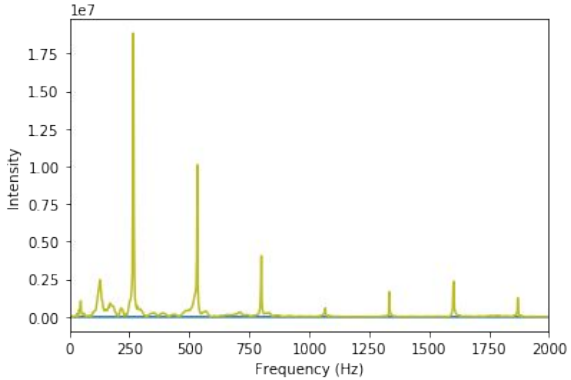




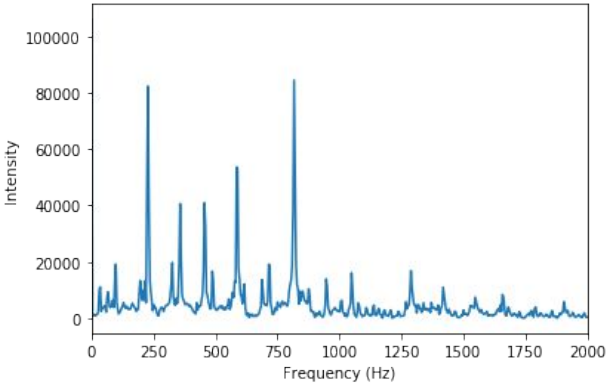
Result 1: C Major Scale 60 BPM



C4 Professional Microphone FFT



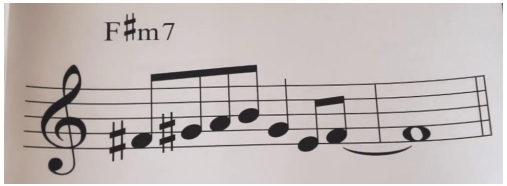
C4 Arduino FFT

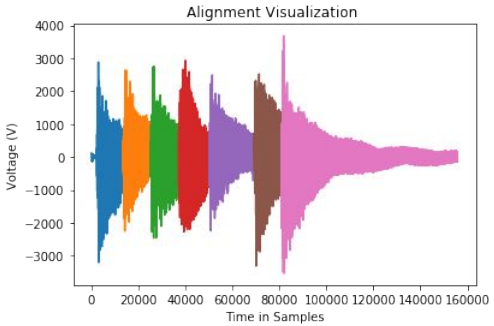


Theoretical frequency of C4 ≈ 261 Hz

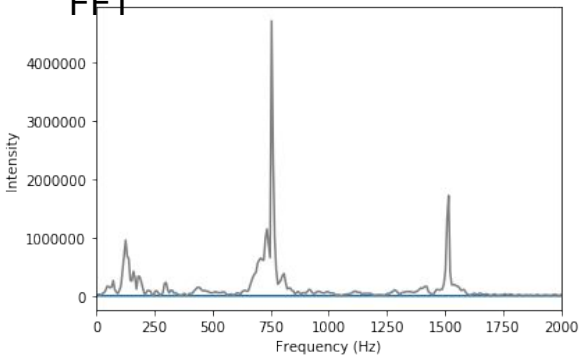
- Professional Microphone data translates perfectly
- Arduino data extremely inaccurate despite accurate alignment
  - A3, E4, **E4**, G4, **G4**, **A4**, **B4**, D5
  - We suspect this to be due to limitations of the Arduino microphone to handle the high amplitude strikes of the piano, which causes saturation and distortion

Result 2: The Lick 120 BPM

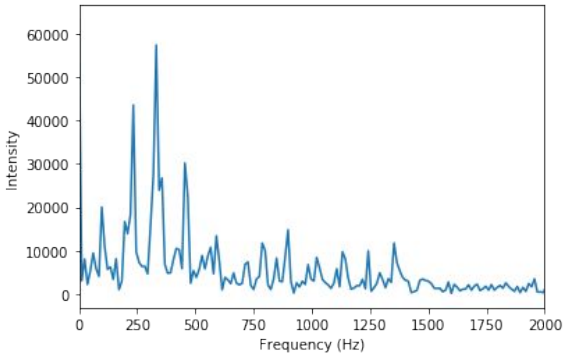




F#4 Professional Microphone FFT



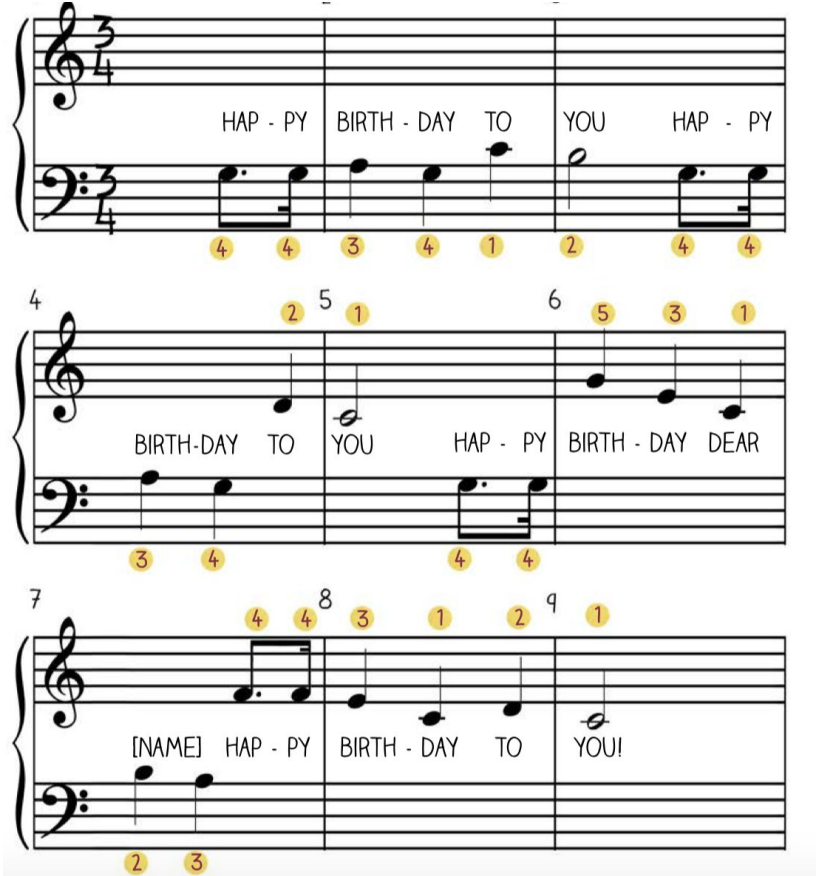
F#4 Arduino FFT

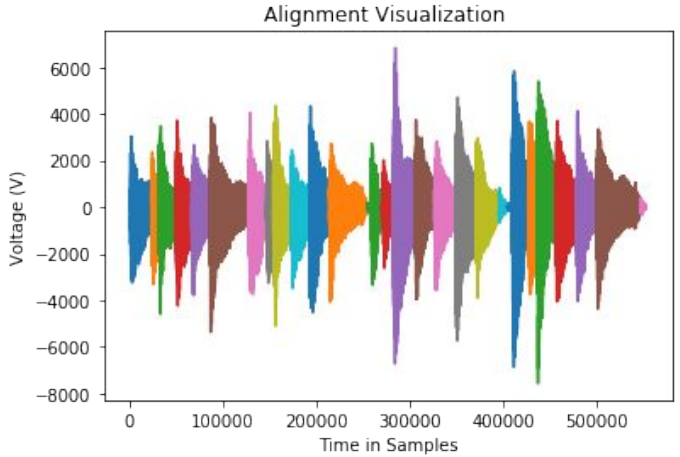


Theoretical frequency of F#4 ≈ 784 Hz

- Profession Microphone data maps notes and note duration perfectly
- Arduino data
  - E4, E4, **A4**, **B4**, B4, **E4**, E4

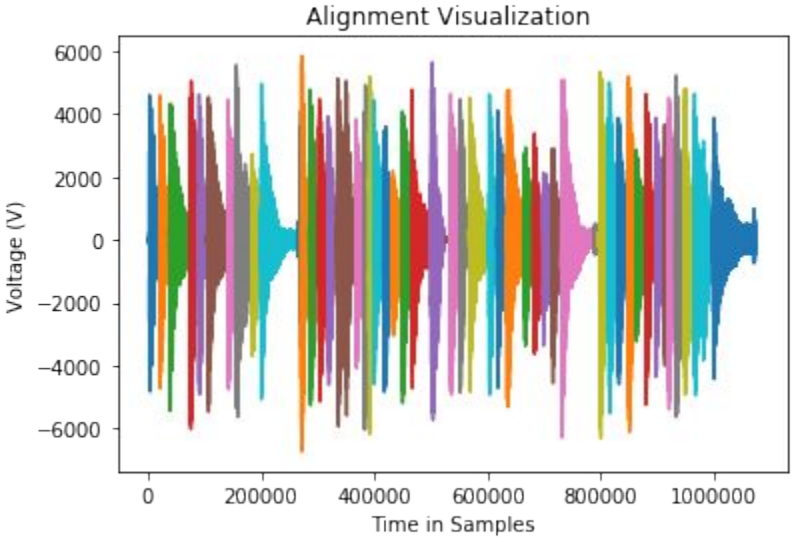
Result 2: Happy Birthday 120 BPM

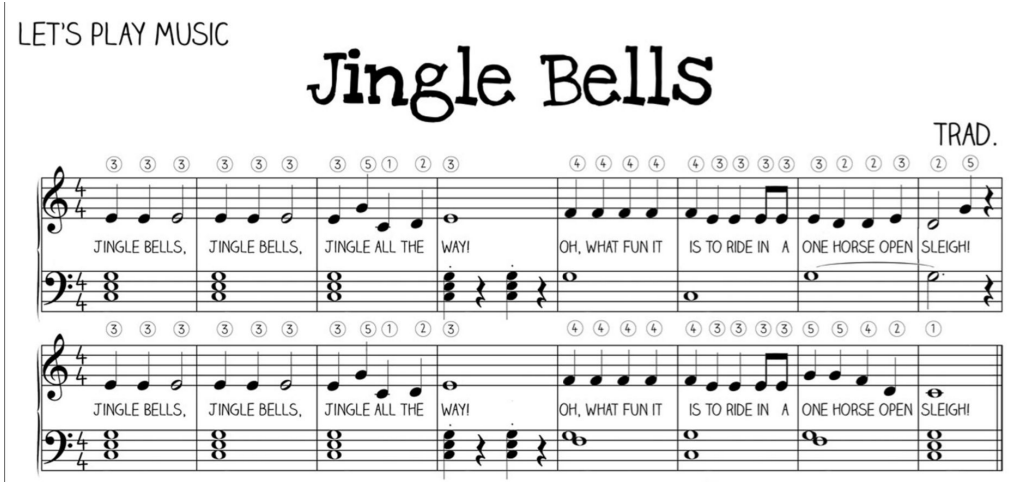




- Accurate alignment becomes more difficult with increasing BPM, varied rhythm, and length of dataset
- Professional Microphone: G4, G4, …, G4, **A4**, G4, D5, C5, G4, **A#2**, G5, E5, C5, B4, A4, **A2**, F5, F5, E5, …
  - Completely accurate except for 3 instances of incorrect partitioning, where a note not present is identified
- Arduino:
  - C#3, G#4, E4, G4, A3, B4, G#4, E4, F4, E4, ...

Result 3: Jingle Bells 160 BPM





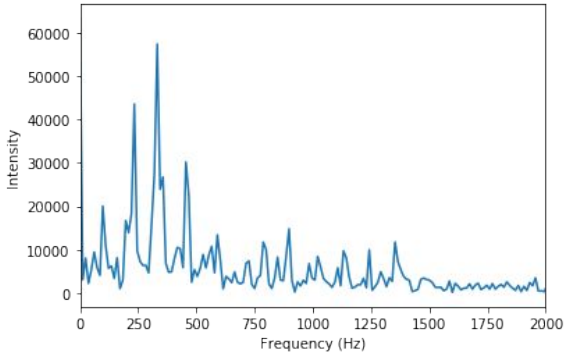
- Professional Microphone: E4, E4, E4, E4, E4, E4, E4, G4, **C4**, D4, E4, **B1**, F4, ..., G4, **B1**, E4, ..., E4, **B1**, ..., F4, **D4**, C4
  - Accurate except for omission of C4 and D4 and additions of B1
- Arduino: D#4, D#4, D#4, D#4, E4, E4, D#4, G4, A3, E4, E4, … ?????
  - Beginning frequencies are somewhat close, very inaccurate in comparison to non-arduino microphone results despite similar note alignment

Discussions

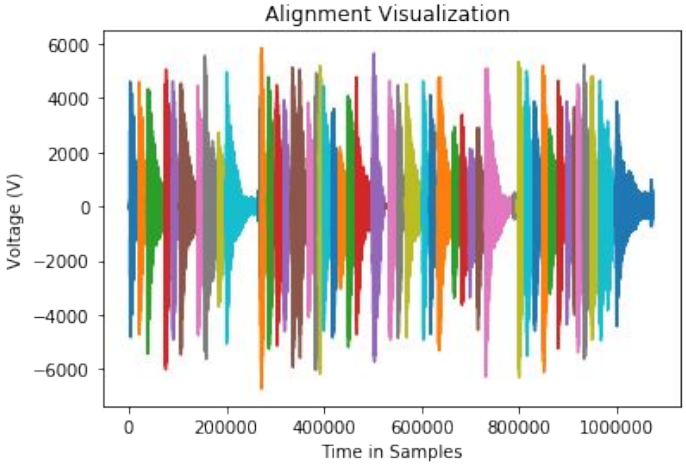
Error Analysis

- Determining thresholds
- Note alignment inaccuracies
- Arduino Issues
  - Using a different microphone gave better data
- Background noise and recording issues

F#4 (≈ 784 Hz) Arduino FFT



Indicates saturation of arduino data



Observing the note alignment of our Jingle Bells data, we can tell that our threshold values struggle to reconcile the conflict between accommodating small partitions for high BPM, low duration notes (16th notes, 32, etc.) and big partitions for high duration notes, where small partitions may result in the recognition of noise as notes.

Conclusions and Future Prospects

Summary

- Obtained mostly successful results with non-Arduino microphone
  - Results indicate future potential optimization of note alignment algorithm
- Very inaccurate Arduino results, probably due to saturation of audio data

Future Optimizations

- Recognition of multiple simultaneous notes with varied rhythm and rests
- Optimize thresholding using machine learning
- Accommodate note alignment technique to work for other instruments