Introduction

In this report, I will discuss the process of translating audio samples of human singing into digital music in the form of MIDI notes.

Motivation

The motivation for solving this problem comes from my experience with creating digital music. I often come up with melodies and songs that I wish I could quickly translate to an audio program such as GarageBand or Ableton. It takes time to set up a new project using Ableton and plug in a midi-keyboard. I am also not an experienced piano player, so I can't translate a tune in my head into a piano score on the fly. I desire a plugin where you could record a melody or beat boxing and have that translate to MIDI where you can then choose the desired instrument. Popular platforms have yet to implement this feature.

The Problem

The problem of translating the human singing voice into a digital and quantizable file has well-defined steps, but no single correct solution. The general steps required are as follows (Faghih, 2022):

1. **Pre-processing:** of the audio recording to isolate make vocals more prominent and unwanted noise lesser, for the purpose of making the analysis easier.
2. **Identify Voiced and Unvoiced Sections:** eliminating regions that can be identified as silence (white noise) or otherwise noises that were not produced by the singer. The purpose is to not focus the analysis in subsequent steps on samples that matter.
3. **Identify Note Onsets/Offsets:** Find when one note ends and/or when another begins.
4. **Identify Note Information:** Between the onsets/offsets identified in step 3, identify the pitch of the note, the volume and any other information that is required.
5. **Identify Properties of the Whole sample:** when analyzing all the notes identified, a tempo or time signature could be estimated to match what the singer had in mind.

Essentially, the problem is to boil down to the task of analyzing an audio signal (time-series data) and extracting information from it to represent it as music. There are various ways to represent music, but the most well-known and widely supported method for the digital realm is with: MIDI (Musical Instrument Digital Interface). A MIDI file represents simply encodes music in binary. Thus, MIDI itself does not make sound, it is just a series of messages like "note on" "note off" "note/pitch", "volume" and other additional data that is optional. More details on how this translation works is discussed later.

Approaches & Related Work

Pitch detection, onset detection, tempo detection, etc. are well researched areas in computational audio, both for musical data and vocals. Many approaches to identifying pitch require analysis in the frequency domain. Many onset detection approaches are also in this domain instead of considering energy. As will be shown, this can be especially important when the volume change is not noticeable between notes.

This problem is very close to the problem of identifying words from human speech. Frequency/pitch can be used to uniquely identify the constants or vowel. Finding onsets of new phonemes and their respective pitch is at the heart of most solutions. Singing could be viewed as a specific instance of this problem, where sounds are drawn out longer, but as will be shown in the next section, the act of singing introduces a whole sound and problems that make the analysis unique and challenging.

This information can be used to identify/classify songs. Once the data is extracted, it can be used to determine other things like vocal range, average pitch, tempo, scale.

Developing a Solution

The Human Voice

The human voice is It's a pitched non-percussive (PNP) instrument. That is, the set of instruments that produce sounds at specific frequencies but are not struck to produce a sound. The human voice can produce sounds at specific frequencies, which we perceive as pitch. This ability to produce tones of varying pitch allows singers to sing melodies and harmonies, making it a pitched instrument. It relies on the vibration of vocal cords and the manipulation of airflow through the vocal tract to create sound.

Although the human voice can produce specific frequencies and shares many similarities with musical instruments such as the violin, it's not so precise. The violon, like many other instruments is crafted to produce precise and consistent sounds. By nature, the organs that help us produce the same sounds are far less rigid or precise. As a result, extracting the information needed to recreate a sample is more difficult. As demonstrated in Figure 1, the piano and tuba create fewer more defined peaks in the plot of their FFT's (Fast Fourier Transform) compared to the human voice singing the same note.

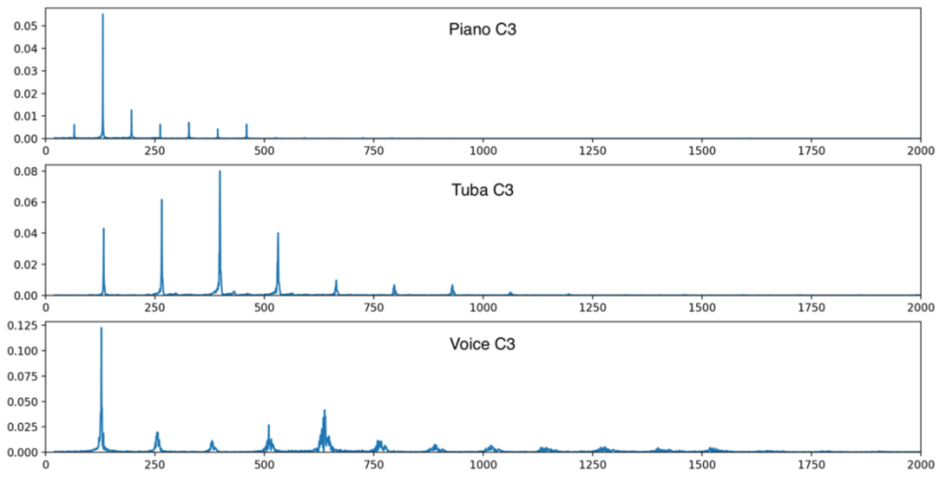


Figure 1, FFT (frequency on x-axis, magnitude on y-axis) of 3 instruments, Piano, Tuba and Voice, playing the note C3.

For one this means it has natural inconsistency with respect to pitch and time dynamics. Additional, notes cannot be changed instantaneously, there is a more graduate shift as we adjust our body to make a different noise. Lastly, unlike many instruments, Timbre is not consistent. From a physiological point of view, we have more variations of formant structures so that we can better articulate the different vowels and constants that we use to sound words. As result however, timbre and pitch may even variate within the duration of a single note (Faghih, 2022).

Moreover, instruments such as the violon are crafted to be consistent in size, shape, such that they produce are the same every time. The human voice on the other hand varies person to person depending on factors such as age and gender. According to (Santiago, 2023), the average frequency range for a male and female is 100-8000 Hz and 350-17500 Hz respectively while the average fundamental frequency (F0) lies in the range of 100-900 Hz and 350-3500 Hz respectively. Harmonics for males are around 900 – 8000 Hz and for females, 3000 – 17000 Hz.

Finally, it is important to note the different styles of singing. Legato singing involves smooth, connected transitions between notes, creating a seamless sound, while regular singing may have slight breaks between notes for clearer articulation (Faghih, 2022). This distinction is crucial in note onset detection algorithms, where legato singing would have less pronounced changes between notes. Vibrato is another form of singing referring to “Vibration” as singer rapidly sings back and forth between 2 notes. However, this is an acquired technique.

In conclusion, tackling the problem for the human voice is more challenging compared to standard instruments. It’s less of an instrument and more of a class of instruments, with each person having a unique sound. These properties played a large role in the development of a solution and exposed limitations in certain methods.

Note Onset Detection

Note Onset Detection (NOD) aims to identify the points in an audio signal where a sound event begins. There are a variety of methods that consider different domains. In general, there are two general NOD approaches (Grosche, 2010):

1. Data driven: Build statistical models by using Machine Learning on a set of training data. For instance, learning a probabilistic (hidden Markov) that exploits the rhythmic regularity in music.
2. Non-Data Driven: Directly calculating from the analyzed signal or its extracted features, the detection function may operate in time or frequency domain.

For this project, a non-data driven approach as it could solve the problem adequately with less resources and time. Many of the approaches I experimented with follow a similar pipeline:

1. Converted into a suitable feature representation that better reflects the properties of interest.
2. Then, a type of derivative operator is applied to the feature sequence and a novelty function is derived.
    1. captures certain changes in the signal's energy or spectrum.
    2. The peaks of such a representation yield good indicators for note onset candidates.
3. Finally, a peak-picking algorithm is employed to locate the onset candidates.

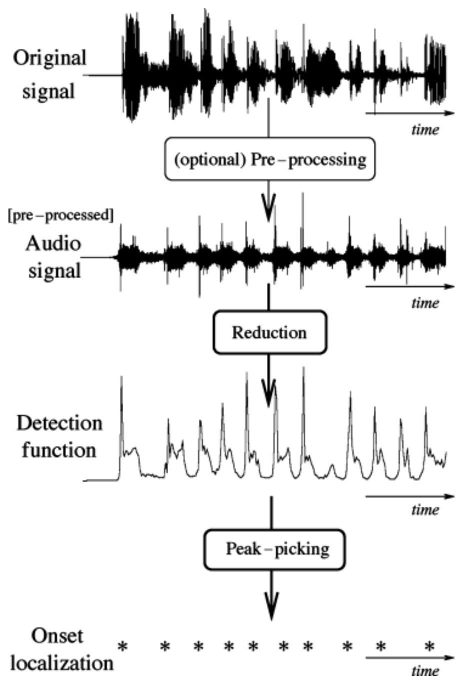


Figure 2, General Approach for Onset Detection. From Faghih, 2022

The “extracted features” that a method chooses to focus on is different. They range from: Magnitude, spectral flux, power any time derivatives of these (Grosche, 2010). For example, Figure 4 shows the algorithm when the chosen feature is spectral flux.

Magnitude

For notes that have a pronounced attack phase, onset candidates may be determined by locating time positions where the signal's amplitude envelope starts increasing. However, no percussive music with soft onsets and blurred note transitions, such as singing, does not have a clear attack faze. It is hard to determine or even to define the exact onset position. So, unless sung notes are separated by silence, simply looking for changes in sound-pressure-level is not adequate. Figure 3 shows that the difference between legato singing and singing where notes are separated on a time/amplitude plot. The changes in sound level are less pronounced as there is no dead silence to separate them. A change in amplitude between the first and second note is as large as some of the amplitude changes within the same note.

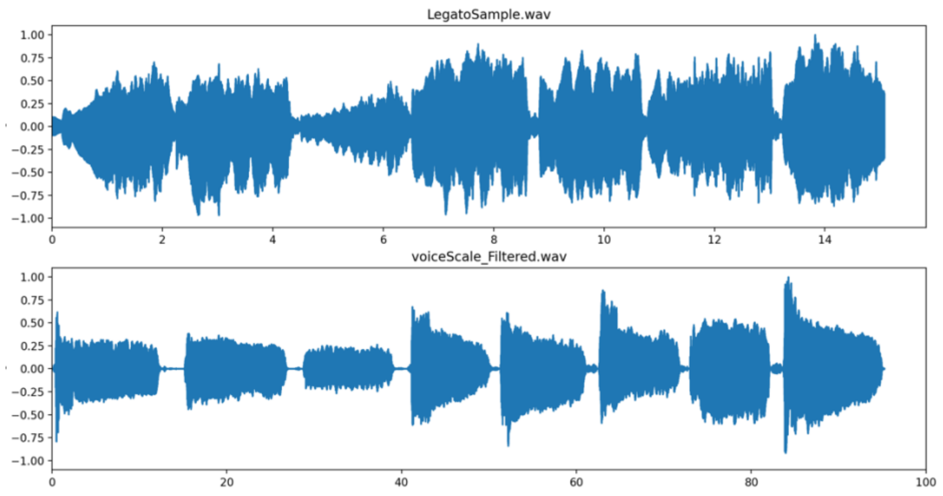


Figure 3, Two audio signals (x-axis = time, y-axis = amplitude). The signal on top is from a legato singer. The recording below it is that of a singer taking breaks before each note. Both are singing the C-scale.

Spectral Based Novelty

A spectral-based approach for computing a novelty function, relies on changes in frequency (spectral content). We quantify the spectral change between successive frames of an audio signal as **Spectral Flux**. This relies on a note onset causing a significant change in spectral content.

Experiments were done following the methods discussed in (Muller, 2019).

It involved computing the STFT (FFT over windowed signal), **compressing** the result to enhance the weaker spectral components. Then the **spectral novelty** is computed by summing over differences between frames. Finally, apply a peak-picking algorithm to get the peaks out of the envelope of the spectral novelty. These should in theory, correspond to onsets and offsets. Note that a peak-picking algorithm is useful in almost every method. It usually involves finding local maxes given certain parameters and then backtracking to the base of the peak.

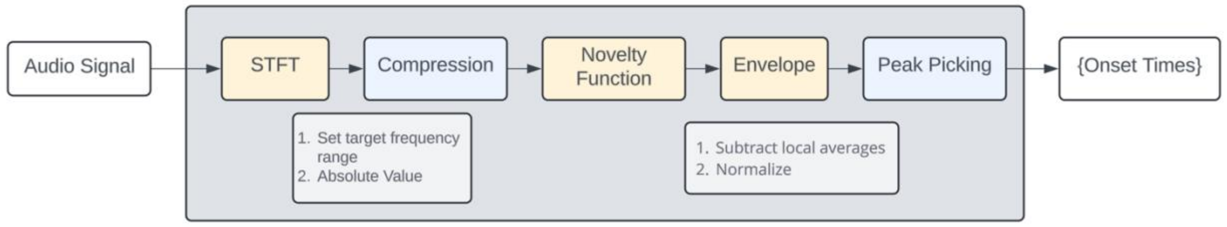


Figure 4, Spectral Based Novelty method for detecting Onsets. The input is an audio signal (time x amplitude) and the output is a set of times (hopefully) corresponding to note onset times.

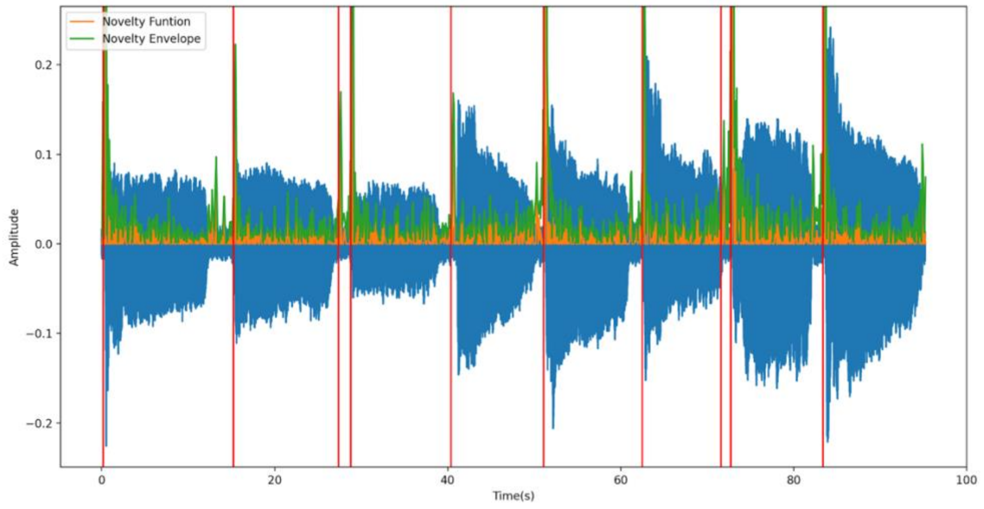


Figure 5, Voice recording of the C-scale. Novelty function is shown in orange/green and the detected onsets in red.

The steps shown in Figure 4 was ran on the audio signal shown in Figure 5. From first glance, the algorithm does a good job at identifying onsets and offsets by select the points where the novelty envelope (in green) is peaking. However, Figure 6 shows a case where an onset is not successfully identified. The end of the E note is not sudden, and rather, the singer slowly moves to the next note, F. As a result of the frequencies being relatively close and the singer’s slow transition, the spectral novelty does not see a significant change.

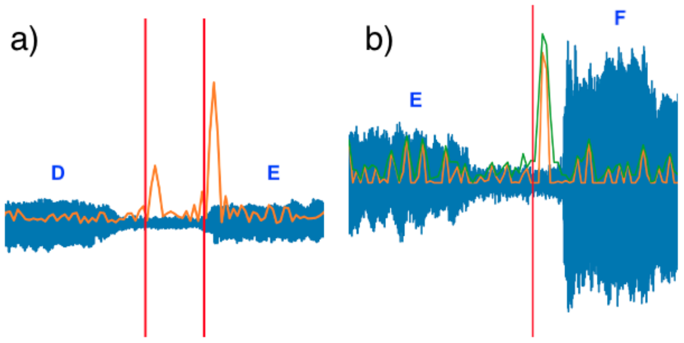


Figure 6, Zooming into section of Figure 4. a) shows the transition between the notes D and E. b) shows the transition between E and F. The offset is missing.

I attempted a few improvements to try and make the end of the sung parts of notes and the transitions between notes clearer This includes Filtering out white-noise before-hand, combining this result with an energy-based novelty and adjusting underlying parameters: STFT parameters, window-sizes for pre/post-processing, logarithmic compressions, peak-picking, etc. These options can improve the results, but ultimately, they make the method weaker in other areas.

Pitch Detection and Chosen NOD Method

The methods of for NOD that I experimented were to be used for that purpose only. After which, a frequency analysis could be done on each interval to determine the pitch. Upon researching for methods, I came across a method which uses changes in pitch to determine where note onsets and offsets occur (Faghih, 2022). Thus, what was two separate problems can be solved using one algorithm.

Before looking at pYIN, it is important to understand what the pitch of human singing really is. In music, pitch generally refers to fundamental frequency. However, musical pitch detection differs from basic frequency estimation due to the presence of harmonics or overtones, particularly prevalent in voices. When amplified or filtered by the singers head, higher harmonics can overshadow the original fundamental frequency. Despite the fundamental frequency potentially being absent or faint in the audio spectrum, the listener's ear and brain prioritize these stronger harmonics, creating the perception of a lower pitch (Micheyl, 2010).

To determine what pitch a human will hear, one needs a pitch detector, not a frequency estimator. There are many pitch detection methods, with varying strengths, possible weaknesses. Upon further research, I decided that autocorrelation in the time domain is preferred for better accuracy compared to FFT analysis, which can suffer from issues such as poor resolution at lower frequencies and complications from modulation, noise, or irregular waveforms (Micheyl, 2010).

Filtering will help. For example, I experimented with a low-cut filter to remove noise below 40 to 60 Hz, effectively reducing unwanted ambient and low-frequency noise captured by the microphone during recording. Also, reducing the gain in the 200 to 500 Hz range, known as "the mud," where excessive energy can cause a muddy or unclear sound in the lower mid-range of the EQ, particularly affecting vowel sounds such as "a," "e," and "u" (Santiago, 2023).

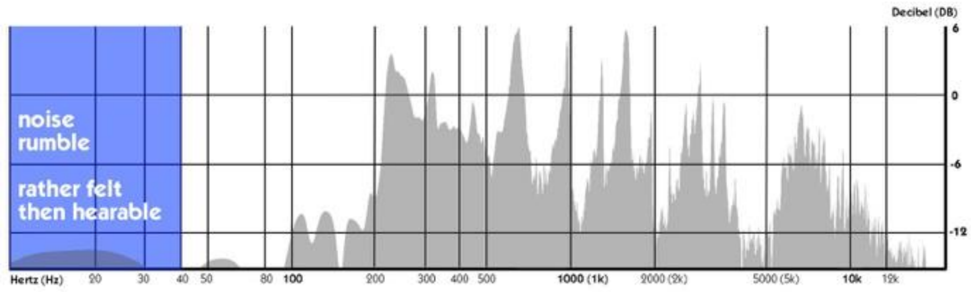


Figure 7, Showing the frequency composition of the average human voice sample. Image from Santiago, 2023.

Autocorrelation and YIN Algorithm

I chose YIN because it is a well-known algorithm based on autocorrelation in the time domain that was relatively easy to implement with *Python*.

Autocorrelation function (ACF) multiples/subtracts a signal by a basic periodic one shifted by some amount of time and then determines which shift maximizes the resulting value. That time corresponds to the period. By doing this over a windowed signal, we get how pitch changes over time.

In voiced sounds characterized by periodic patterns and a fundamental frequency (f0), the ACF exhibits a prominent peak at T=0, rapidly diminishing afterward. In contrast, unvoiced sounds lacking periodicity show an ACF peak at T=0 followed by subsequent peaks, reflecting the absence of a fundamental frequency or pitch.

The YIN algorithm first computes the ACF and calculates the difference function to identify the period corresponding to the fundamental frequency. However, it also performs cumulative normalization on the difference function and applies thresholding to detect the pitch candidates based on the local minima. The full algorithm I used to shown in

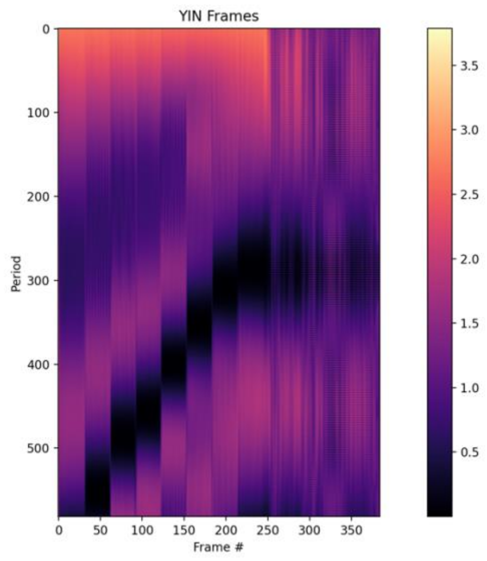


Figure 8, YIN frames show the signal split up into frames, and each period (y-axis) has some magnitude indicating how strong that period is in the signal.

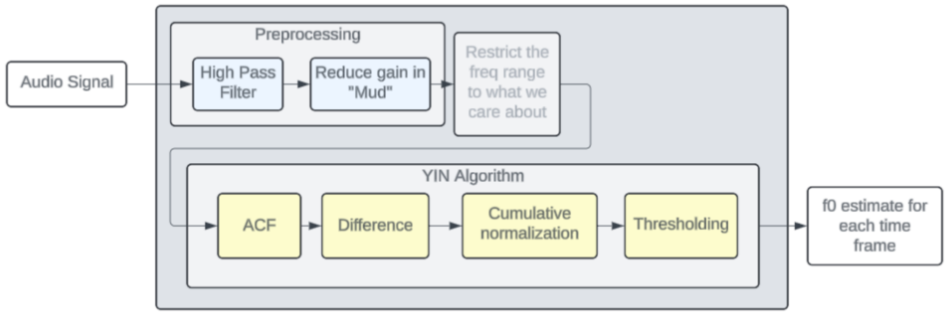


Figure 9, Pitch Detection Algorithm. The input audio signal is first pre-processed before being sent to YIN algorithm where the output is the F0 for each window of the signal.

Chosen NOD Method

Finally, now that we have a good method of extracting our desired feature for onset detection, pitch, we will apply a similar pipeline as Figure 4. But, first, why is pitch a better feature. Spectral changes often fail to differentiate the many variations in a singing voice because such as vibrato, soft onset (a long attack duration / vague envelope shape) and legato (transition from a note to another note where there is no intervening silence). The trajectory changes of F0 when moving from one note to another tends to be more noticeable. F0 is either erratic (white noise) before the onset or moves smoothly (legato), even when the consecutive notes are in the same pitch frequency.

I was drawn to the paper “A New Method for Detecting Onset and Offset for Singing in Real-Time” (Faghih, 2022). It’s the only paper I’ve found that tries to solve the problem specifically for singing.

The algorithm outline is shown in Figure 10. It begins by stretching the pitch contour to enhance subtle pitch changes within the human vocal range. Next, it determines the status of each point by differentiating the stretched pitch contour and extracting means, standard deviations, and the number of consecutive points sharing the same slope. Each point is categorized as None, Offset, Start Transition, End Transition, or Onset based on whether the slope exceeds a certain threshold relative to the mean and standard deviation. For instance, an Offset point precedes a Start Transition, while an Onset point succeeds an End Transition.

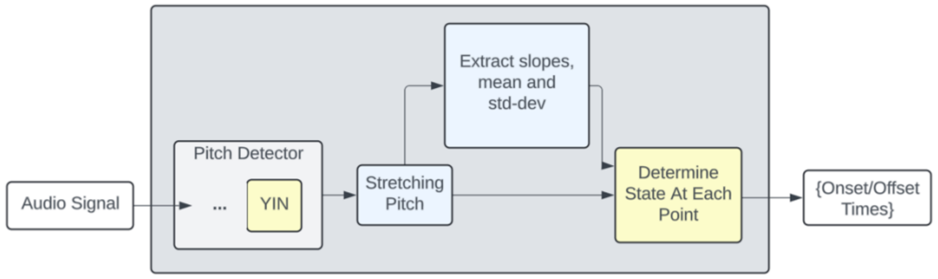


Figure 10, Full NOD Algorithm.

The process and output can be seen in Figure 11. It correctly identifies each onset and offset. The top graph shows the original signal with the annotations on it. The middle is the pitch contour, and the bottom is the slope analysis from step 4 that is used to determine the status at each point. You can see that the points where the slope (purple) exceeds the threshold (red) is where onset/offset happens.

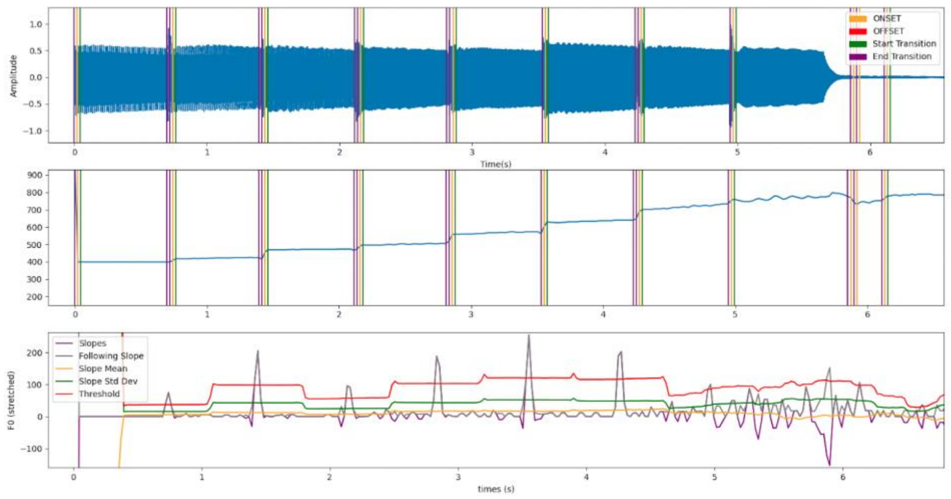


Figure 11, Applying the NOD Algorithm to a sample of Legato Singing. The top graph shows the input signal with the outputed onset times on top of it. The middle graph shows the same times over the extracted pitch. The bottom plot shows the process used.

Implementation

The code for the project is available on GitHub: https://github.com/rhudaj/Voice-to-MIDI-Translator. For more detail on what each file does and how to run the code, consult the READ-ME.md file at this link. This section will merely explain at a high level and show essential code.

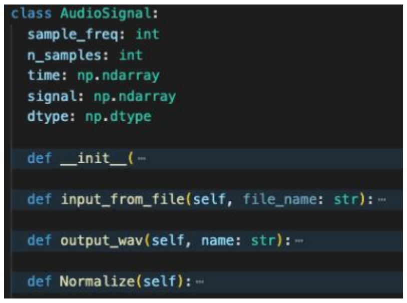
Python Setup

The entirety of project was built with Python because it is easy to read and has many useful libraries for handling all kinds of data. Four external libraries were used:

1. **Numpy:** adds support for large, multi-dimensional arrays and matrices (*ndarray*), along with a large collection of high-level mathematical functions to operate on these arrays.
2. **Librosa:** A python package for music and audio analysis. I am doing all the analysis and creating the key algorithms myself, I am only using some of their helper functions that don't do anything too significant (conversions between units, plotting a spectrogram).
3. **Scipy:** library to let me input a .wav file into a *ndarray*.
4. **MidiUtil:** for encoding note information into a midi-format binary file.

Input

The first step in the pipeline is to translate an audio signal into a data structure that can easily be worked with. To do this, I created the "AudioSignal" data structure which is a wrapper for two *ndarray*'s (**Numpy**). One store the values of time of each sample and the other to store the corresponding amplitude at that time. Other useful, metadata is included such as sample frequency, total # samples and the data type. *ndarray*'s are used over built in Python data structures because Numpy provides efficient operations on them for aggregation, Algebra, and traversal.



```
class AudioSignal:
    sample_freq: int
    n_samples: int
    time: np.ndarray
    signal: np.ndarray
    dtype: np.dtype

    def __init__(⋯

    def input_from_file(self, file_name: str):⋯

    def output_wav(self, name: str):⋯

    def Normalize(self):⋯
```

Figure 12, AudioSignal data structure.

Final Algorithm

Using my research and discoveries regarding the human voice and how that causes limitations in existing algorithms, I developed a pipeline to combine the onset/pitch detection algorithm with a few extra steps to get the MIDI representation. The final process is shown in Figure 13. In terms of implementation, a few important gaps in explanations need to be filled such as the volume and tempo estimation.

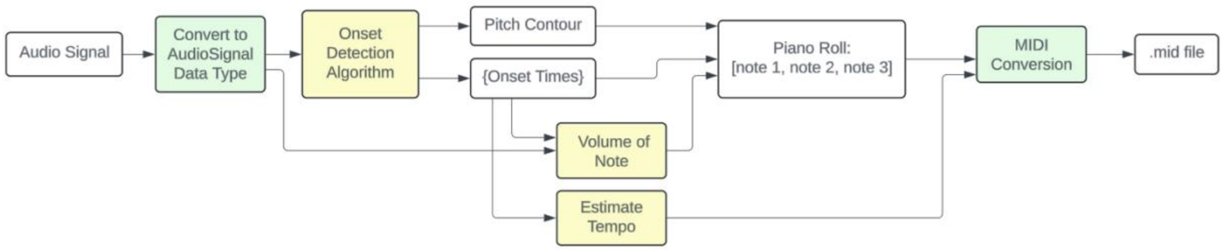
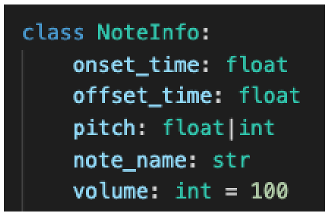

Audio Signal
Convert to AudioSignal Data Type
Onset Detection Algorithm
Pitch Contour
{Onset Times}
Volume of Note
Estimate Tempo
Piano Roll: [note 1, note 2, note 3]
MIDI Conversion
.mid file

Figure 13, Full pipeline to translate the Audio Signal into a MIDI file. "Onset Detection Algorithm" refers to the algorithm developed in the previous sections.

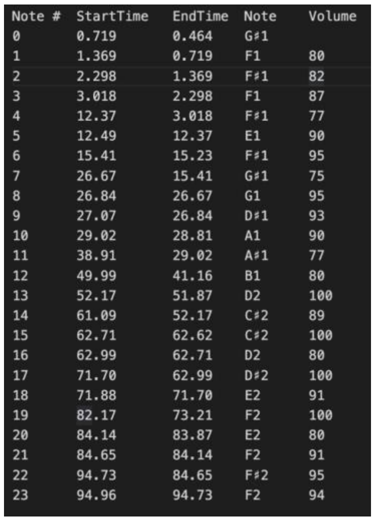
Creating a Piano Roll

The NOD algorithm returns both the pitch contour (the pitch for each frame of the sample) as well as a set of times indicating onsets and offsets. This data is translating to a sequence of *NoteInfo* objects, shown below. The piano roll for the legato voice-scale is represented in Figure 15.



```
class NoteInfo:
    onset_time: float
    offset_time: float
    pitch: float|int
    note_name: str
    volume: int = 100
```

Figure 14, NoteInfo, representing a single note in the sample.



```
Note #  StartTime   EndTime  Note    Volume
0       0.719       0.464    G♯1
1       1.369       0.719    F1      80
2       2.298       1.369    F♯1     82
3       3.018       2.298    F1      87
4       12.37       3.018    F♯1     77
5       12.49       12.37    E1      90
6       15.41       15.23    F♯1     95
7       26.67       15.41    G♯1     75
8       26.84       26.67    G1      95
9       27.07       26.84    D♯1     93
10      29.02       28.81    A1      90
11      38.91       29.02    A♯1     77
12      49.99       41.16    B1      80
13      52.17       51.87    D2      100
14      61.09       52.17    C♯2     89
15      62.71       62.62    C♯2     100
16      62.99       62.71    D2      80
17      71.70       62.99    D♯2     100
18      71.88       71.70    E2      91
19      82.17       73.21    F2      100
20      84.14       83.87    E2      80
21      84.65       84.14    F2      91
22      94.73       84.65    F♯2     95
23      94.96       94.73    F2      94
```

Figure 15, Displaying the piano roll output of a legato voice-scale.

Volume and Tempo Estimation

For each note, the average 'volume' is taken of its time interval to represent the volume of the note. Since the signal is stored as floating point in the range [-1, 1], the absolute value is taken, and the average is multiplied by 100 (full volume) and then only the integer part is stored.

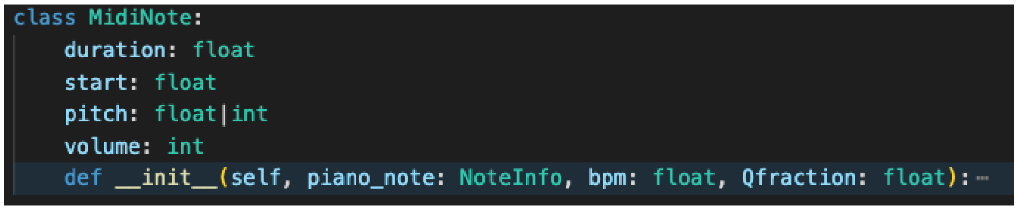
Estimating tempo is more challenging. Unlike volume which depends on one note, tempo depends on the start/end times of each note and the difference between them. The algorithm I followed was based on (Faghih, 2022) and it is very similar to the first NOD algorithm I looked at. The idea is to first compute the tempogram: local autocorrelation, that helps identify recurring rhythmic patterns in the music. Then, we find the peaks which correspond to the most prominent periodicities. The output of this step is the estimated beats-per-minute (bpm).

MIDI Translation

Finally, with the sequence of notes having a pitch, volume, and time, and having the bpm of the whole recording, we can put together a set of MIDI notes. The timing of the notes is identified with just the note duration and the start in terms of multiples of some user specified *note_fraction.* For example, ½ and ¼ refers to a half and quarter note respectfully. The calculation is explained by equations (1) and (2). The result is stored in a *MidiNote* object which is initialized using the *NoteInfo* object.

$$(1)\quad note\ duration = \frac{240 * fraction}{bpm}$$

$$(2)\quad note\ start = \frac{onset\ time\ (s)}{note\ duration}$$



```
class MidiNote:
    duration: float
    start: float
    pitch: float|int
    volume: int
    def __init__(self, piano_note: NoteInfo, bpm: float, Qfraction: float):⋯
```

Figure 16, MidiNote data structure, initialized with a NoteInfo object in the constructor.

Output

The final output is a .mid file. The Python library, MidiUtil, was used. It has a function to open a file and write the set information from the set of *MidiNote* objects that was created. The figure below demonstrates the result of three different audio signals, each with a different tempo, and style of singing. The left shows the input audio signal. The right shows the result of the MIDI file when imported into GarageBand without any other adjustments. These software's know how to read such a file and adjust the bpm based on it. As shown, the MIDI notes do a good job at matching the singing, even for the case of legato singing in the second graph.

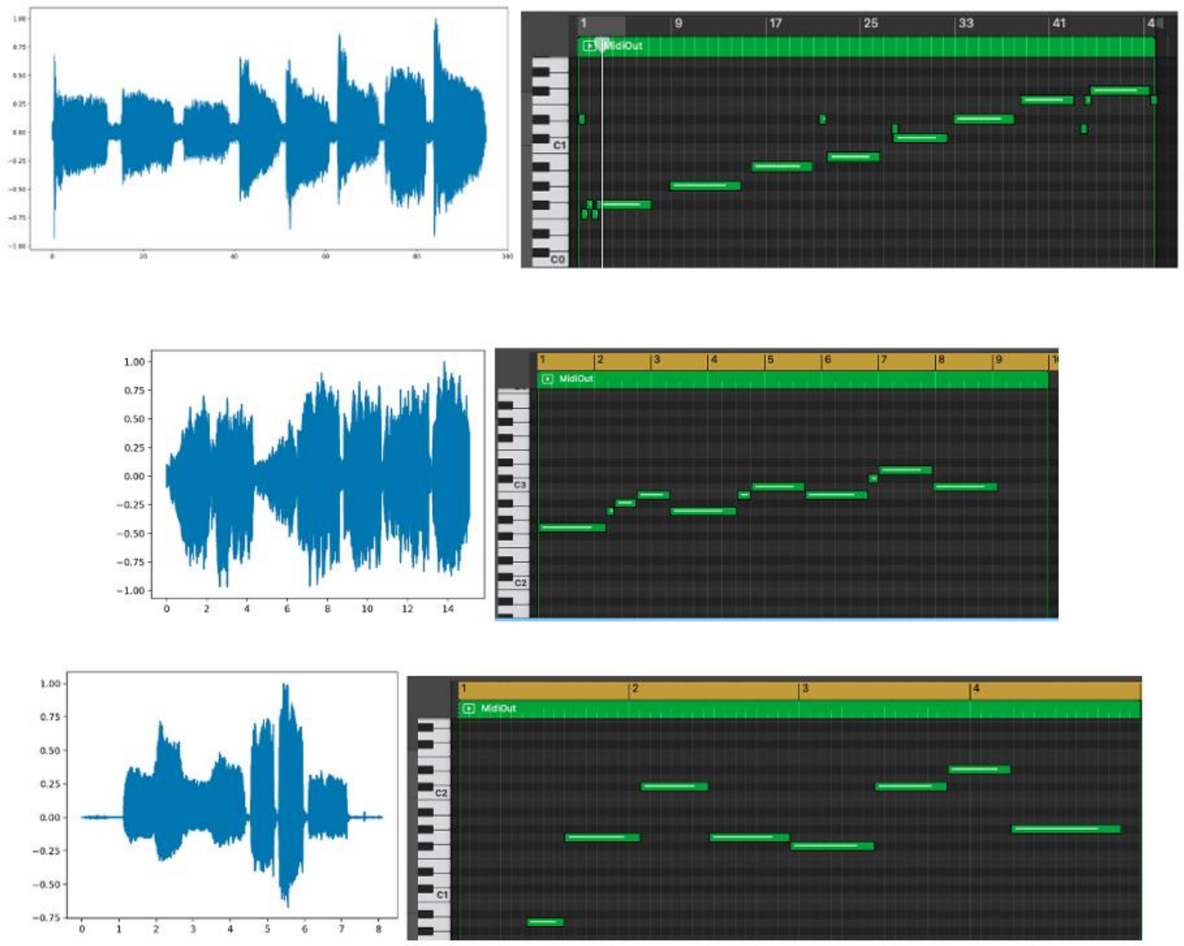


Figure 17, Showing the algorithm results for three unique audio signals.

Interface and Utilization

Throughout the process I've described, certain sub-problems require parameters. A *Settings* object was defined to act as the main interface to control the algorithm. This simply stores all the user-defined constants and has function to set them based on a string from the command line. At the time of this report, the best I could do for using the interface is through the command line. Simply specify the input file and a set of optional parameters. The key list of parameters is explained in Table 1. These are those parameters that will be important to set based on the data that is inputted and the type of person singing. The other parameters best kept unchanged, so they are not included here.

Table 1

List of key user-defined parameters to manipulate the algorithm.

Name	Type, Default Value	Description
note_min	String, “E2”	Lowest note that will be considered for the PD algorithm (82 Hz).
note_max	String, “B5”	Highest note that will be considered for the PD algorithm (987 Hz).
frame_length	int = 4096	Many of the algorithms require aggregation over windows of the signal, called frames.
note_fraction	float = 1/4	Specifies the quantization to apply while translating to MIDI. Defaults to ¼ which refers to a quarter note.
min_note_sep	float = 0.1	Minimum length of a note. Any duration of pitch less than this quantum will be considered as an outlier, not as a new note.
p_sustain	float = 0.9	Probability of staying in the same note for two frames in a row.
p_stay_silence	float = 0.7	Probability of staying in the silence for two frames in a row.
p_spread	float=0.2	Probability that the audio signal deviates by one note due to vibrato.

Testing - Datasets

There is no direct way to confirm if the MIDI output is correct. After all, the output is somewhat subjective. There’s no one way to translate the data to a discrete MIDI file, some data will be lost, and it is up to the person recording to decide if the translation is adequate. However, the key parts of the algorithm such as the PD and NOD could be tested.

First, I tested the algorithms with real instruments. It was able to reproduce notes perfectly. As explained, a piano has hard onsets and clear fundamental/harmonic frequencies. These were tested using samples I created myself in GarageBand.

Next, I tested the PD algorithm on individual singing notes that I got from Dawn Black's Singing Voice Audio Dataset (Caro, 2017). The frequencies of the different sung notes from the C-scale are shown in Figure 18. The fundamentals of each note, generally corresponding to the first peak, were identified correctly.

Furthermore, the NOD algorithm based on (Faghih, 2022) was tested using the same datasets that they used. I successful matched the results from the original source who implemented it in a different programming language.

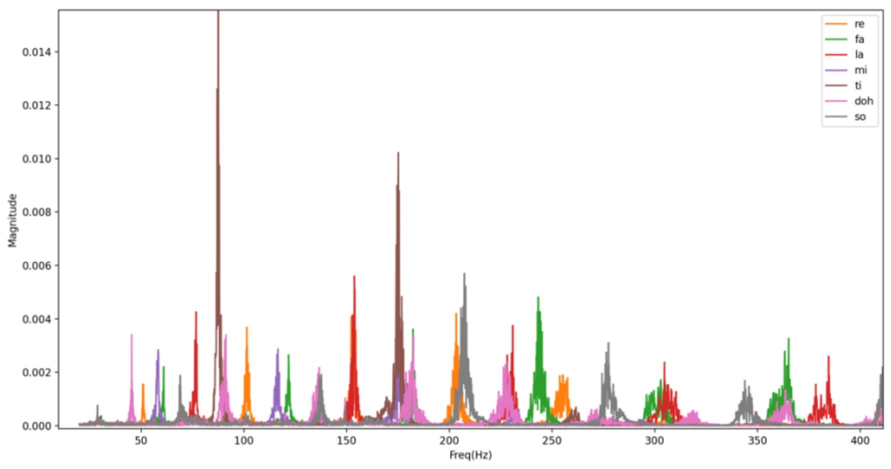


Figure 18, FFT plotted for each note sung from the C-scale.

Conclusion

Overall, the system is currently able to translate voice well. I have tested it on my own voice recordings and the notes have been roughly 80% accurate. The system performs perfectly for singing that has long notes and clear transitions. For other types of singing, it is more error prone. With some adjustments of the parameters however, it can work better for a legato singer.

Limitations and Improvements

The current iteration of the audio-to-MIDI translation project is designed to handle single-channel audio, catering to one person singing a single note at a time.

Further, Runtime optimization stands out as a significant area for improvement; the iterative process used in the current implementation could be streamlined and potentially combined to enhance efficiency, possibly leading to real-time performance, which would greatly enhance the user experience.

Furthermore, the project's user interface is currently limited to a command-line interface (CLI). With the properly modular coding I did, there's a clear opportunity to develop a more intuitive graphical user interface (GUI), by using an audio plugin compatible with existing software or an exposing the code as an API.

Moreover, the current system requires users to manually adjust parameters each time they use the software, which can be cumbersome and inefficient. Moving forward, there's a need to implement a more adaptive approach where parameters are adjusted dynamically based on user profiles and singing styles. An existing machine learning model could easily interface with the code and adapt the parameters automatically based on the input.

Lessons learned

Reflecting on the development journey of the project, I've gained valuable insights that will shape my approach to future projects. Firstly, I've come to appreciate the importance of thorough research and planning before diving into implementation. While the initial design of the system addressed the core problem, I now recognize the need for a more comprehensive understanding of user requirements and potential challenges to inform a more robust solution.

Moreover, I've learned the significance of iterative development and continuous refinement. Throughout the project, I encountered various roadblocks and limitations that required adaptation and adjustment. Additionally, while I worked independently on this project, I recognize the value of seeking input and feedback from peers and mentors. Collaborative brainstorming sessions and code reviews can provide fresh perspectives and identify potential blind spots, ultimately leading to more innovative solutions.

Lastly, I've learned the importance of embracing experimentation and embracing failure as part of the learning process. Not every approach will yield the desired results, and setbacks are inevitable. However, each failure presents an opportunity for growth and learning.

References

Müller, M., & Zalkow, F. (2019, November). FMP Notebooks: Educational Material for Teaching and Learning Fundamentals of Music Processing. In ISMIR (pp. 573-580).

Santiago, K. E. y. (2023, August 31). EQing vocals: What's happening in each frequency range in the human voice. Flypaper. https://flypaper.soundfly.com/produce/eqing-vocals-whats-happening-in-each-frequency-range-in-the-human-voice/

Faghih, B., Chakraborty, S., Yaseen, A., & Timoney, J. (2022). A new method for detecting onset and offset for singing in real-time and offline environments. Applied Sciences, 12(15), 7391.

De Cheveigné, A., & Kawahara, H. (2002). YIN, a fundamental frequency estimator for speech and music. The Journal of the Acoustical Society of America, 111(4), 1917-1930.

Mounir, M., Karsmakers, P., & van Waterschoot, T. (2021). Musical note onset detection based on a spectral sparsity measure. EURASIP Journal on Audio, Speech, and Music Processing, 2021, 1-17.

Nicholson, R. (2012, April 3). Musical pitch is not just FFT frequency. MusingPaw. http://www.musingpaw.com/2012/04/musical-pitch-is-not-just-fft-frequency.html

Grosche, P., Müller, M., & Kurth, F. (2010, March). Cyclic tempogram—a mid-level tempo representation for musicsignals. In 2010 IEEE International Conference on Acoustics, Speech and Signal Processing (pp. 5522-5525). IEEE.

Caro Repetto, R., & Serra, X. (2017). A Collection of music scores for corpus based jingju singing research.

Bittner, R. M., Pasalo, K., Bosch, J. J., Meseguer-Brocal, G., & Rubinstein, D. (2021). vocadito: A dataset of solo vocals with $ f_0 $, note, and lyric annotations. arXiv preprint arXiv:2110.05580.

Micheyl, C., Keebler, M. V., & Oxenham, A. J. (2010). Pitch perception for mixtures of spectrally overlapping harmonic complex tones. The Journal of the Acoustical Society of America, 128(1), 257-269.