PLAY WITH YOURSELF - SIMPLE MUSIC ACCOMPANIMENT 2017

Adar Guy
University of Victoria
adarguy@uvic.ca

Colin Malloy
University of Victoria
malloyc@uvic.ca

Hector Perez
University of Victoria
hectorp@uvic.ca

ABSTRACT

Automatic accompaniment consists of leveraging audio feature extraction techniques in order to produce musical accompaniment for an input audio signal. In this article, we will introduce an automatic music accompaniment system we developed, called "Play With Yourself". The system receives musical audio input recorded by the user and produces a suitable MIDI accompaniment based on features extracted from the audio input. The features extracted include tempo, onset times, and harmony. The output MIDI file can be used with virtual instruments in a digital audio workstation (DAW) to create a harmonic and rhythmic accompaniment for the original audio file. Play With Yourself is not a real time tool, but is still advantageous for musicians that have previously recorded solo music and want to add texture to it in order to create a demo.

1. INTRODUCTION

Content based audio feature extraction frameworks attempt to compute a numerical representation that can be used to characterize a segment of audio. This process begins with the detection of a signal's segment boundaries, followed by the classification of each segment into a corresponding class (silence, music, non-music). Classification can either be step-by-step (each feature is used individually in different classification steps in order of computational complexity and differentiating power), or more effectively, feature vector based audio classification which uses a set of features together as a vector to calculate the closeness of the input to the training sets. Common audio features of interest are global and instantaneous temporal features, energy features, spectral shape features, harmonic features and perceptual features. Common pre-computations of the extraction model attempts to provide adequate signal representations for latter processing of descriptor extraction. This pre-computational process primarily concerns the estimation of the energy envelope of a signal, the computation of the short time Fourier transform (STFT), and the sinusoidal harmonic modeling of the signal. Play With Yourself makes use of these processes to extract audio features that are necessary for creating an automatic accompaniment.

Determining global and instantaneous temporal features are essential in order to begin the the accompaniment process. The features associated with this task are beats-per-minute (BPM), onset, and beat detection. When combined appropriately, these features can create an accurate 'time-map' of the associated audio signal that includes temporal variations for the length of the signal. An accurate 'time-map' can be used to create a drum pattern that follows the tempo and rhythm of the corresponding signal and will accommodate for natural fluctuations. However, in order to produce a harmonic accompaniment, more features must be extracted that will help determine single/multiple (chord) pitches so the system can predict and produce harmonically appropriate sounds.

An accompaniment system would make use of these features in tandem to create a timeline of harmonic transient events that correspond to a musical accompaniment for the audio signal. This could include percussive drum patterns as previously mentioned, a melodic accompaniment such as a bass guitar, and/or a harmonic accompaniment such as piano or guitar. The extraction and classification of audio features in this way would compose the first stage of the Play With Yourself Accompaniment tool.

The second stage in creating an accompaniment is generative. After determining the audio features for a given signal, the program organizes the information in a MIDI readable format. Some of this information contains user defined settings for MIDI instrument choices, onset threshold (busyness), dynamic threshold, time signature, percussive patterns, and chord style. User defined parameters such as the beat correction window size and onset detection threshold will affect the behavior of the accompaniment program during processing. Play With Yourself provides a preview of every accompaniment generated and it is encouraged that these user settings be adjusted to achieve the most desired accompaniment to the user. Play With Yourself attempts to create a reasonable rhythmic and harmonic accompaniment for an input audio signal. As output the system generates a MIDI file which contains the accompaniment, and can be subsequently be used in a DAW.

2. MOTIVATION

Automatic accompaniment generation would be useful for musicians who want to quickly create demos of their music and do not have access to additional musicians or a studio for recording. Play With Yourself will allow musicians to

quickly and efficiently make demo tracks from recordings. This software is intended to be used by average musicians in tandem with DAW software but can be used as a standalone application. The goal of our system is to create a reasonable rhythmic and harmonic accompaniment for a pre-recorded input audio signal. The system generates a MIDI file which contains the accompaniment, and can be subsequently be used in a DAW. This project aims to create a useful tool for amateur musicians to create demos.

3. METHOD AND RESOURCES

As stated, Play With Yourself consists of two main processes that segment the computation into two stages.

3.1 Stage One: Detection and Feature Extraction

As stated, Play With Yourself consists of two main processes that segment the computation into two stages. The first stage of the system is detection and feature extraction. In this stage, pre-computational processes to determine the STFT and the energy envelope are required in order to calculate global descriptors such as log-time attack (logarithm of the time duration between the start and end of the stable portion of a signal), and the temporal centroid (typically achieved by low-pass filtering the analytical signal amplitude). An efficient implementation relies on the computation of the instantaneous RMS values of the local signal, and an appropriate window size (20-60 ms) for the given time frame (obtained from the FFT of the corresponding signal frame). The system then finds the onset times from the input signal and determines which of those onsets are beats. The user sets a threshold that a beat must surpass in order for the system to identify it. Identified beats are used for determining metric information for the audio input. The system also estimates the overall tempo for the audio signal.

Next, the system performs harmonic estimation to detect the chords present in the signal. In order to accomplish this task, the sinusoidal harmonic model must be computed which involves estimating the peaks of the STFT of the windowed signal segment. Peaks close to multiples of the fundamental frequency of each frame are then chosen in order to estimate sinusoidal harmonic frequency and amplitude. The system uses the previously determined beat-times to compare with timings for changes in the spectral centroid (the barycenter of the spectrum), and the spectral spread (spread of the spectrum around its mean value). To assist in this task, the system determines the temporal variation of the spectrum which is computed from the normalized cross-correlation between two successive amplitude spectra, representing the amount of variation of the spectrum along time. The global spectral shape description in the form of mel frequency cepstral coefficients (Fourier Transform of the logarithm of the spectrum computed on mel-bands) represents the shape of the spectrum with very few coefficients. This proves particularly useful when taking into account the mid-frequencies of a particular signal in order to extract harmonic features and build an appropriate accompaniment.

3.2 State Two: Accompaniment Generation

The system then progresses to stage two: the generative stage. Here, the information obtained in the previous steps is put to use in generating the MIDI accompaniment. The accompaniment consists of three parts: melodic, harmonic, and percussive.

The melodic accompaniment is generated using the harmonic estimations and beat times. Currently the tonic of each chord is used to determine the melodic accompaniment pitches. Timing and durations for the notes are determined using the beat and onset information. The MIDI notes are timed to the onset times from the audio signal.

The harmonic accompaniment also uses the harmonic estimations, but instead of using just the tonic it uses the full harmonic estimation to create a chordal accompaniment. The durations of the chords are determined by the extracted beat information while ignoring the onset information. In this way the chords are played less frequently and chordal accompaniment onsets are determined mainly by chord changes.

The system then converts the note/chord names into their corresponding MIDI values (from a preset list) for the melodic and harmonic accompaniments. For the percussive accompaniment, MIDI values (from a preset list of patterns) are sequentially placed in an array.

The timing of the percussive accompaniment is based on the extracted tempo. The system estimates where the next beat should be based on the tempo and then adjusts the timing based on onsets detected within a window from the expected placement.

The last step is to create the output MIDI files. Lists corresponding to a melodic MIDI value sequence, a harmonic MIDI value sequence, and a rhythmic MIDI value sequence are used along with the start/end times for each accompaniment to create three MIDI tracks - one for each accompaniment type. The note volumes for each track are based on the relative strengths of the onset peaks of the audio signal which are then scaled by a user input amount. MIDI program numbers for the accompaniment instruments are also user defined.

Various resources were used for this project. The source code is written in the Python programming language because Python has many powerful tools and packages for signal processing and information retrieval. The LibROSA package was used for beat and tempo extraction. The PyMIR library was used for chord extraction. Lastly, the MIDIUtil library was used for the creation of multi-track MIDI files within the program. Many modifications were made to these packages, however, in order to make them work together and increase accuracy in the Play With Yourself system.

4. IMPLEMENTATION OVERVIEW

The operation of the Play With Yourself system can be summarized as follows.

4.1 Beat Detection

LibROSA's 'beat_track' method is used to detect onsets and beats. This method works by first detecting the strength of different onsets. The tempo is estimated using onset correlation second. Lastly, to select the beats, onsets are compared against a strength threshold and the beat timing predicted by the estimated tempo. If the onset surpasses the threshold and is within an acceptable range from the estimated beat, then it is selected as a beat. The times of the beats, in seconds and milliseconds, are computed with the 'frames_to_time' method.

4.2 Chroma Vector Extraction

In Pymir, the 'chroma' method is used to compute chroma vectors associated with frames of the audio signal. The frequency spectrum of a given frame is calculated and then the frequencies are converted into MIDI numbers and folded into twelve pitch classes (using the modulo operation). The vectors are then normalized and divided by the maximum value making all the values in the vector between one and zero.

4.3 Chord Prediction

Pymir's 'getChord' function uses the chroma vectors to predict the chord for a specific frame. It compares the chroma vector from the previous step with 24 vectors (representing the 24 major and minor chords possible in the equal temperament tuning system) via cosine similarity. The vector that obtains maximum similarity is chosen and the function returns the chord with which it corresponds.

4.4 MIDI File Creation

Finally, the MIDIUtil library is used to create a MIDI file. The 'MIDIFile', 'addNote', and 'writeFile' functions are used. The 'MIDIFile' method initializes an object that can later be output as a MIDI file. The 'addNote' method writes note event information into tracks of the MIDIFile object. This method requires information about the event, such as MIDI note number, start time, duration, and volume. The 'writeFile' method converts the object to a MIDI file and write it to disk. In the Play With Yourself context the output MIDI file contains the generated melodic (single line), harmonic (chord progression), and rhythmic accompaniments.

5. PROBLEMS ADDRESSED

Many problems became evident over the course of testing the system with different inputs. Many modifications and refinements were made to the libraries used as well as the Play With Yourself code.

5.1 LibROSA Beat Detection and Tempo Estimation Problems

LibROSA's beat detection method did not always work correctly. It was especially inconsistent with more complex inputs. Specifically, there were usually some beats at the end of a signal that were completely missed by the system. Also, some beats selected by LibROSA deviated significantly from the true locations of the onsets. These beats were selected because they were consistent with the tempo. In some instances LibROSA incorrectly selected a time between two onsets as the beat. This caused problems with the accompaniment, since it was desynchronized from the onsets in the input audio signal.

This was solved by implementing a new method as a substitution for libROSA's beat selection method. Essentially, this method checks a 'window' near the location in time where libROSA detected a beat. If an onset is found within that window, then the timing of the beat is mapped to the onset. This helps the timing of the accompaniment coincide better with the timing of the audio signal.

A secondary problem existed for tempo estimation. LibROSA's method could potentially detect the tempo as a multiple of the true tempo. This ambiguity is difficult to resolve. Thus it was decided that the user will have control over halving or doubling the estimated tempo in order to halve or double the accompaniment speed.

5.2 Pymir Chord Estimation Problems

Pymir's chord detection fared well in most cases. Sometimes, however, a chord was incorrectly detected as the minor version instead of the major or vice versa. This problem has not yet been addressed but a possibility is to implement a Hidden Markov chain that would help the system discern transition probabilities between chords.

5.3 MIDIUtil Timing Problems

Another problem arose when using the MIDIUtil library to write note events with 'addNote'. The problem was that the times used to write notes are not absolute. This means that it was not possible to simply specify the point in time where the event should be placed.

MIDIUtil expects to receive relative times instead, where the start of one note event depends on the ending of another. The solution was to implement a method of automating a process of converting relative times to absolute times.

6. SYSTEM TESTING

Testing was primarily carried out by using guitar patterns recorded by a group member. These patterns were in 4/4 time signature and various keys. They varied in tempo and complexity in order to test the different potential sources of inaccuracy in the system.

Due to the specialized nature of the audio input, it was necessary to record test tracks since any existing data sets would not fit the situation correctly. The benefit of creating test audio files ourselves is that we know the chord

progressions, rhythms, and have a single, isolated instrument. Most patterns in 4/4 that were tested were correct. Inaccuracies in the generated accompaniment usually occurred in the chord estimation.

7. CURRENT STATE OF THE PROJECT

Play With Yourself successfully detects beats and chords, generates accompaniments, and produces a MIDI output file. It is designed with work with input signals that are in 4/4 time, not overly rhythmically complex, and can only detect major and minor chords. The system has recently been extended to work with other time signatures, but is untested in this area.

The output MIDI files are easily imported into a DAW. The patterns created are simple since the system places notes generated on the first downbeat of every bar. The system does not yet use pre-made patterns for the harmonic or melodic accompaniments. It generates a unique accompaniment based on the harmony and timing extracted from the input audio. The percussive accompaniment is selected from a list of pre-composed drum patterns.

A recent addition to the project has been an extension to the harmonic accompaniment. There are now several options to generate more varied chordal accompaniments. These variations include different chord inversions and open/closed chord forms. Previously, only one set of root-position chords were available.

The program is run from the command line, but a browser-based GUI has been created that can interact with the program. The third party library 'Flask' is used to connect the python code to an interface that works in a browser window. The interface contains sliders, and value input boxes, where the user can specify their preference. This facilitates a smoother workflow for the user, who can understand visually what parameters are affecting how the program runs. This GUI is not to be confused with the proposed DAW plugin mentioned later in this article. The DAW version will not run from the command line, but instead run inside of a DAW.

The parameters that the user can change regarding beat detection are: onset strength threshold and window size (for detecting whether an estimated beat is closed to a beat consistent with the tempo).

Other parameters the user can change are: dynamic threshold (for the volumes of MIDI notes), instruments to be used in the accompaniment, and the beat pattern to be used.

The user also now has the ability to audition the accompaniment as an audio signal of the accompaniment and original audio signal before outputting the MIDI file accompaniment. If the user is not pleased with the generated accompaniment, the user can adjust the various parameters until a more suitable accompaniment is generated.

8. FUTURE PLANS

The baseline goal for the project has been achieved. This entailed creating a simple harmonic and rhythmic accompaniment for audio in 4/4 time as a MIDI file that can be imported into a DAW. However, many improvements can still be made to the system.

The program would ideally support inputs with more complex rhythmic and harmonic patterns. This includes time signatures besides 4/4 and chords tonalities beyond major and minor.

The accompaniments generated can be made more musically appropriate through the use of compositional rules based on various musical styles. More complex drum patterns can be included. These would require extending the system to compute more subdivisions of the beat than it currently does, but would allow for much more flexibility and realism in accompaniment.

In the last stages of the project, some improvements have been included, but require fine tuning and further extension. The first improvement is a browser based graphical user interface for the system. The polyphonic accompaniment system has also been extended by creating more chord lists to pull from. Ultimately it would be ideal to move away from pre-made chord lists and include stylistic rules for selecting chords, voicings, and percussive patterns.

It would also be ideal for the program to run as a plugin in a DAW instead of being command-line based with a browser GUI. The general plan for doing this is to port the system to the JUCE development environment which is designed specifically for developing audio plugins. This would require an entire rewrite of the program, however, since JUCE runs on c++ and not Python. In section 9 we elaborate on how Play With Yourself would potentially operate within a DAW. We also include a mockup of a potential GUI design.

9. PLAY WITH YOURSELF IN A DAW

Play With Yourself would work well in a DAW environment but such an implementation is outside the scope of the current project. As an ideally realized plugin version, Play With Yourself would be a plugin for a virtual instrument track that includes an analysis component and an optional sample player component. The MIDI track generated by Play With Yourself could also be exported from the plugin and used with another virtual instrument to allow more flexible sound options.

Figure 1 shows a visual representation and functional description of how the plugin might operate in a DAW.

The user would select a portion of an audio track, have Play With Yourself analyze it, and generate an accompaniment pattern. The accompaniment could then be customized by modifying the parameters of the program, such as the style, busyness level, instrument sounds, etc. Just as the user and audition the accompaniment in the current version of the software, the mockup includes a 'Preview' button which would allow the user to audition the accompaniment before exporting it to a MIDI track.

The following is a description of the projected user selectable parameters:

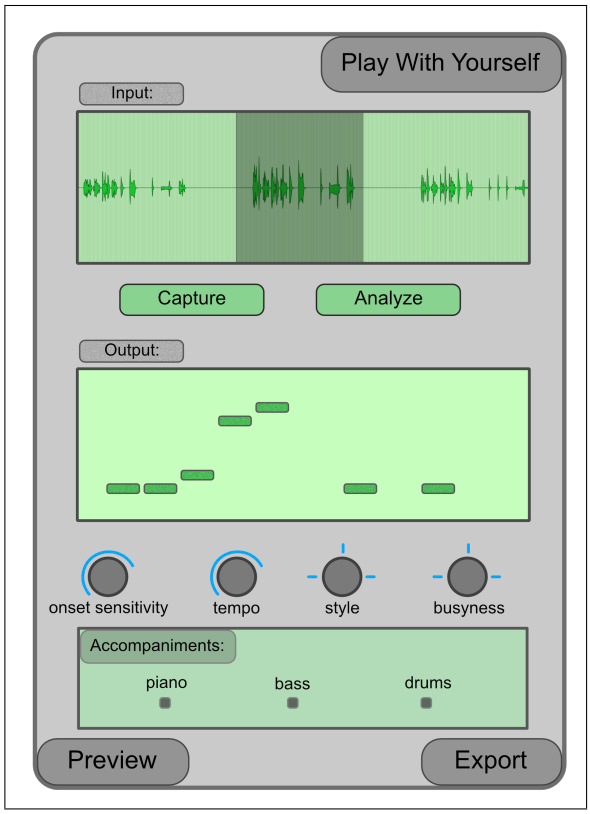

Figure 1. Top: waveform selected for analysis. Middle: Generated MIDI accompaniment. Bottom: User selectable parameters

1. Onset sensitivity: defines a threshold that the onset strength must surpass in order to be detected.
2. Tempo: allows the user to halve or double the tempo.
3. Busyness: defines the complexity of the generated accompaniment.

A plugin implementation should be compatible with the different plugin architectures such as VST, AAX, and AU. This is important for allowing the program to run in all of the major DAW software environments (such as Pro Tools, Logic, Live, etc).

An ideal version of the plugin would also include a robust sample playback engine so that the user could design the accompaniment instruments from with the plugin itself. This would streamline the experience as well as provide a method for creating a marketplace around the plugin. Users could potentially purchase the base version of the plugin which does the analysis and accompaniment generation, but then could make further purchases to augment the functionality of the plugin and expand the sample libraries available.

10. LITERATURE AND PRIOR WORK

Chroma vectors are commonly used for key estimation [4, 9]. In chroma vector analysis, a high resolution Fourier transform is performed on a signal and the resultant frequency spectrum is 'folded' into a 12 pitch histogram. This allows the identification of different chords. PyMIR allowed us to use a similar implementation in our project.

Solutions for rhythm estimation are quite varied. However, it is common to use onset envelopes for this process. Following are some proposed implementations for this end.

Goto and Muraoka developed real-time beat tracking systems that work with raw audio signals which contain drums [6, 13-14]. These systems create multiple hypothesis using complex probabilistic calculations, and determine the most suitable one at any given time.

Algonhiemy and Tewfik proposed the use of binary and ternary (trellis) trees to determine patterns in micro and macro scales within the audio signal [11]. They had to consider two cases: when the input audio included long pauses and when it didn't. Their approach was based on the hypothesis that for most music the beats are clearest in the low frequencies. For this reason, classical music would not work well with this system.

Timing nets (a form of neural network) are used in a self-adjusting beat detector by R. Harper and Ed Jernigan [19]. The nodes in the network represent hypothesis of where the next beat will occur. Each nodes contains a 'spike train.' The node that contains the spike train which is most similar to the estimated onset curve is chosen as the current state of the system.

Rosenthal worked on an MIT research project called 'Machine Rhythm' which performs rhythm detection on polyphonic MIDI input [5]. He also proposed a hierarchical structure, which contains different 'levels' of beats. Each level corresponds to the onset envelope extracted for a different frequency band [11].

E. D. Scheirer developed a tempo and beat detection system. He started by separating the input into different frequency bands, and extracting the respective onset envelopes [7]. He then inputs each envelope into a bank of resonators, and if there is a matching frequency between the envelope and a resonator, maximum output will be achieved [11].

Peter Desain and Henkjan Honing developed a method that used expectancy curves that are generated for equal time length sections of the input [18]. The desired result was an overall expectancy curve with local maxima indicating onsets of beats. This was obtained by multiplying all the shorter expectancy curves.

11. CONCLUSION

We have presented the 'Play With Yourself' automatic accompaniment system in this article. We took an offline (non-real time) approach for the project. The main limitation of this is that the system must use pre-recorded audio.

The project focused on creating an accurate accompaniment for simple audio inputs. This allowed for making the system more robust thorough testing. However, this is still a lot of potential for further extension of the system.

There are many different approaches available for detecting the rhythmic and harmonic contents of an audio

signal. We implemented algorithms that were not complicated, but they still yielded acceptable results.

The system could be helpful for musicians with limited resource that wish to quickly improve their music by adding other instruments.

There are many potential improvements to be made to the system. The main improvements include porting the system to a DAW plugin environment, creating a graphical user interface, and supporting more complex input signals.

12. REFERENCES

[1] A. Driesse. "Real-time Tempo Tracking Using Rules to Analyze Rhythmic Qualities," *Proceedings of the 1991 International Computer Music Conference*, pp. 578–581, 1991.

[2] A. Shenoy, and Y. Yang. "Key, Chord, and Rhythm Tracking of Popular Music Recordings," *Computer Music Journal*, Vol. 29, No. 3, pp. 75–86, 2006.

[3] A.T. Cemgil. "Polyphonic Pitch Identification and Bayesian Inference," *Proceedings of the 1991 International Computer Music Conference*, 1991.

[4] C. Harte, and M. Sandler. "Chromagram: Automatic Chord Identification Using a Quantised Chromagram," *Proceedings of the AES 118th Convention*, 2005.

[5] D. Rosenthal. *Machine Rhythm: Computer Emulation of Human Rhythm Perception*, PhD dissertation, Massachussetts Institute of Technology, 2012.

[6] D. Rosenthal, M. Goto, and Y. Muraoka. "Rhythm Tracking Using Multiple Hypotheses," *Proceedings of the 1994 International Computer Music Conference*, pp. 85–85, 1994.

[7] E. Scheirer. "Tempo and Beat Analysis of acoustic Musical Signals," *Journal of the Acoustic Society of America*, Vol. 103, No. 1, pp. 588–601, 1998.

[8] G. Loy. "Musicians Make a Standard: The MIDI Phenomenon," *Computer Music Journal*, Vol. 9, No. 4, pp. 8–26, 1985.

[9] G. Peeters. "Chromagram: Automatic Chord Identification Using a Quantised Chromagram," *Proceedings of the International Symposium on Music Information Retrieval*, pp. 115–120, 2006.

[10] K. Abdullah-Al-Mamun, F. Sarker, and G. Muhammad. "A High Resolution Pitch Detection Algorithm Based on AMDF and ACF," *Journal of Scientific Research*, Vol. 1, No. 3, pp 508–515, 2009.

[11] M. Alghoniemy, and A.H. Tewfik. "Rhythm and Periodicity Detection in Polyphonic Music," *IEEE 3rd Workshop on Multimedia Signal Processing*, pp. 185–190, 1999.

[12] M. Davy, and S.J. Godsill. "Bayesian Harmonic Models for Musical Signal Analysis," *Bayesian Statistics*, Vol. 7, pp 105-124, 2003.

[13] M. Goto. "An Audio-based Real-time Beat Tracking System for Music With or Without Drum-sounds," *Journal of New Music Research*, Vol. 30, No. 2, pp. 159–171, 2001.

[14] M. Goto and Y. Muraoka. "Real-time Beat Tracking for Drumless Audio Signals: Chord Change Detection for Musical Decisions," *Speech Communication*, Vol. 27, No. 3, pp. 311–335, 1999.

[15] N. Griffith. *Modelling the Acquisition and Represention of Musical Tonality as a Function of Pitch-use Through Self-organizing Artificial Neural Networks*, PhD dissertation, University of Exeter, 1993.

[16] P.E. Allen, and R.B. Dannenberg. "Tracking Musical Beats in Real Time," *Proceedings of the 1990 International Computer Music Conference*, pp. 140–143, 1990.

[17] P. Cariani. "Neural Timing Nets," *Neural Networks*, Vol. 14, No. 6, pp. 737–753, 2001.

[18] P. Desain and H. Honing. "Advanced Issues in Beat Induction Modeling: Syncopation, Tempo and Timing," *Proceedings of the 1994 International Computer Music Conference*, Vol. 20, pp. 92–94, 1994.

[19] M. Jernigan and R. Harper. "Self-adjusting Beat Detection and Prediction in Music," *IEEE Internation Conference on Acoustics, Speech, and Signal Processing, 2004 Proceedings*, Vol. 4, pp. iv, 2004.

[20] S.A. Raczynski, T. Nishimoto, N. Ono, S. Sagayama, E. Vincent, and J. Wu. "Multipitch Estimation by Joint Modeling of Harmonic and Transient Sounds," *2011 IEEE Internation Conference on Acoustics, Speech, and Signal Processing, 2011 Proceedings*, pp. 25–28, 2011.