POST-PROCESSING FIDDLE∼ : A REAL-TIME MULTI-PITCH TRACKING TECHNIQUE USING HARMONIC PARTIAL SUBTRACTION FOR USE WITHIN LIVE PERFORMANCE SYSTEMS

Andrew N. Robertson, Mark D. Plumbley

Centre for Digital Music
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, England
andrew.robertson@elec.qmul.ac.uk

ABSTRACT

We present a method for real-time pitch-tracking which generates an estimation of the relative amplitudes of the partials relative to the fundamental for each detected note. We then employ a subtraction method, whereby lower fundamentals in the spectrum are accounted for when looking at higher fundamental notes. By tracking notes which are playing, we look for note off events and continually update our expected partial weightings for each note. The resulting algorithm makes use of these relative partial weightings within its decision process. We have evaluated the system against a data set and compared it with specialised offline pitch-trackers.

1. INTRODUCTION

Polyphonic or multiple pitch-tracking is a difficult problem in signal processing. Most existing work in multi-pitch tracking is designed for Music Information Retrieval which takes place offline on large data sets. A method for multiple frequency estimation by the summing of partial amplitudes within the frequency domain was presented by Klapuri [5], who makes use of an iterative procedure to subsequently subtract partials within a pitch detection algorithm. Pertusa and Inesta [7] list potential fundamental frequency candidates in order of the sum of their harmonic amplitudes.

Existing real-time algorithms for pitch detection include *fiddle∼*, a Max/MSP object by Puckette et al. [8] based on a Fourier transform which employs peak picking. Jehan [4] adapted the algorithm to analyse timbral qualities of a signal. In the time domain, de Cheveigné and Kawahara's Yin [2] is a widely-known algorithm which uses auto-correlation on the time-domain signal to calculate the most prominent frequency. However, these algorithms are more suited to monophonic signals and they are not reliable enough to generate a MIDI transcription of audio from a polyphonic instrument.

We proceed from the observation that any given pitch will also create peaks at frequencies corresponding to its partials. In our approach, we iteratively subtract partials within the frequency domain in order to aid a real-time pitch detector. A learning method is employed to optimise the expected amplitudes of the partials of each detected note by continually updating the weights whenever a note is detected. In addition, we model the variations within the amplitude and summed partial amplitudes of detected notes. The weightings for each partial derived from observations are used within the decision-making process.

Our motivation for this method is for use within live performance, to generate information about new notes played by an instrument. This can then be used to provide accompaniment, either directly, or by aligning the information with an expected part. Previous research into pitch tracking for interactive music has highlighted the importance of minimal latency and accuracy within noisy conditions [3]. Since our algorithm is employed for real-time audio-to-MIDI conversion within a performance system, we require a fast detection of notes and fast computation time.

2. METHOD

2.1. Implementation and Pre-Processing

Our algorithm has been implemented in Java within a Max/MSP patch and in doing so, we made use of the *fiddle∼* object [8] in the pre-processing stage. Ordinarily, *fiddle∼* provides its own fundamental frequency estimation, but it also gives the 'uncooked' data of the top N frequencies from the peak picking process and their respective amplitudes above a suitable threshold. Since *fiddle∼* has been optimised for fast processing within a real-time environment, it is well-suited to providing an efficient FFT and noise reduction process used to provide the data for our partial-removal system. We use a frame of 2048 samples with a hop-size of 1024, so that our detection of notes is a fast as possible

whilst still detecting as low as 80Hz.

2.2. Update the Amplitudes

The input to the algorithm is the list of top N frequencies and their amplitudes (typically 8 to 20) from *fiddle*∼. The algorithm continually tracks the amplitude of all MIDI notes. First we calculate the corresponding MIDI notes to the incoming peak frequencies and update their respective amplitudes. The amplitudes of all MIDI notes not present in the peak frequencies list are decreased by 20% for each input frame (every 23 ms). This allows for errors if notes are accidentally skipped in this 'top 10' procedure. It is quite common in the duration of a note, for one of the peak frequencies to shift to an adjacent note in a frame and this prevents original amplitude dropping to zero and triggering a note off.

2.3. Track New and Existing Notes

We begin with the lowest note and work up the range of frequencies. For every note present in the incoming peak frequencies list, potentially a new note-on, and every note already playing, potentially a note-off event, we calculate the 'power' of the note, $P(m)$, by summing the product of the amplitude of the respective partials with our weighting matrix, $W_m(k)$ for that note. This is given by

$$P(m) = \sum_{k=1}^{L} W_m(k) A(m + h[k]) \quad (1)$$

where A(m) is the amplitude of MIDI note m, L is the number of partials summed (we chose $L = 6$), k is the partial number (the note's frequency as an integer multiple of the fundamental frequency), $h[k]$ is the interval in semitones between frequencies f_0 and kf_0, and $W_m(k)$ is the weight vector derived from the observed signal, of the amplitude of the k^{th} partial relative to the amplitude of the fundamental, specific for each individual note in the spectrum.

2.4. Notes On and Notes Off

For currently playing notes, we look for a note-off event:

If $P(m) < \theta_{-}.\bar{P}(m)$,
then output a MIDI note-off for pitch m,

where θ_{-} is a threshold and $\bar{P}(m)$ is an estimate of the median of the power in a positively detected note. Figure 1 shows how this quantity varies over the range of notes.

For non-playing notes, we calculate the change in power as a ratio between the current frame and the previous frame.

$$r(m) = \frac{P_t(m)}{P_{t-1}(m)} \quad (2)$$

We check that the MIDI note has at least one partial note, $m + h[k]$, that is one of the top N peaks, such that $k <= 4$

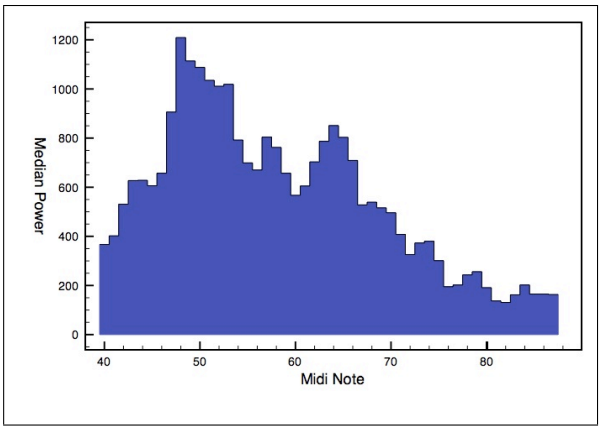

Figure 1. Median power over the range of piano notes. The power of played notes varies dramatically with pitch so that learning the median value for triggering plays an important role.

and if the only partial present is $k = 3$ (19 semitones) then $(m+7)$, the fifth, is not also a peak. Then:

If $P(m) > \theta_{+}.\bar{P}(m)$ and $r(m) > \theta_r$ and $A(m) > \theta_a.\bar{A}(m)$,
output a MIDI note-on for pitch m,

where θ_{+}, θ_a and θ_r are thresholds for power, amplitude and ratio, $r(m)$, respectively.

This ensures a significant measure of summed harmonic amplitudes and a significant increase in this measure since the last observed frame. In practice, values for the ratio threshold, θ_r, tend to be between 1.4 and 3, depending on the level of response required. The higher the ratio, the less likely the algorithm is to trigger a false positive.

In the case of a note on or if the note on was within the last three frames, we adapt our weights $W(p_n)$. Our current observation would suggest:

$$W^*(k) = \frac{A(m + h[k])}{A(m)} \quad (3)$$

We track how many observations have been made in the past and adapt so $W(p_n)$ is the average of these and the new observation $W^*(p_n)$. We perform this update for all notes within 5 semitones of the played note since some notes are played less frequently, yet we can reasonably assume that the tone and timbre with respect to partial weightings is approximately the same as the surrounding notes. By including notes close to our observed note, we adapt the weights more quickly to a useful approximation.

We also update our estimate for the median of the amplitude and power of a note out at that MIDI pitch using an exponential moving average:

$$\bar{A}(m) = (1 - \alpha).\bar{A}(m) + \alpha.A(m) \quad (4)$$

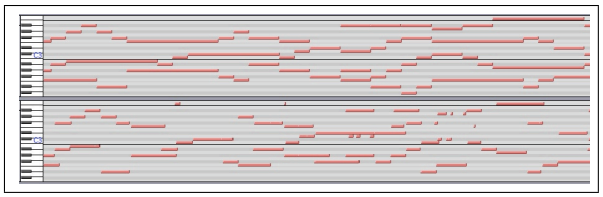

Figure 2. Ground-truth MIDI from Bach's 'Well Tempered Clavier' (top) and the MIDI output from the corresponding synthesized audio as input to the pitch-tracker (bottom).

where α (typically 0.2) defines the response of the median estimate to new data.

2.5. Partial Subtraction

Having evaluated the current note's strength, if the note is either playing or a new note, then we subtract from the amplitudes of its partials higher in the frequency range. High frequencies will have considerable amplitude due to this lower fundamental, so the subtraction process helps to prevent false positives from partials. Hence, we use the following update rule:

$$A(m+h[k]) = A(m+h[k]) - W_m(k).A(m) \qquad (5)$$

for $1 \leq k \leq L$. We aim to optimise these weights by introducing some feedback at this stage. If the subtraction process results in $A(k)$ becoming less than zero, then we decrease $W_m(k)$. If it is greater than zero then we increase the weight. Hence, all playing notes function to the optimise the average weighting of their respective pitch class (the note and surrounding notes).

There is an assumption here that for the majority of notes the instrument is *relatively monophonic*. The weights are adjusted on the basis that if a fundamental is playing, the partial is not also playing as part of a polyphonic chord. Whilst this may not be strictly true (as when an octave plays), it is true for the most part, so that when an octave does play, the residual power in the first partial after the subtraction process should still be substantial enough to trigger the recognition of the octave note.

We have used the algorithm in live performances on an acoustic guitar, using it to create a texture of synthesized sounds behind the guitar. By filtering notes to an appropriate scale, we can help to avoid dissonance from false detections.

Experimentation with a MIDI-triggered electric piano sound suggests that there is a detection latency between 60 and 90 msec. This is still quite considerable for use within a live context when fast passages are played. By comparison, Miller Puckette's *bonk*$\sim$ onset detector has a latency of approximately 10 to 30 msec for the same notes. However, the onset detector is able to make use of a frame-size of 256 samples (or 5.8 msec), whereas for the Fourier analysis involved in adequate pitch detection, we require a frame-size of 2048 samples (or 46 msec).

Piece	Correct (%)	False Positive	Number of notes
WTC1f	80.0	31.3	1075
WTC1p	71.0	39.3	833
WTC2f	76.4	36.8	647
WTC2p	78.4	27.3	1408
WTC8f	79.0	41.0	1014

Table 1. Detection Rates against synthesized harpsichord audio from Bach's 'Well-Tempered Clavier'.

3. EVALUATION

When used within performance, this provides good subjective results. To our knowledge, there is no existing Max/MSP polyphonic real-time object available for direct comparison. The *fiddle*$\sim$ and *yin*$\sim$ objects are monophonic and were not designed for polyphonic pitch detection, and their use in this context gives subjectively poor results in comparison. We would like to provide an objective measure of success within a performance application. However, we can so far only compare with offline systems.

On this, we tested the tracker on several synthesized harpsichord recordings of Bach's 'Well-Tempered Clavier'. By sending MIDI files to a Yamaha Stage Piano and testing the pitch-tracker on the corresponding synthesized audio, we can simulate the task of audio-to-MIDI conversion for a polyphonic instrument, whilst having ground-truth of the notes actually triggered.

A representation of the MIDI ground truth and the corresponding output from the detector is shown in Figure 2. The results are shown in Table 1 and the average latency measured between 70 and 90ms. The percussive, distinctive nature of the harpsichord sound seems to be an optimal input for the pitch-tracker resulting in high performance statistics of approximately 80% correct detections. The precision currently appears to comparable with some offline trackers. The MIREX 2007 [1] competition results rate offline trackers with a precision of between approximately 40 and 70%. The equivalent precision for our real-time pitch-tracker here would be over 50%, but it is important to note that the MIREX competition uses a wide database of varied sounds and hence the result on the Bach pieces may be artificially high.

Marolt [6] has developed an offline pitch-tracker, Sonic, specialised for piano input which uses adaptive oscillators and neural networks. We also tested the tracker on his data set using a selection of three synthesized audio samples and three samples from performances with a real piano.
The synthesized pieces were: J. S. Bach, Partita no. 4,

Piece	Correct (%)	False Positive	Number of notes
Synthetic			
Partita no.4	46.0	38.0	496
Humoresque	42.1	49.6	545
Sonata no.15	61.1	22.4	651
Real			
Suite no.5	50.1	43.7	652
Nocturne no.2	38.1	37.6	252
Entertainer	45.1	42.7	567

Table 2. Detection Rates against the piano data set used to test Sonic.

BWV828, ; A. Dvorak, Humouresque Op. 101, no. 7, ; W. A. Mozart, Sonata no. 15 in C major, K. 545, 3. mvm. The real pieces were: J. S. Bach, English Suite no. 5, BWV810; F. Chopin, Nocturne Op. 9 no. 2; S. Joplin, The Entertainer.

Sonic obtains a success rate of approximately 90 % on this data set, with a false detection rate of approximately 9%, whereas our real-time tracker only succeeded in detecting between 40 and 50% of the notes with considerably less precision (approximately 40% false positives). A large proportion of the false positive rate is due to high frequency content from harmonics present within the original signal, which are more tolerable in a live context than inharmonic and lower frequency errors.

Our proposed system therefore gives offline figures that are significantly worse than specialised systems designed for polyphonic transcription tasks. However, our subjective observations are that it is very successful in a live performance context. This raises the issue of how we can perform an evaluation that fairly reflects success in such performances. In [9] we used subjective evaluation to assess the success of a real-time beat tracker, and this is one of the directions for future work.

In addition, preliminary analysis indicates that many of the errors made by the system are harmonics misidentified as notes. When used for creating a texture for live performance, these errors are less significant than random errors. This may go some way to explaining the large difference between objective error and high perceived success in the performance application, and it would be interesting to explore an error measure designed to distinguish between these different types of errors.

4. CONCLUSION

We present a system for real-time polyphonic pitch tracking for live performance applications, based on iterative subtraction of estimated partial amplitudes from the frequency representation. Our approach uses a fast deductive procedure based on the existence of partials for any given note. By continually updating estimates for the weight of the partials relative to the fundamental, the median values for the amplitude and power of all notes, our algorithm is capable of performing moderately well on databases designed for offline multiple pitch-tracking algorithms.

Although the algorithm does not perform as well in objective offline tests as other algorithms designed for offline use, our approach does give subjectively high success in a live performance application. In future work, we will further explore this evaluation issue, including investigating the relative importance of various types of misidentifications, and the use of subjective testing methodologies.

5. REFERENCES

[1] Music Information Retrieval Evaluation Exchange. [Online]. Available: http://www.musir-ir.org/mirex2007

[2] A. de Cheveigné and H. Kawahara, "Yin, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 111, pp. 1917–1930, 2002.

[3] P. de la Cuadra, A. Master, and C. Sapp, "Efficient pitch detection techniques for interactive music," in *Proc. International Computer Music Conference*, 2001, pp. 403–406.

[4] T. Jehan and B. Schoner, "An audio-driven perceptually meaningful timbre synthesizer," in *Proc. International Computer Music Conference*, 2001, pp. 381–388.

[5] A. Klapuri, "Multiple fundamental frequency estimation by harmonicity and spectral smoothness," *IEEE Trans. Speech and Audio Processing*, vol. 11, pp. 804–816, 2003.

[6] M. Marolt, "A connectionist model of finding partial groups in music recordings with application to music transcription," in *Proc. Int. Conf. on Adaptive and natural computing algorithms*, Ribiero *et al.*, Eds., 2005, pp. 494–497.

[7] A. Pertusa and J. M. Inesta, "Multiple fundamental frequency estimation using Gaussian smoothness," in *Proc. International Conference on Acoustics, Speech and Signal Processing*, 2008, pp. 105–108.

[8] M. Puckette, T. Apel, and D. Zicarelli, "Real-time audio analysis tools for Pd and MSP," in *Proc. International Computer Music Conference*, 1998, pp. 109–112.

[9] A. Robertson and M. D. Plumbley, "A Turing Test for B-Keeper: Evaluating a real-time beat tracker," in *Proc. International Conference on New Interfaces for Musical Expression (NIME)*, 2008, pp. 319–324.